Masterarbeit

Fakultät für
Informations-, Medien-
und Elektrotechnik

Technology
Arts Sciences
TH Köln

Energiesektorkopplung: Entwicklung von Optimierungsstrategien in Open-Source-Programmen

Benedikt Solf

Matrikelnummer: 11113913

Studiengang: Elektrotechnik, Studienrichtung Energietechnik

Erstgutachter: Prof. Dr. Eberhard Waffenschmidt

Zweitgutachterin: Prof. Dr. Beate Rhein

Abgabefrist: 24.04.2017

Ich erkläre an Eides statt, dass ich die vorgelegte Abschlussarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort, Datum, Unterschrift: _____

Kurzfassung

Diese Arbeit beschäftigt sich mit der Simulation und Optimierung von Energiesektorenkopplung. Dies geschieht mit der frei verfügbaren Programmbibliothek oemof, die in Python zur Programmierung von Beispielen genutzt wird. Es wird untersucht, inwiefern die Software im Rahmen des Forschungsprojekts ES-Flex-Infra an der TH Köln vergleichend zur Berechnung von sektorübergreifenden Optimierungsproblemen eingesetzt werden kann. Dazu werden die Möglichkeiten der Programmbibliothek oemof beschrieben, Probleme erläutert und mit selbst verfassten Programmcodes Belege für das Pro und Contra oemofs verfasst. Abschließend werden Parameter, nach denen ein Energiemodell optimiert werden kann, erläutert und die Umsetzbarkeit in oemof überprüft.

Abstract

This master thesis is about the simulation and optimization of interlinking of energy sectors. This will be done with the free access library program oemof, which is being used in python to program examples. It will be analyzed how the software, within in the framework of the research project ex-flex-infra at TH Köln, could be of use to comparative calculations of sector comprehensive optimization problems. Therefore, possibilities of the library program oemof will be described, problems discussed and pros and cons will be shown with self-composed program codes. In the end parameters, how an energy model could be optimized, will be explained and its feasibility in oemof will be tested.

Danksagung

Mein Dank gilt vor allem meinem Betreuer M.Sc. Christian Brosig, der mich während der gesamten Arbeit mit Informationen und Ratschlägen tatkräftig unterstützte. Durch den gemeinsamen Austausch während der Arbeit entstanden somit neue Ansätze und Ideen. Ebenso möchte ich mich bei meinem Erstgutachter Prof. Dr. Eberhard Waffenschmidt für die unkomplizierte Zusammenarbeit und die Möglichkeit im Rahmen dieser Arbeit die Simulation und Optimierung der Kopplungsstellen von Energiesektoren an der Technischen Hochschule Köln näher zu betrachten, bedanken.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichnis

1. Einleitung

1.1. Motivation

Die Arbeit entstand im Rahmen des Forschungsprojekts ES-Flex-Infra (Modellierung und Optimierung der Kopplung von Energiesektoren zur Flexibilisierung der Energieinfrastruktur). In dem Projekt geht es um den größer werdenden Bedarf an Energiespeicherung im Verlauf der Energiewende. Die Energiewende umfasst die Änderung der Nutzung von nicht nachhaltiger Energie in Form von Kohle, Gas und Kernkraft hin zu erneuerbaren Energien, die in Abhängigkeit der Sonnenstrahlung unbegrenzt nutzbar sind [1]. Die erneuerbaren Energien sind fluktuierende Energieerzeuger. Dies bedeutet, dass Wege gefunden werden müssen, um die fluktuierende Energieerzeugung den Lasten anzugleichen, da sonst die Lasten nicht ausreichend bedient werden können oder überflüssige Energie verloren geht. Hierfür bieten sich die Zwischenspeicherung von Energie, die Umwandlung zwischen Energieformen und die Kombination beider Verfahren an [2].

Um die Energiewende möglichst effizient und wirtschaftlich vertretbar zu gestalten, wird an der Verknüpfung der Energiesektoren (Strom, Gas und Fernwärme) geforscht. Damit sollen Synergien in Speicherungen und Lastflüssen genutzt werden. Ziel des Projekts ist es, sektor-verknüpfte Energiesysteme zu untersuchen und diese zu optimieren. Aufgrund der Energiewende und einer steigenden Anzahl erneuerbarer Energien, die eine fluktuierende Einspeisung mit sich bringen, gilt es mehr denn je, die Potenziale der einzelnen Energiesektoren miteinander zu verknüpfen, um eine effizientere Energieversorgung zu ermöglichen. Wind- und PV-Anlagen produzieren teilweise heute schon zu viel Energie, die nicht ausreichend genutzt oder gespeichert werden kann. Infolgedessen müssen die Anlagen zum Schutz der Stromnetze abgeschaltet werden. Mit einer Kopplung der Energienetze kann Energie effizienter genutzt werden, was ökonomisch und ökologisch von Bedeutung ist. Die Simulation und die Optimierung der Energiesektorenkopplung ist dafür der Anfang der Forschung.

Für das Forschungsprojekt wird eine open-source-basierte Software gesucht, die es ermöglicht, frei von Lizenzen eigene Berechnungen an der TH Köln durchzuführen und den kompletten Rechenweg nachzuvollziehen. Aufgrund der vielen Möglichkeiten des Programms wurde hierfür die Programmbibliothek oemof von M.Sc. Christian Brosig vorgeschlagen und in Absprache mit Prof. Dr. Waffenschmidt als Grundlage dieser Masterthesis festgelegt.

1.2. Ziel und Vorgehensweise der Arbeit

In dieser Arbeit wird die Programmbibliothek oemof daraufhin überprüft, ob und in welchem Umfang sie für die Lösung von Berechnungen der Energiesektorenkopplung innerhalb des Forschungsprojekts ES-Flex-Infra wie auch in zukünftigen Projekten an der TH Köln, die sich mit der Simulation der Energiesektorenverknüpfungen beschäftigen werden, anwendbar ist.

Um den Aufbau des Programms darzulegen, werden schrittweise zuerst die Darstellungsformen der verschiedenen Energieversorgungssysteme, energieeinspeisende Objekte, Verbraucher, Kopplungsstellen und Leitungen erläutert. Es wird außerdem analysiert, welche Vereinfachungen angenommen werden, und Auswirkungen derselben bestimmt.

Nach der Beschreibung des Programms folgen einzelne mit oemof aufgebaute Szenarien, um das Programm auf relevante Eigenschaften für das Forschungsprojekt zu überprüfen, wie die Darstellungsweise von Energiesystemen, die Berechnungsdauer einzelner Simulationen, die Optimierung des Standortes von Kopplungsstellen und die Simulation einzelner Kopplungsstellen. Diese geben einen Aufschluss darüber, ob und inwiefern die Programmierungsumgebung der Programmbibliothek oemof in Python für das Forschungsprojekt brauchbar ist.

Neben diesen Tests werden auch die Grundlagen des Programms und die Berechnungsweise näher erläutert. Im Anschluss werden Optimierungsansätze für die Berechnung der Energiekopplungsstellen aufgestellt und es wird überprüft, ob diese Ansätze mit oemof umgesetzt und welche Ansätze nicht ausreichend simuliert und berechnet werden können.

2. Hintergrund

2.1. Oemof

Der Name des Projektes oemof setzt sich aus den englischen Wörtern „open energy modelling framework“ („offener Energie-Modellierungsrahmen“) zusammen. Es ist ein „Werkzeug zur Modellierung und Analyse von Energiesystemen“ [3]. Es handelt sich hierbei um eine open-source-lizenzierte Software nach GPL Version 3. Die Software entstand am Reiner-Lemoine-Institut in Berlin.

Das Reiner-Lemoine-Institut forscht und entwickelt im Bereich der erneuerbaren Energien. Es hat sich zur Grundphilosophie gemacht, einen wesentlichen Beitrag zur Energieversorgung, die zu 100 % auf erneuerbaren Energien beruht, zu leisten, und die mit „gestalterischer sowie finanzieller Beteiligung aller Bürger entsteht“ [4]. Außerdem entwickelt und arbeitet das Institut an weiteren Projekten in und um den Bereich der Eingliederungen regenerativer Energien.

Oemof ist objektorientiert programmiert, in Python implementiert und greift auf schon vorhandene Bibliotheken zu. Eine der wichtigsten Bibliotheken hierbei ist SOLPH, die es möglich macht, lineare Probleme und gemischt-ganzzählige lineare Probleme (MILP), die innerhalb der Berechnung auftreten, zu lösen.

Der objektorientierte Aufbau mit der Einbindung vorhandener Bibliotheken ermöglicht es, frei einen Modellierungsansatz zu wählen und ein Energiesystem zu modellieren. Die Hauptanwendung von oemof ist die Berechnung von optimalen Energieflüssen in einem Energieversorgungssystem.

Aktuell (04.2017) gibt es von oemof die Version v0.1.1, auf der auch diese Arbeit beruht. Oemof ist im stetigen Wandel und wird weiterentwickelt.

Für oemof wird die Weiterentwicklung des Programms durch eine Gruppe freiwilliger Wissenschaftler und Interessierter unterstützt, die sich meist halbjährlich zu Kongressen treffen. Hier kann sich jeder auch freiwillig mit in das Projekt oemof einbringen. Zum Zweck der Weiterentwicklung ist es auf GitHub veröffentlicht [5].

2.2. Python

Für das Programmierumfeld nutzt oemof Python, das 1991 entwickelt worden ist und zu den höheren Programmiersprachen gehört, aber einen einfachen und gut lesbaren Programmierstil aufweist. Python unterstützt verschiedene Programmparadigmen. Neben der objektorientierten Programmierung, die oemof verwendet, werden auch die imperative, funktionale und aspektorientierte Programmierung unterstützt [6]. Wie auch oemof bietet Python ein open-source-gestütztes Netzwerk, die „Python Software Foundation“ [7]. Dieser Umstand und vor allem die einfach aufgebaute Syntax bieten eine gute Grundlage für oemof.

2.3. ES-Flex-Infra

Gefördert wird das Projekt aus Mitteln des Europäischen Fonds für regionale Entwicklung (EFRE) und durch das EFRE-Programm aus Nordrhein-Westfalen. Das Projekt steht unter der Leitung von Prof. Dr. Ingo Stadler und wird an der Technischen Hochschule Köln durch Prof. Dr. Eberhard Waffenschmidt und M.Sc. Christian Brosig aus dem Bereich der Energietechnik sowie von Prof. Dr. Rhein und Andreas Schwenk aus dem Bereich der Informatik unterstützt. Projektpartner sind die Rheinische NETZGesellschaft, das Fraunhofer Institut für Wissenschaftliches Rechnen und Algorithmen (SCAI) sowie werusys Industrieinformatik. Ziel ist die Modellierung und Optimierung von Energiesektorkopplungen zur Flexibilisierung der Energieinfrastruktur [8].

2.4. Der Open source - Gedanke

Die Grundidee, die auch oemof mit seinem open-source-gestützten Aufbau verfolgt, ist, eine Software für alle frei zugänglich zu machen und damit die Weiterentwicklung zu fördern. Ein weiteres wichtiges Element im Open-Source-Ansatz ist das Vertrauen der Anwender. Bei einer Kommerzialisierung eines Programms ist es schwer, in die zugrunde liegende Berechnungsweise zu blicken, was zu Vertrauensverlust zu dem Programm und seinen Ergebnissen führen kann [9]. Ist der Benutzer jedoch in der Lage, ein Programm komplett einzusehen, nachvollziehen und eventuell selbst daran mitwirken zu können, entsteht Vertrauen. Dieses Vertrauen ist vor allem bei einem sensiblen Thema, wie die Energiewende, wichtig.

3. Aufbau der Programmbibliothek oemof

Oemof ist objektorientiert aufgebaut und in die Bibliotheken solph und outputlib unterteilt. Die Bibliothek solph umfasst alle Bausteine, um ein Energiemodell aufzubauen. Diese werden für das Programm als ein gemischt ganzzahliges lineares Problem eingelesen. Folgende Bausteine beinhaltet es:

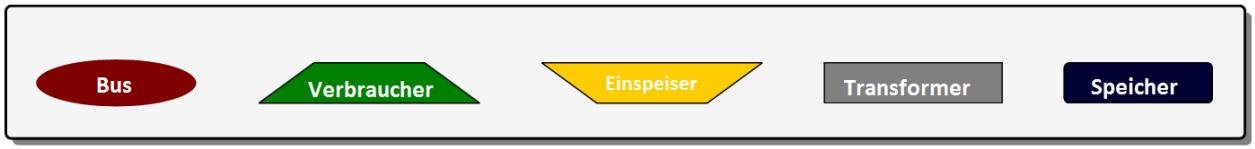

Abbildung 1: Bausteine zum Aufbau eines Energiesystems in oemof [10]

Ein Bus kann als ein Standpunkt eines Energienetzes oder als ein alleiniger Anschlusspunkt für die restlichen Objekte angesehen werden. Ein Transformator ist ein Verbindungselement zwischen zwei Bussen. Dieser kann zwischen zwei Standpunkten eines Energienetzes eingesetzt werden als ein Leitungselement oder zwischen zwei verschieden definierten Bussen, zum Beispiel zwischen einem Gas- und Strombus als eine Umwandlungsstelle definiert werden wie ein Gaskraftwerk. Einspeiser führen dem Energiemodell Energie hinzu, Verbraucher werden mit dieser Energie versorgt. Oemof ist einheitenlos, der Benutzer kann den Energieeinheiten, die im Energiesystem auftreten, auf einen Wert festlegen. Folgendes beispielhaftes Energiemodell könnte somit aufgebaut werden:

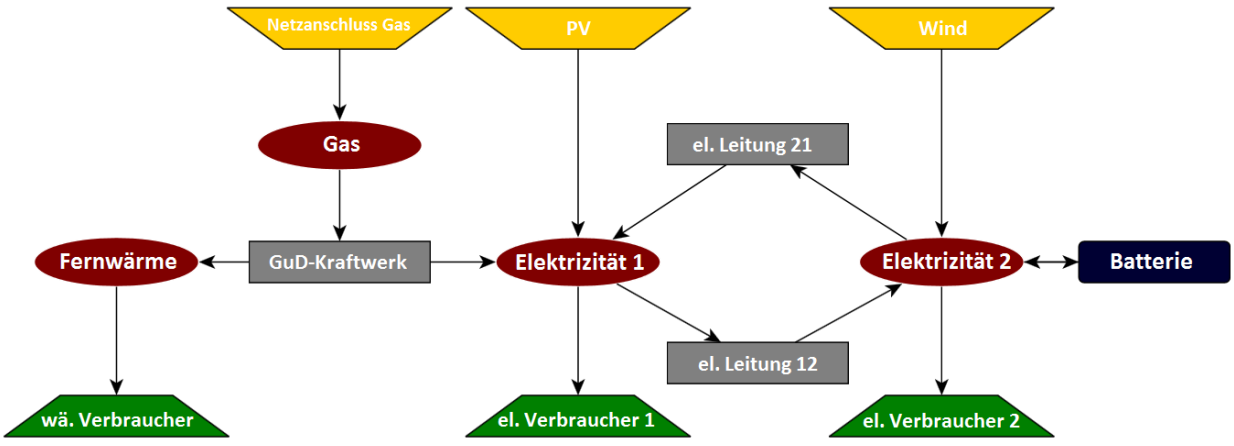

Abbildung 2: Beispielhafter Aufbau eines Energiesystems [10]

Die einzelnen Objekte werden innerhalb des Programms definiert und charakterisiert. Die Möglichkeiten, die sich hier ergeben, werden im Kapitel 4.2 erläutert. Nach dem Aufbau und dem Lösen des optimalen Energieflusses des Energiemodells werden mit der outputlib die

Ergebnisse in Form von graphischen Auswertungen oder Tabellen ausgegeben (siehe Kapitel 5.2).

Die Bibliotheken basieren auf pyomo, ein auf Python-basierendes Software-Paket, das eine Vielzahl an Darstellungen von Optimierungsmodellen unterstützt. Pyomo basiert wie oemof auf Open Source.

4. Aufbau von Objekten in oemof

Dieses Kapitel beschreibt zunächst die Möglichkeiten, verschiedene Komponenten in oemof darzustellen, und erläutert die Probleme und Begrenzung der Darstellungsweise. Anschließend werden die Komponenten, die einerseits in den nachfolgenden Szenarien verwendet werden und andererseits von Relevanz bei zukünftigen Berechnungen sind, einzeln vorgestellt und ihr Aufbau und Programmcode in oemof dargestellt. Dies dient dazu, einen Überblick über die Darstellungsweise von Komponenten eines Energieversorgungssystems zu erlangen. Grundlage der Informationen über die Technologiecharakterisierung ist das „Virtuelle Institut: Strom zu Gas und Wärme“ [11].

Anmerkung zu dem Begriff „Kosten“ im gesamten Kapitel: Variable und fixe Kosten, die in folgenden beispielhaft aufgebauten Objekten enthalten sind, stellen nur eine Möglichkeit der Umsetzung dar und sind in Relation zu den anderen Kosten innerhalb des Energiesystems zu sehen. Die Kosten pro Energieeinheit sind, wie jede andere Eigenschaft, einheitenlos und müssen in einem Szenario erst auf einen Wert definiert werden (zum Beispiel €).

4.1. Energienetze

Jedes Energienetz kann zuerst mit einem einzigen Bus dargestellt werden. Es gibt keine Beschränkung in der Anzahl an Bussen. An jedem dieser Busse können verschiedene Objekte (Einspeiser, Lasten oder Umwandlungsstellen zu anderen Energienetzen) hinzugefügt werden [10]. Das Verknüpfen der Objekte mit den Bussen findet innerhalb der Programmierung der Objekte statt, sodass die Erstellung eines neuen Busses im Energiesystem nur folgenden Quellcode enthält:

```
# erstelle Elektrizitäts_Bus
bel = solph.Bus(label="Elektrizitätnetz")
```

Code 1: Bus

Aufgrund der einheitenlosen Darstellungsweise gibt es keine Unterscheidung zwischen einem Strom-, Gas- und Fernwärmebus. Die Namensgebung ist der einzige Unterschied. Die Objekte haben direkten Kontakt und können unendlich viele Energieeinheiten übertragen, sofern in den jeweiligen Objekten nicht Eigenschaften wie die maximale Nennleistung oder die Übertragung der Energieeinheiten pro Zeit auf ein Maximum beschränkt wird. Objekte wie Leitungen sind in oemof zunächst nicht vorgesehen. Um ein komplettes Energienetz mit Leitungen aufzubauen, wird im Kapitel 3.8 eine mögliche Darstellung von Leitungen vorgestellt und diskutiert.

4.2. Einspeisende Objekte

4.2.1. Einleitung

Energieeinspeisende Objekte sind Energieerzeuger, die in das Energieversorgungssystem Energie einspeisen. Damit sind hier sowohl die erneuerbaren Energien, wie Windanlagen, Photovoltaik und Solaranlagen, als auch Netzanschlüsse, die eine Verbindung von einem übergeordneten Energiesystem zu dem selbst aufgebauten Energienetz darstellen und Energie in dieses einspeisen, gemeint. So können sowohl in übergeordnete Energienetze eingebettete als auch komplett autarke Bilanzräume betrachtet werden. Dieser Sonderfall samt seiner Darstellung wird zum Schluss dieses Kapitels betrachtet. Die konventionellen Einspeiser, etwa ein Kraftwerk, werden im Kapitel 4.9.7 unter Umwandlungsstellen gefasst, da sie keine transportfähige Energie (Strom, Gas oder Fernwärme) erzeugen, sondern diese umwandeln. Die einzelnen Komponenten sind in diesem Kapitel beschrieben. In oemof sind alle energieeinspeisenden Objekte der Klasse *source* zugeordnet.

4.2.2. Auswahl der Eigenschaften in oemof

Die Programmbibliothek oemof bietet ein großes Portfolio an Eigenschaften, die einem energieeinspeisenden Objekt zugeordnet werden können. Diese sind in der Bibliothek feedin_lib hinterlegt. Die nachfolgende Tabelle gibt einen Überblick der Möglichkeiten:

Tabelle 1: Eigenschaften der Objekte [10]

Programmbefehl	Erläuterung
nominal_value=()	Leistung der Anlage
min=()	Mindestbetrag / Minimale Teillastfähigkeit
max=()	Maximale Lastfähigkeit
actual_value=()	Aktuelle Leistung der Anlage
positive_gradient=()	Pro Zeitschritt die max. Erhöhung der aktuellen Leistung
negative_gradient=()	Pro Zeitschritt das max. Herabsetzen der aktuellen Leistung

summed_max=()	Maximum aller Leistungen * Zeitschritte
summed_min=()	Minimum aller Leistungen * Zeitschritte
variable_costs=()	Kosten des Durchflusses pro Einheit
fixed_costs=()	Fixkosten der Anlage
conversion_factors=()	Umwandlungsfaktor (Wirkungsgrad)

Die Eigenschaften sind in dem Paket solph.network innerhalb der oemoef-Bibliothek hinterlegt. Um diese zu benutzen, muss vorab die Bibliothek in das Programm eingebunden werden. Anschließend lassen sich die verschiedenen Befehle auswählen. Objekte, die Energie erzeugen, gehören in oemof zur Klasse *source*.

Einem Objekt muss die Leistung sowie die aktuelle Leistung der Anlage zugeordnet werden und es muss ein Bus ausgewählt werden, in den die Energie eingespeist wird. Um Objekten eine weitere Eigenschaft, die in der Tabelle 1 dargestellt ist, zuzuordnen, wird der Befehl dem Programmcode des Objekts zugefügt und diesem mit einem Gleichheitszeichen ein Wert in einer Klammer zugeordnet:

```
# create Objekt
solph.Source(label='Einspeiser', outputs={bus1: solph.Flow(
    actual_value=data['Leistungsprofil'], nominal_value=1000, fixed=True,
    fixed_costs=1000)})
```

Code 2: Beispielobjekt

Weitere Eigenschaften werden nach dem Setzen eines Kommas hinzugefügt. So ergeben sich für die Darstellung der folgend beschriebenen Objekte verschiedene Möglichkeiten, die eine Charakterisierung der einzelnen Objekte ermöglicht. Abgesehen von dem Nennwert der Anlage sowie den Fixkosten lassen sich neben festen Werten auch ganze Datenreihen den jeweiligen Eigenschaften zuordnen. In diesen Datenreihen ist für jeden Zeitschritt der Simulation ein Wert zugeordnet. Dies benötigt man nicht nur zur Integration von Leistungsprofilen, sondern auch, um zeitlich abhängige Eigenschaften einen variablen Wert zuordnen zu können. So können zum Beispiel Wirkungsgrade von Objekten, die im Sommer aufgrund der Wärme einen anderen Wirkungsgrad besitzen als im Winter, variabel dargestellt werden und nicht nur mit einem festen Wert definiert werden.

4.2.3. Photovoltaikanlage

4.2.3.1. Allgemein

Eine Photovoltaikanlage wandelt Licht in elektrische Energie um. Einzelne Solarzellen-Module, die zusammengefügt eine Anlage ergeben, bestehen meist aus zwei Siliziumschichten, die sich durch eine negative und eine positive Dotierung unterscheiden. Die beiden Schichten, die aufeinander liegen, bilden so einen pn-Übergang, an dem bei Sonneneinstrahlung durch die Strahlung, die eine genügend starke Wellenlänge besitzt, ein Gleichstrom erzeugt wird. Über zwei Kontakte außen an den beiden Schichten kann die entstehende Spannung abgegriffen werden. Polykristalline Module erreichen einen Wirkungsgrad von 16 % [11].

Die Module lassen sich in frei wählbarer Anzahl zusammenfügen. Daraus ergeben sich Anlagengrößen, die für private Häuser geeignet sind, um entweder den temporären Bedarf an elektrischer Energie sicherzustellen, in Kombination mit einem elektrischen Speicher die gewonnene Energie zu speichern oder direkt ins Stromnetz abzugeben. Im großen Maßstab können ganze PV-Felder auch gewerblich für die Energiegewinnung genutzt werden. Die installierte Leistung in Deutschland an PV-Anlagen beträgt aktuell 41.3 GW [Stand 1/2017] [12].

4.2.3.2. Umsetzung in oemof

Die grundlegende Eigenschaft der PV-Anlage als energieeinspeisendes Objekt ist die aktuelle Leistung, die in das elektrische Netz an einem Standpunkt eingespeist wird. Hierfür wird ein Leistungsprofil über eine .csv-Datei eingebunden (s. Kapitel 3.7.2). Die Zuordnung zu einem Leistungsprofil wird am Anfang des Programms festgelegt. Der Ausgang der Energie ist der Bus des vorab festgelegten elektrischen Netzes (Bus). Daraus ergibt sich folgendes Codebeispiel:

```
# create Photovoltaikanlage
  solph.Source(label='Photovoltaikanlage', outputs={bel: solph.Flow(
      actual_value=data['pv'], nominal_value=4000, fixed=True,
      fixed_costs=20)})
```

Code 3: PV-Anlage

Die PV-Anlage lässt sich problemlos in oemof darstellen.

4.2.4. Windkraftanlage

4.2.4.1. Allgemein

Man unterscheidet zwischen Onshore- und Offshore-Windkraftanlagen. Neben Großwindkraftanlagen gibt es noch Kleinwindkraftanlagen im Bereich von ein paar Hundert Watt bis hin zu einigen kW Leistung [13].

Die Energie wird gewonnen durch die Umwandlung von kinetischer Energie über die Bewegung der Rotorblätter in mechanische Energie und anschließendem Antreiben eines Generators in elektrischen Strom. Physikalisch gesehen können maximal 59 % der Energie des Windes genutzt werden. Mit den mechanischen Verlusten an Rotor, Getriebe und im Generator lassen sich abhängig von der Bauart des Windrads circa 45 % der im Wind enthaltenen Energie nutzen [14].

4.2.4.2. Umsetzung in oemof

Die in oemof umgesetzte und hier aufgezeigte Kleinwindkraftanlage hat eine beispielhafte Leistung von 10 kW. Fixkosten können gewählt werden. Bei größeren Anlagen und kleingewählten Zeitschritten ist auf die Begrenzung der Erhöhung der aktuellen Leistung durch den Befehl *positive_gradient=()* zu achten, da eine Anlage nicht ohne zeitliche Verzögerung ihre maximale Leistung erbringen kann. Startkosten könnten ebenfalls definiert werden, falls bei größeren Anlagen diese eine Rolle spielen.

```
# create Kleinwindkraftanlage
solph.Source(label='Kleinwindkraftanlage', outputs={bel: solph.Flow(
    actual_value=data['wind'], nominal_value=10000, fixed=True,
    fixed_costs=1000)})
```

Code 4: Windkraftanlage

Diese wenigen Eigenschaften der Windkraftanlage können somit ausreichend in oemof dargestellt werden.

4.2.5. Thermische Solaranlage

4.2.5.1. Allgemein

Als thermische Solaranlage werden jene Solaranlagen bezeichnet, die Sonneneinstrahlung in thermische Energie umwandeln. Meist auf Dächern verbaute Solarmodule, die eine wärmeführende Flüssigkeit enthalten, werden durch die Sonneneinstrahlung erwärmt. Eine

Pumpe kann die erhitzte Flüssigkeit zum Wärmetauscher abführen und dort erfolgt der Wärmeaustausch zwischen der wärmeführenden Flüssigkeit und einem angeschlossenen Wärmespeicher, wie zum Beispiel einem Warmwasserspeicher [15].

4.2.5.2. Umsetzung in oemof

```
# create Solaranlage
  solph.Source(label='Solaranlage', outputs={bwarm: solph.Flow(
    actual_value=data['solar'], nominal_value=3000, fixed=True,
    fixed_costs=1000)})
```

Code 5: Solaranlage

Dem Objekt wird über den *actual_value* ein Leistungsprofil als .csv-Datei hinterlegt. Dieses Leistungsprofil sollte temperaturabhängige Werte beschreiben, da in oemof kein separates Modul für die Umgebungstemperatur hinterlegt werden kann. Ausgegeben wird die Energie in den Bus „bwarm“. Da die Wärmeübertragung eine Vorlaufzeit benötigt, könnte man mit dem Befehl *positive_gradient=()* die minimale Zeit, bis der Durchfluss erhöht wird, angeben. Dies spielt bei größeren Anlagen und einer Betrachtung von sehr kleinen Zeitschritten eine Rolle.

4.2.6. Anschlüsse zur Energieabgabe in übergeordnete Netze

4.2.6.1. Umsetzung in oemof

Da nicht nur autarke Energiesysteme betrachtet werden, müssen Verbindungen zu einem vorhandenen Energienetz geschlossen werden. Energie kann nur in eine Richtung fließen. Ein solcher Anschluss wird für das Programm wie ein energieeinspeisendes Objekt behandelt. Der Unterschied für die anschließende Berechnung innerhalb des Programms ist der Preis der Energie im Vergleich zu den Kosten der definierten energieeinspeisenden Objekte.

```
# create El. Anschluss
  solph.Source(label='Anschluss', outputs={bel: solph.Flow(
    actual_value=data['Netz'], nominal_value=10e10, fixed=True,
    fixed_costs=1000)})
```

Code 6: Anschluss an ein übergeordnetes Netz

4.3. Energetische Lasten

4.3.1. Allgemein

Eine energetische Last gibt den aktuellen Bedarf von einer Energieform an. Die einzelnen Verbraucher mit ihren Lasten ergeben in einem Zusammenschluss ein sogenanntes Lastprofil, welches den Gesamtbedarf an einer Energieform widerspiegelt. Die Last wird in der Einheit Watt angegeben. Zu jedem Zeitpunkt kann sich die aktuelle Last ändern. Deswegen ist in einem Lastprofil in regelmäßigen Abständen von 15, 30 oder 60 Minuten, je nachdem welche zeitliche Dauer für die Berechnung sinnvoll ist, die aktuelle Last angegeben. Indem man die jeweiligen Lasten mit der Zeit multipliziert, erhält man den Gesamtverbrauch eines Zeitabschnitts in Wh. Die Energieformen Strom, Gas und Fernwärme werden getrennt voneinander behandelt.

Äquivalent dazu haben energieeinspeisende Objekte wie Photovoltaikanlagen, Windräder oder Solaranlagen ebenso Einspeiseprofile. Diese geben statt der Last die derzeitige Leistung der Anlagen an.

4.3.2. Lastprofile

Um für ein Szenario realistische Verbraucherwerte einzubetten, werden Standardlastprofile der einzelnen Energieformen je nach Szenario ausgewählt. Für die Lastprofile der einzelnen Objekte werden unterschiedliche Daten eingelesen, abhängig von dem Ziel des einzelnen Szenarios. Wichtig ist, dass variable Last- und Leistungsprofile in oemof eingebunden werden können.

Es gibt verschiedenen Arten von Lastprofilen, deren Auswahl vor allem entscheidend ist, wenn es um den Aufbau eines Referenznetzes geht, um Vorhersagen für zukünftige Energieflüsse zu bestimmen. In dieser Arbeit wird aber nur die Funktion oemofs hinsichtlich des Projektes betrachtet. Deswegen spielt die Auswahl der Lastprofile innerhalb der folgenden Versuchsszenarien keine wichtige Rolle.

4.3.3. Umsetzung in oemof

Die Lastprofile für jeden einzelnen Verbraucher und die Leistungsprofile jedes energieeinspeisenden Objekts werden mit einer Excel-Tabelle hinterlegt. Die Tabelle besteht aus einer Spalte, wo zu jedem Zeitpunkt der aktuelle Wert der Last hinterlegt ist. Weitere Lastprofile können ebenfalls in die gleiche Excel-Tabelle eingebettet werden, indem in der ersten Zeile der Excel-Tabelle die Namen der Objekte, die einer Last zugeordnet sind, mit Kommata und einem Leerzeichen zusammengefügt werden. Anschließend werden in dieser

festgelegten Reihenfolge die Daten der Last- und Leistungsprofile in den weiteren Spalten hinterlegt. Beispielhaft für vier Zeitschritte folgende Tabelle:

```
Zeitschritt,Verbrauch,pv,wind
1,100,0,0.4
2,80,0,0.474873
3,70,0,0.55654
4,100,0,0.265059
```

Tabelle 2:Last- und Leistungsprofile einlesen

Um für Übersicht zu sorgen, können auch separate einzelne Excel-Tabellen zu den einzelnen Lasten angelegt werden. Um einem erstellten Objekt ein Lastprofil zuzuordnen, wird folgender Programmcode eingegeben:

```
def optimise_storage_size(filename="Jahr 2016.csv", solvername='cbc',
    debug=True, number_timesteps=8760, tee_switch=True):
    logging.info('Initialisiere das Energiesystem')
    date_time_index = pd.date_range('1/1/2016', periods=number_timesteps, freq='H')
```

Code 7: Zuordnung Lastprofile

Der Filename „Jahr 2016.csv“ bezieht sich auf den Namen der Excel-Tabelle, in der die Lastprofile der verwendeten Objekte und Verbraucher hinterlegt sind. Der Filename wird im Programm nur einmal übergreifend eingelesen und im weiteren Verlauf des Programms nicht mehr verwendet. Diese Datei muss sich im selben Ordner wie der Programmcode befinden, um vom Programm zur späteren Berechnung eingelesen werden zu können. Durch das Ändern der Anzahl der Zeitschritte und dem dazugehörigen Anpassen der Werteliste der Lasten in der .csv Datei kann die Frequenz, mit der neue Werte eingelesen werden, für die Berechnung verändert werden.

4.4. Anschluss für überschüssige Energie

Um überschüssige Energie abzuführen, kann beispielhaft ein sogenannter *sink* (engl.: Senke) definiert werden:

```
# create Abgang überschüssiger Energie
  solph.Sink(label='Überschuss_El', inputs={bel: solph.Flow()})
```

Code 8: Abgabe überschüssiger Energie

Dieser kann aus einem definierten Bus, im Beispiel *bel*, Energie aufnehmen. Der *sink* wird erst dann bedient, wenn die im System angeschlossenen Verbraucher gesättigt sind und durch weitere Einspeisung von energieeinspeisenden Objekten überschüssige Energie im Bus vorhanden ist.

4.5. Leitungen

4.5.1. Einleitung

Die Programmbibliothek oemof beinhaltet aktuell kein separates Leitungsobjekt. Somit ist eine tiefer gehende Netzsimulation, wie beispielsweise die Lastflussanalyse nicht möglich. Aufgrund der einheitenlosen Bauteile lassen sich jedoch trotzdem Leitungen mithilfe von Umwandlungsstellen darstellen. Hierfür wird die grundsätzliche Idee, dass ein Energienetz lediglich mit einem Bus dargestellt wird, abgeändert. Erstellt man mehrere Busse und verbindet diese mit Umwandlungsstellen, so erhält man einen Aufbau eines Energieverteilnetzes, welches eine Anzahl von n Bussen beziehungsweise eine Anzahl von n Stellen aufweist, an denen Energie ein- und abgespeist sowie in einen anderen Standpunkt des Energiesystems übertragen werden kann.

Das in oemof für die Leitungen verwendete Objekt „Transformator“, welches eigentlich dazu dient, eine Koppelstelle zwischen zwei Energiesystemen darzustellen, kann somit als eine Leitung verwendet werden. Dafür muss man sich im Klaren sein, dass nun auch nur die Eigenschaften, die für einen „Transformator“ zur Verfügung stehen, nun auch für einen Leitungsaufbau zur Verfügung stehen.

Hierzu werden nun die Parameter der Umwandlungsstellen, welche für den Leitungsaufbau verwendet werden aufgezählt, anschließend beschrieben, welche Eigenschaften damit dargestellt werden können und auf welche Eigenschaften man verzichten muss.

4.5.2. Verwendete Eigenschaften der Umwandlungsstellen zur Darstellung einer Leitung

Tabelle 3: Eigenschaften einer Leitung

Programmbefehl	Erläuterung
nominal_value=()	Nennwert
min=()	Mindestbetrag
conversion_factors=()	Umwandlungsfaktor

Die für Umwandlungsstellen typischen Eigenschaften wie Hochfahrkosten, Mindest- und Maximalauslastung sowie die maximale Erhöhung der Auslastung einer Anlage spielen für die Darstellung eines Bauteils wie dem einer Leitung keine Rolle. Die wichtigste Eigenschaft einer Leitung ist der Nennwert, also in diesem Fall die Menge an Energieeinheiten, die zu einem Zeitpunkt die Leitung durchfließen kann. Mit dem Umwandlungsfaktor lassen sich Verluste darstellen.

4.5.2.1. Elektrische Leitungen

Aufgrund der einheitenlosen Darstellungsweise muss auf eine Aufteilung der Energie in Strom und Spannung verzichtet werden. Diese berechnet man generell mit einer Lastflussberechnung. Ziel der Lastflussberechnung ist es, die Spannungen, die an den verschiedenen Knoten (Standorte) in einem Netzwerk vorhanden sind und die sich bei einer definierten Einspeisung und Entnahme von Leistungen einstellen, zu berechnen. Ein gängiges Verfahren hierfür ist das Knotenpotentialverfahren. Bei diesem Verfahren lassen sich die einzelnen Knotenpotentiale ermitteln, indem die Spannungen als Funktion von Strömen an Knoten, Lastimpedanzen und Knoten-Leistungen dargestellt werden. Damit lassen sich die Durchflüsse eines Netzwerkes berechnen [16].

Dies steht allerdings zuerst im Widerspruch zu der Optimierung auf den optimalen Energiefluss eines Netzwerks innerhalb oemofs, da mit der Lastflussberechnung eben dieser Fluss berechnet wird. Es müssten innerhalb oemofs nur die Flüsse der Einspeiser optimiert werden und die restlichen Objekte eines Energiesystems müssten festdefiniert bzw. durch die Ergebnisse der Lastflussberechnung definiert werden.

Des Weiteren fällt direkt auf, dass leitungsspezifische Eigenschaften der elektrischen Leitungen wie Widerstandsbelag, Leitwertsbelag sowie der Induktivitäts- und Kapazitätsbelag nicht darstellbar sind [17].

Eine Lastflussberechnung ist somit nicht möglich. Die Verluste elektrischer Leitungen sind kein fester Wert, sondern werden durch die induktive oder kapazitive Art der an der Leitung zugeschalteten elektrischen Verbrauchern und Energieeinspeisern entscheidend bestimmt.

Das grobe Werkzeug einer Beschränkung des Energieflusses lässt immerhin eine realistische, wenn auch nicht genaue Darstellung einer realen Leitung zu. Ähnlich verhält es sich bei den Verlusten einer Leitung. Diese werden im Fall einer elektrischen Leitung durch den Widerstandsbelag und den Leitwertbelag definiert. Bei einem Aufbau eines Energienetzes bietet es sich daher an, Leitungen eine maximale Kapazität sowie einen Verlust in Form von der Eingabe eines Wirkungsgrads zuzuordnen. Diese maximale Kapazität sollte den Wert des Energieflusses beschreiben, der für die Leitung keinen kritischen Zustand bedeutet. Dies ist nur eine Vereinfachung und stellt nicht ein realitätsnahes Abbild einer elektrischen Leitung dar.

4.5.2.2. Fernwärmeleitungen

Fernwärme ist definiert als Übertragung von thermischer Energie durch ein Rohrleitungssystem mittels eines Fluids von einem zentralen Wärmeerzeuger zu Verbrauchern wie zum Beispiel einer Warmwasserheizung [18]. Wärmeverluste von Fernwärmeleitungen sind maßgeblich durch die Güte der Dämmung bestimmt. Ein weiterer entscheidender Faktor ist die Flussgeschwindigkeit. Ebenso sind die Temperaturumgebung, die Bodenart, in der das Rohr verlegt ist, und zwischen Vor- und Rücklauf des Leiters zu unterscheiden [19]. Eine temperaturabhängige Wertetabelle, die die Verluste beschreibt, könnte für den Wirkungsgrad einer Leitung hinterlegt werden.

Tabelle 4: Eigenschaften einer Fernwärmeleitung [20]

NW	c	Q_{max}	Q_a	Dämmdicke	k_i-Wert	Verluste		
mm	m/s	kW	MWh/a	mm	W/m^2K	kWh/ma	%/100 m	Entf. für 10 %
25	1,2	60,4	121	25	2,91	80,1	13,26	75,4 m
40	1,3	167,8	336	34	2,01	88,3	5,26	190 m
65	1,41	479,1	958	44	1,45	103,3	2,16	463 m
100	1,5	1.209	2.419	52	1,12	123,3	1,02	980 m
150	1,59	2.881	5.762	60	0,91	149,7	0,52	1920 m

Die Tabelle zeigt die Eigenschaften einer Fernwärmeleitung und beruht auf typischen Werten der Auslegungsvorlauftemperatur von 70 Grad (Rücklauf 45 Grad), mittlere Vorlauftemperatur von 60 Grad (Rücklauf 40 Grad), Wärmeleitfähigkeit der Dämmung von 0,04 W/mK und einer Erdtemperatur von 10 Grad. Daraus ergeben sich die unterschiedlichen Verluste der Leitungstypen. Ein im Stadtgebiet typisches Rohr mit einer Nennweite von 65 mm weist auf 100 m somit einen Verlust von 2,16 % aus. Das Problem hierbei ist, dass die genaue Temperatur, die in einem Rohr vorherrscht, nicht bekannt ist und somit die Verluste nur vereinfacht dargestellt werden können. Die Tabelle soll dabei helfen, die stark variablen Leitungsverluste zu vereinfachen und trotzdem möglichst realitätsgetreu darzustellen.

4.5.2.3. Gasleitung

Ebenso ist keine genaue Beschreibung des Gasnetzes durch verschiedene Leitungen möglich. Das Gasnetz ist mit verschiedenen Leitungstypen aufgebaut, die unterschiedliche Gasdurchsätze und Gasdrücke vorweisen und dementsprechend variable Gasverluste aufweisen. Generell gibt es drei Druckstufen innerhalb der Gasversorgung: Hochdruck (p=> 1 bar), Mitteldruck (100mbar <p>1bar) und den Niederdruck (22mbar<p<100mbar), welche die Versorgungsunternehmen aus einem vorgelagerten Gasnetz heraus versorgen [20]. Um einen realistischen Gasdurchsatz für die jeweiligen Szenarien aufzubauen, muss entschieden werden, welche Art von Gasnetz passend ist – betrachtet man ganze Stadtteile und somit ein übergeordnetes Gasnetz oder nur eine Haussimulation, bei der man mit Niederdruck rechnet. In oemof kann hier also auch nur vereinfacht von einer Leitung mit einer maximalen Durchflussmenge ausgegangen werden und einem Verlustfaktor in Form eines Wirkungsgrades.

4.5.3. Energieform

Ein weiteres Problem, das bei der Darstellung der Leitungen auftritt, ist eine einheitliche Definition der Energieform in allen drei hier verwendeten Energiearten. Generell wird der Energiegehalt des Stroms in kWh beziehungsweise in MWh gemessen.

Gas wird in m^3 gemessen. Einem Kubikmeter wird dann ein spezifischer Brennwert (kwh/m^3) zugeordnet. Der Brennwert definiert die Energiemenge, die bei der Verbrennung und der darauffolgenden Abkühlung des Gases freigesetzt wird.

Dieser ist beim Gas abhängig von der Zusammensetzung der verschiedenen Gase im Netz, die sich zwar in einem vorgegebenen Rahmen bewegen muss, um den störungsfreien Betrieb zu garantieren, sich jedoch leicht unterscheiden kann und somit den tatsächlichen Brennwert schwer zu bestimmen lässt. Hauptsächlich unterscheidet man in Deutschland

zwischen L-Gas (low caloric gas), das in Norddeutschland und den Niederlanden gefördert wurde. Dieses hat einen Brennwert zwischen 8–10 kwH/m³ bei einem Methangehalt von 80–87 %. In den Nordseeerdgasfeldern wird das sogenannte H-Gas (high caloric gas) gefördert, welches einen Brennwert von 10–12 kWh/m³ und einem Methangehalt von 87–98 % aufweist [21]. Der höhere Brennwert beruht auf einen höheren Butan- sowie Propananteil im Gas. Die Energieversorgungsunternehmen rechnen im Einzelnen mit einem durchschnittlichen Brennwert, auch wenn dieser den Energiegehalt nicht genau angibt.

Für Fernwärme wird als Übertragungsmedium Wasser verwendet. Die Einheit ist meistens wie beim Strom in kWh und teilweise in m³ angegeben. Den m³ muss dann natürlich ein temperaturabhängiger Wert zugeordnet werden, um die tatsächliche Energiemenge zu bestimmen.

Es bietet sich hier an, die übergeordnete Energieeinheit Wh als Richtwert festzulegen. Dies bedeutet, dass die Gaseigenschaften von Kubikmetern beziehungsweise Litern in Wh umgerechnet werden müssen. Sofern es sich anbietet, kann auch kWh oder MWh als Richtwert benutzt werden.

4.5.3.1. Umsetzung in oemof

Zusammengefasst lässt sich an dieser Stelle sagen, dass die beschränkte Auswahl an vorhandenen Leitungsparametern in oemof zwar für eine realistische Berechnung noch ausreicht, diese aber nicht befriedigend sind, um eine reale Leitung für den Aufbau eines Energieversorgungssystems darzustellen. Vor allem im Fall der elektrischen Leitungen müssen aufgrund des einheitenlosen Aufbaus in oemof sowie des nicht vorhandenen separaten Moduls eines Leitungselements und somit begrenzten Auswahl an Parametern, Abstriche gemacht werden.

Oemof befindet sich in der Weiterentwicklung und es ist nicht ausgeschlossen, dass in Zukunft die Möglichkeit besteht, weitere Eigenschaften der Leitungen zu definieren.

Eine Umwandlungsstelle hat die Grundeigenschaft, Energieeinheiten aus einem Bus zu entnehmen und diese in einen weiteren Bus einzuspeisen. Dies auf eine Leitung reflektiert bedeutet, dass Energie von einem der n Punkte in einem Energienetz nur in eine Richtung zum nächsten Punkt des Energienetzes abgegeben werden kann. Da eine Leitung eines Energienetzes jedoch die Eigenschaft besitzen soll, Energie in beide Richtungen zu übertragen, werden pro Verbindung zwischen den Punkten im Energienetz jeweils zwei Leitungen, Hin- und Rückleiter, benötigt. Zwischen zwei Standorten 1 und 2 eines Energiesystems wird eine Leitung einerseits L12 definiert. L12 steht für den Energiefluss von

Punkt 1 zu 2. Das Gegenstück dazu ist die Leitung L21, sie beschreibt den umgekehrten Energiefluss. Die Hin- und Rückführung ist nötig, da keine negativen Werte innerhalb der Berechnung durchgeführt werden.

Negative Werte würden die Eigenschaften der verwendeten Objekte ad absurdum führen, da somit zum Beispiel eine Koppelstelle plötzlich die Eigenschaft erhalten würde, Energie in zwei Richtungen umwandeln zu können. Folgend ist ein Aufbau eines Hin- sowie eines Rückleiters zwischen zwei Standorten eines Energiesystems beispielhaft aufgebaut:

```
# create L12 - Durchfluss von Bus 1 zu Bus 2
   solph.LinearTransformer(
      label="L12",
      inputs={bel1: solph.Flow()},
      outputs={bel2: solph.Flow(nominal_value=400, variable_costs=10)},
      conversion_factors={bel2: 0.9})
```

Code 9: Hin-Leitung

```
# create L21  - Durchfluss von Bus 2 zu Bus 1
   solph.LinearTransformer(
      label="L21",
      inputs={bel2: solph.Flow()},
      outputs={bel1: solph.Flow(nominal_value=400, variable_costs=10)},
      conversion_factors={bel1: 0.9})
```

Code 10: Rück-Leitung

4.6. Energiespeicher

4.6.1. Einleitung

Energiespeicher dienen zur Speicherung von Energie, um sie zu einem späteren Zeitpunkt zu nutzen. Ein Energiespeicher ist definiert als eine energietechnische Einrichtung, die drei Prozesse enthält: Einspeichern (Laden), Speichern und Ausspeichern (Entladen) [22].

Technologisch lassen sich Energiespeicher in fünf Gruppen unterteilen: mechanische, chemische, elektrochemische, elektrische und thermische Speicher. Sie nehmen eine wichtige Rolle innerhalb der Energiewende ein, da sie fluktuierende Energieerzeugung durch Auf- und Entnahme von Energie ausgleichen können. Des Weiteren lassen sich Energiespeicher hinsichtlich der zeitlichen Dauer der Speicherung in Kurz- bis Langzeitspeicher unterscheiden. Beispiel für einen Kurzzeitspeicher ist ein einfacher Kondensator, der Energie für Bruchteile einer Sekunde speichert; ein Langzeitspeicher ist zum Beispiel ein Untertagespeicher, der Gas saisonal speichern kann. In diesem Kapitel werden typische Energiespeicher der betrachteten Strom-, Gas- und Fernwärmenetze aufgezeigt und auf die Umsetzbarkeit hin in oemof überprüft.

4.6.2. Möglichkeiten der Charakterisierung in oemof

In oemof werden Energiespeicher unter der Klasse „storage“ zusammengefasst. Eine Reihe an Eigenschaften gehören dieser Klasse an, die in folgender Tabelle einzusehen sind:

Tabelle 5: Eigenschaften der Energiespeicher [10]

Programmbefehl	Erläuterung
nominal_capacity=()	Nennwert der Kapazität
nominal_input_capacity_ratio=()	Verhältnis zwischen Zufluss und Kapazität
nominal_output_capacity_ratio=()	Verhältnis zwischen Abfluss und Kapazität
initial_capacity=()	Anfangswert der Kapazität (geladen/leer)
capacity_loss=()	Kapazitätsverlust pro Zeitschritt
inflow_conversion_factor=()	eingehender Wirkungsgrad
outflow_conversion_factor=()	ausgehender Wirkungsgrad
capacity_min=()	Kapazitätsminimum
capacity_max=()	Kapazitätsmaximum

Eine der Haupteigenschaften ist der Nennwert der Kapazität, der hier die Einheit Wh zugeordnet wird, wenn die Leistung als Watt definiert ist. Neben der Kapazität ist der Wirkungsgrad eine der entscheidenden Eigenschaften des Speichers. Dieser gibt an, wie hoch der Verlust pro eingespeicherte und wieder ausgegebene Energieeinheit des Gesamtsystems des Energiespeichers ist. Verluste, die ein Energiespeicher mit

fortschreitender Zeit erleidet, können mit dem Befehl *capacity_loss=()* pro Zeitschritt festgelegt werden. Den verschiedenen Eigenschaften können entweder feste Werte zu geordnet werden oder eine Tabelle an Werten zugeordnet werden. In dieser Tabelle ist für jeden Zeitschritt ein Wert, der die Eigenschaft definiert, zugeordnet. Einbindung und Aufbau dieser Tabellen sind in Kapitel 4.3.3 beschrieben.

4.7. Verwendete Energiespeicher

4.7.1. Elektrische Batterie

4.7.1.1. Allgemein

Ein viel verwendeter Energiespeicher zur Speicherung elektrochemischer Energie ist der Lithium-Ionen-Akkumulator. Er zeichnet sich im Gegensatz zu anderen Akkumulatoren-Typen vor allem durch seine chemische Beständigkeit aus. Beim Beladen wandern die Lithium-Ionen von der positiv geladenen Elektrode zur negativ geladenen Elektrode des Akkumulators.

Trotzdem ist nach circa 1.000 Be- und Entladevorgängen mit einem Kapazitätsverlust von rund 20 % zu rechnen [23].

Der Preis für Li-Ion-Akkumulatoren ist in den vergangenen fünf Jahren von 10 € pro installierter kWh auf circa 5 €/kWh gefallen. [24] Aufgrund der Massenfertigung und noch verfügbarer Ressourcen ist mit einem weiteren Preisverfall zu rechnen. Dennoch ist die elektrische Form der Energiespeicherung eine der teuersten und einer der Gründe, weshalb Koppelstellen hier für Abhilfe schaffen sollen.

4.7.1.2. Umsetzung in oemof

Ausgewählt wird ein Standard-Lithium-Ionen-Akkumulator, der eine Nennkapazität von 2kWh besitzt. Der monatliche Kapazitätsverlust beträgt 2 % [23], dieser wird auf den ausgewählten Zeitschritt heruntergerechnet. Bei einem Zeitschritt von einer Stunde beträgt der Kapazitätsverlust pro Zeitschritt 0.001 %. Der Umwandlungsfaktor beschreibt den Wirkungsgrad und lässt sich sogar aufteilen auf den Eingang und den Ausgang. Zusammengenommen liegt er bei 90 % [23]. Die variablen Kosten beschreiben wie gehabt

den Preis der Energie pro gespeicherte Energieeinheit. Mit der *capacity_ratio=()* lässt sich das Verhältnis zwischen Zufluss und der vorhandenen Kapazität beschränken, um eine Aufladung des Akkumulators zeitlich einzugrenzen.

```
# create Batteriespeicher
solph.Storage(
    label='Batteriespeicher',
    inputs={bel: solph.Flow(variable_costs=10)},
    outputs={bel: solph.Flow(variable_costs=10)},
    capacity_loss=0.001, initial_capacity=0, nominal_value=2000, capacity_max=2000,
    nominal_input_capacity_ratio=1/3, nominal_output_capacity_ratio=1/3,
    inflow_conversion_factor=1, outflow_conversion_factor=0.9)
```

Code 11: Lithium-Ionen-Akkumulator

Ein- sowie Ausgang des Speichers ist der Bus des elektrischen Netzes, mit dem der Batteriespeicher verbunden ist. Eine Batterie lässt sich in oemof gut konzipieren und alle relevanten Eigenschaften sind umsetzbar. Sofern die Batterie Temperaturschwankungen ausgesetzt ist, die Einfluss auf die Selbstentladung der Batterie nehmen, ist es möglich, anstatt eines starren Wertes für den Umwandlungsfaktor eine Datenreihe einzubinden, die zu jedem Zeitschritt einen jeweiligen Umwandlungsfaktor ausgibt. Somit können trotz der einheitenlosen Darstellung temperaturabhängige Eigenschaften dargestellt werden.

4.7.2. Wärmespeicher

4.7.2.1. Allgemein

Wärmespeicher sind in thermische und chemische Energiespeicher zu unterscheiden. Thermische Energie ist sensibel, sprich fühlbar, im Gegensatz zur chemischen Energie, die latent ist [22]. Ein einfacher Wasserspeicher speichert die Energie thermisch. Dieser ist in Kombination zu einer Solarthermie-Anlage ein typischer Bestandteil im privaten Gebrauch. Je nach Speicherdauer wird der Wärmespeicher unterschieden in Tages- oder Wochenspeicher. Der Wärmeverlust ist von Speicheroberfläche, Außentemperatur, Innentemperatur und Isolierung des Speichers abhängig.

4.7.2.2. Umsetzung in oemof

Die Verluste eines Speichers können mit einem Umwandlungsfaktor beschrieben werden. Ist aufgrund von variablen Temperaturen dieser Faktor ein Wert, der sich temporär ändert, so kann hier vorab eine Zuordnung von Temperatur zu Umwandlungsfaktor stattfinden und

diese variablen Werte mit einer Tabelle eingelesen werden. Die Unterteilung in eingehenden und ausgehenden Umwandlungsfaktor kann die Verluste genauer unterteilen. Mit der ein- und ausgehenden *capacity_ratio* kann das Verhältnis zwischen Zufluss und Kapazität festgelegt werden. Ein- sowie Ausgang ist das Wärmenetz. Daraus ergibt sich folgender beispielhafter Aufbau:

```
# create Wärmespeicher
  solph.Storage(
    label='Wärmespeicher',
    inputs={bwarm: solph.Flow(variable_costs=100)},
    outputs={bwarm: solph.Flow(variable_costs=100)},
    capacity_loss=0.002, initial_capacity=0, nominal_value=20000,
    nominal_input_capacity_ratio=0.2, nominal_output_capacity_ratio=0.2,
    inflow_conversion_factor=0.99, outflow_conversion_factor= data['aktueller Wirkungsgrad'])
```

Code 12: Warmwasserspeicher

4.7.3. Biogasspeicher

4.7.3.1. Allgemein

Ein Biogasspeicher dient dazu, Schwankungen einer Biogasanlage durch temporäre Speicherung auszugleichen, bevor das Gas zum Verbrennen in einem Blockheizkraftwerk oder ähnlichen Anlagen genutzt wird. Gasspeicher sind unterteilt in Nieder-, Mittel- und Hochdruckspeichern. Niederdruckspeicher speichern das Gas bei einem Druck bis zu 30mbar. Die Mittel- und Hochdruckspeicher weisen einen Druck von 5–200 bar auf. Hierfür werden Stahlbehälter benötigt. Mehr Druck bedeutet mehr Brennwert pro Volumen, dafür steigen aber auch neben den Fixkosten die Kosten an Kompression. [25]

4.7.3.2. Umsetzung in oemof

Die verschiedenen Arten an Speichern können problemlos durch unterschiedliche Fixkosten und variable Kosten und unterschiedlichen Kapazitätsverlust dargestellt werden. Dieser kommt bei den Niederdruckspeichern zum Tragen. Der maximale Durchfluss pro Zeit ist durch die *capacity_ratio=()* eingeschränkt. Die wichtigste Eigenschaft, der Wirkungsgrad liegt bei annähernd 100 %. Folgender Aufbau eines Speichers könnte somit gewählt werden:

```
# create Biogasspeicher
solph.Storage(
    label='Biogasspeicher',
    inputs={bgas: solph.Flow(variable_costs=100)},
    outputs={bgas: solph.Flow(variable_costs=100)},
    capacity_loss=0.0001, initial_capacity=0, nominal_value=10000,
    nominal_input_capacity_ratio=0.1, nominal_output_capacity_ratio=0.1,
    inflow_conversion_factor=0.99, outflow_conversion_factor=0.99)
```

Code 13: Biogasspeicher

Es können alle relevanten Eigenschaften in oemof umgesetzt werden. Gasspeicher sind in der Regel keinen zu extremen Temperaturen ausgesetzt, die einen Anstieg des Drucks in einem Maße erlauben würden, dass die Verluste temporär ansteigen würden. Eine Temperaturabhängigkeit der Parameter spielt deshalb keine Rolle.

4.8. Umwandlungsstellen

4.8.1. Allgemein

Eine Umwandlungsstelle wandelt Energie von einer Form in eine andere Form um. Die Energieformen, die hier betrachtet werden, sind Strom, Gas und Fernwärme. Bei der Umwandlung entstehen Verluste. Umwandlungsstellen werden auch als Koppelstellen bezeichnet, da sie eine Kopplung zwischen verschiedenen Energiesektoren darstellen. Sie bieten große Chancen, überlastete Energienetze zu entlasten, indem sie die Belastung an Energie besser verteilen, und werden als Schlüssel angesehen, um Energie, die sich nur sehr kostenintensiv speichern lässt, in eine kostengünstigere Energieform umzuwandeln und anschließend zu speichern. Ein typisches Beispiel hierfür ist eine Power-to-Gas-Anlage, in der Strom, der vom elektrischen Netz nicht mehr aufgenommen werden kann und nur mit teuren Batteriespeichern gespeichert werden könnte, umzuwandeln in Gas, welches sich wesentlich kostengünstiger speichern lässt.

Neben den typischen Kopplungsstellen fallen unter diesen Begriff aber auch Energieerzeuger, die aus einer Energieform eine andere erzeugen, die sich einfacher transportieren lässt. Beispiele dafür sind ein Gas- oder Kohlekraftwerk.

Auch fallen Gasthermen, Heizstäbe und elektrische Heizungen, die in den meisten Häusern verbaut sind, unter den Begriff.

Folgend werden zuerst die allgemeinen Möglichkeiten beschrieben und anschließend die einzelnen Umwandlungsstellen charakterisiert und Probleme bei der Umsetzung erläutert.

4.8.2. Möglichkeiten zur Darstellung in oemof

Ergänzend zu den Eigenschaften, die auch schon den Objekten zugeordnet sind (siehe Tabelle 1), bieten die nachfolgend in der Tabelle 6 dargestellten Eigenschaften weitere Möglichkeiten zur Definition einer Umwandlungsstelle. Dazu ist die Haupteigenschaft, der Wirkungsgrad einer Umwandlungsstelle in der Klasse solph.network unter dem Begriff *conversion_factors=()* gespeichert. Man gibt den Faktor an, mit dem die Energieeinheiten in einem zugeordneten Energienetz (=Bus) ausgegeben werden sollen.

Tabelle 6: Eigenschaften Umwandlungsstellen [10]

Programmbefehl	Erläuterung
startup_costs=()	Startkosten eines Objektes
shutdown_costs=()	Herunterfahrkosten eines Objektes
minimum_uptime=()	Minimale Zeit, bis der Durchfluss größer wird.
minimum_downtime=()	Minimale Zeit, bis der Durchfluss nach dem Ausschalten den Wert 0 erreicht.

Die Start- und Herunterfahrkosten einer Anlage sowie die Minimierung der Erhöhung des Durchflusses erlauben eine detailgetreuere Darstellung als nur die Eigenschaften aus der solph.network Klasse, die grundsätzlich für die energieeinspeisenden Objekte zur Verfügung steht. Nach einer Namenszuordnung werden die ein- und ausgebenden Busse bestimmt. Diese Energieflüsse sind meistens in den folgenden Szenarien zu berechnen und deshalb strikt mit dem Befehl *flow* verbunden (siehe Kapitel 5).

4.8.3. Probleme

Oemof bietet mit den zusätzlichen Charakterisierungsmöglichkeiten aus Tabelle 6 zu den Eigenschaften der Tabelle 1 eine vielfältige Auswahl an Eigenschaften, um Umwandlungsstellen individuell darstellen zu können. Probleme ergeben sich wie bei den Energiespeichern aus der einheitenlosen Darstellungsweise. Hier muss man bei genauerer Betrachtung einzelner Fälle auf Details verzichten. Diese fallen jedoch nur bei einer expliziten Darstellung einzelner Umwandlungsstellen auf.

4.9. Auswahl an Umwandlungsstellen

4.9.1. Gas-Brennwertkessel

4.9.1.1. Allgemein

Ein Gas-Brennwertkessel erhitzt Brauchwasser durch Verbrennung von Gas. Durch die Nutzung der Kondensationswärme unterscheidet sich der Brennwertkessel von einer konventionellen Gastherme. Das erwärmte Wasser kann zur Warmwassernutzung oder zum Heizen verwendet werden [26].

Der Energiefluss innerhalb des Gas-Brennwertkessels stellt sich folgendermaßen dar:

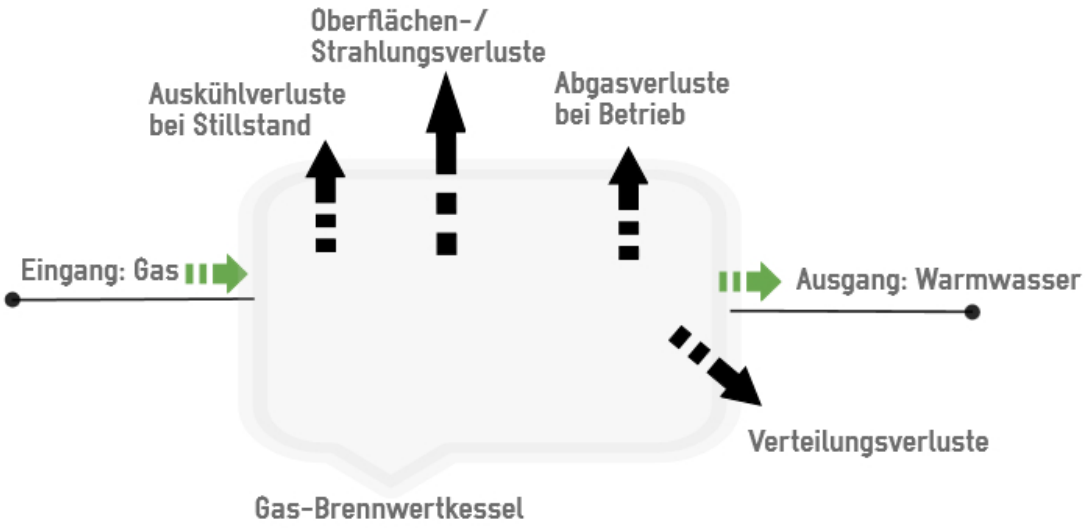

Abbildung 3: Gas-Brennwertkessel

Über den Eingang wird Energie aus dem Gasnetz eingespeist. Aus den unterschiedlichen Verlusten, bestehend aus Auskühlverlusten, Strahlungsverlusten und Abgasverlusten, ergibt sich ein Gesamtwirkungsgrad von 95 %. Die anfallenden Verluste kommen hauptsächlich durch Oberflächen- und Strahlungsverluste zustande.

4.9.1.2. Umsetzung in oemof

```
# create Gas-Brennwertkessel
  solph.LinearTransformer(
    label="Gas-Brennwertkessel",
    inputs={bgas: solph.Flow()},
    outputs={bwarm: solph.Flow(nominal_value=10000, variable_costs=10)},min={3000},
    conversion_factors={bwarm: 0.95})
```

Code 14: Gas-Brennwertkessel

Die maximale Nennleistung beschränkt den Durchfluss. Die variablen Kosten beschreiben die Kosten pro durchflossene Energieeinheit. Sie sind ein Faktor, der in Relation zu den Kosten der übrigen Objekte steht. Sofern der Gasbrennwert-Kessel eine minimale Teilzeitkraft vorweisen würde, könnte diese mit *min=()* ausgedrückt werden. Der Gesamtwirkungsgrad der Anlage wird mit dem *conversion_factor=()* festgelegt. Bei dieser Anlage ergeben sich keinerlei Probleme bei der Umsetzung in oemof.

4.9.2. Power-to-Gas

4.9.2.1. Allgemein

Es ist ein Begriff, der die Umwandlung von Strom mittels Elektrolyse zu Wasserstoff und eventuell anschließender Methanisierung zu einem Brenngas bezeichnet. Der Prozess der alkalischen Elektrolyse, also die Umwandlung von Wasser zu Wasserstoff und Sauerstoff mittels Zugabe von Strom, ist ein Teilprozess dessen, der aber auch nach diesem Schritt ein Ende finden kann. Denn reiner Wasserstoff kann dem Erdgas auch direkt hinzugefügt werden. Dabei ist auf die maximale Konzentration von 5 % Wasserstoff in Erdgas zu achten [27]. Andererseits kann Wasserstoff auch gut gelagert, transportiert und anschließend in industriellen Prozessen verarbeitet werden oder auch für Brennstoffzellen als Treibstoff seine Verwendung finden.

Bei Simulation größerer Anlagen, die in Relation zum vorhandenen Gas die 5 % übersteigen könnten, könnte man vorab eine Begrenzung des aktuellen Durchflusses festlegen, sofern zu jedem Zeitpunkt eine zehnmal so große Menge an Gas im Netz vorhanden ist.

Für die Elektrolyse, bei der etwa 24 % Verluste anfallen, wird Strom benötigt. Zusammen mit den Gleichrichter- und übrigen Kraftwerksverlusten ergibt sich ein Gesamtwirkungsgrad von etwa 61 %. Mit anschließender Methanisierung ergibt sich ein Gesamtwirkungsgrad von etwa 45 % [28]. Daraus ergibt sich folgender Ablauf der Energie:

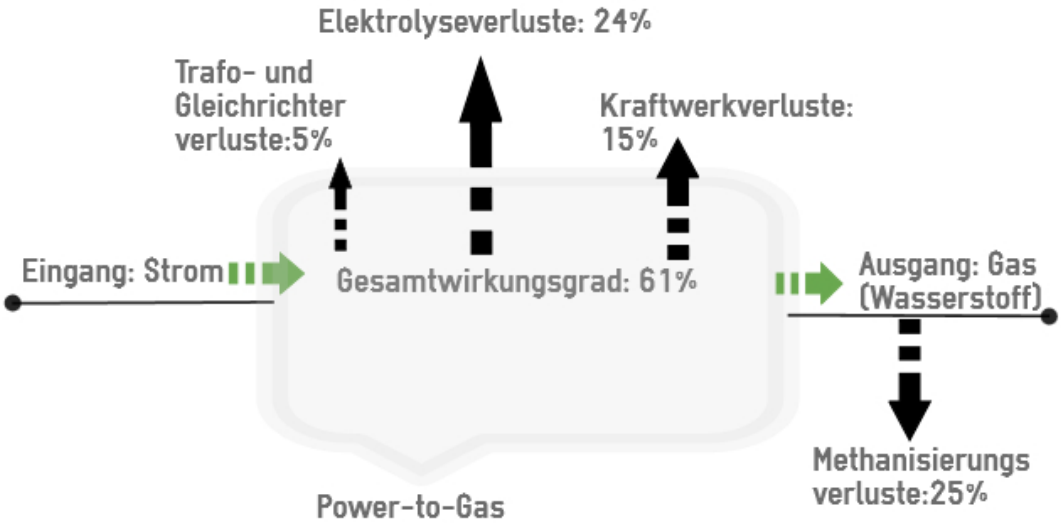

Abbildung 4: Power-to-Gas

Es ist also zu unterscheiden, ob man nur von der alkalischen Elektrolyse ausgeht oder den Prozess der anschließenden Methanisierung miteinbezieht, um problemlos ins vorhandene Gasversorgungsnetz einspeisen zu können. Die Methanisierung ist ein eigener chemischer Prozess, in dem Wasserstoff exotherm mit Kohlenstoffdioxid oder -monoxid zu Methan umgewandelt wird [29]. Aktuell gibt es erst eine Handvoll Elektrolyseanlagen in Deutschland mit einer Leistung von insgesamt 50 MW, die der Forschung und Entwicklung dienen und nicht wirtschaftlich arbeiten. Die größte Elektrolyseanlage mit einer Leistung von bis zu 6 MW steht in Mainz [30].

4.9.2.2. Umsetzung in oemof

Es wird in dem gezeigten Beispiel von einer alkalischen Elektrolyse ausgegangen. Die variablen Kosten entstehen durch Betrieb und Wartung der Anlage und stehen in Relation zu den variablen Kosten der anderen Anlagen/Objekten. Startkosten der Anlage wie auch die maximale Erhöhung der Anlage, die bei kleinen Zeitschritten durchaus relevant sein kann, können in oemof umgesetzt werden. Die definierten Werte der Eigenschaften der Anlage sind beispielhaft und stellen aufgrund nicht vorhandener Literaturnachweise nur eine Möglichkeit dar. Einzig der Umwandlungsfaktor in Höhe von 61% ist nach dem virtuellen Institut nachweisbar [28].

```
# create Power-to-Gas
   solph.LinearTransformer(
      label="Power-to-Gas",
      inputs={bel: solph.Flow()},
      outputs={bgas: solph.Flow(nominal_value=1000000, variable_costs=10)},
      min={200000}, positive_gradient={500000}, startup_costs={5000},
      conversion_factors={bgas: 0.61})
```

Code 15: Power-to-Gas

Wenn man zur Elektrolyse den Prozess der Methanisierung von Wasserstoff mit einbezieht, verringert sich der Wirkungsgrad der Gesamtanlage und die Kosten erhöhen sich. Ebenso muss die Nennleistung der Anlage an den maximalen Wert, der bei der Methanisierung umgesetzt werden kann, angepasst werden, wie auch der maximale Anstieg des Durchflusses in Abhängigkeit pro Zeitschritt (*positive_gradient={}*).

4.9.3. Heizstab

4.9.3.1. Allgemein

Ein Heizstab kann in verschiedenen Arten der Erwärmung dienen. Hier wird die Brauchwassererwärmung behandelt. Ein Heizstab dient dazu, entweder überschüssige elektrische Energie in Wärme eines Warmwasserspeichers umzuwandeln oder um bei Knappheit des Warmwasservorrats auszuhelfen. Der Wirkungsgrad beträgt annähernd 100 % aufgrund von Leitungsverlusten und wird daher mit 99 % in folgender Abbildung festgelegt:

Abbildung 5: Heizstab

4.9.3.2. Umsetzung in oemof

Ein Heizstab hat simple Eigenschaften, die sich in oemof umsetzen lassen. Eine gewählte Nennleistung begrenzt die Umwandlung der Energie. Folgender Aufbau in oemof wäre umsetzbar:

```
# create Heizstab
  solph.LinearTransformer(
    label="Heizstab",
    inputs={bel: solph.Flow()},
    outputs={bwarm: solph.Flow(nominal_value=1000, variable_costs=10)}, min={20},
    conversion_factors={bwarm: 0.999})
```

Code 16: Heizstab

Der Ausdruck *min={}* gibt die minimale Teilzeitkraft des Heizstabs an.

4.9.4. Elektro-Wärmepumpe

4.9.4.1. Allgemein

Bei einer Elektro-Wärmepumpe wird der Umwelt, damit kann Erde, Wasser oder Luft gemeint sein, Wärme entzogen. Diese gewonnene Wärme wird mit einer elektrischen Wärmepumpe befördert, auf ein nutzbares Temperaturniveau gehoben und an den Warmwasserspeicher oder direkt an die Heizung abgegeben. Da sie abhängig von der Umgebungstemperatur mit 30 % Energieeinsatz durch die Nutzung der Umgebungswärme 100 % Energie in Form von Wärme erzeugt, gilt sie als sehr emissionsarm und ökologisch – sofern die eingesetzte elektrische Energie nicht durch konventionelle Kraftwerke bereits einen geringen Wirkungsgrad durchlief und bei der Verbrennung von fossilen Brennstoffträgern große Mengen CO_2 freigesetzt worden sind. Das Verhältnis zwischen der Wärme, die dem Wärmekreis abgegeben wird, zu der eingesetzten Energie nennt man Leistungszahl. Sie wird als COP abgekürzt (Coefficient of Performance).

Dieses Verhältnis zwischen Einsatz und Erzeugung ist stark temperaturabhängig. Bei geringen Temperaturunterschieden, wie zum Beispiel im Sommer, wo wenig Energie benötigt wird, wird ein Verhältnis von 1:3 und temporär sogar noch höher erreicht. Im Winter jedoch, wo ein hoher Wärmebedarf besteht, ist das Verhältnis weitaus geringer [31].

Auch wenn die Energiegewinnung aus der Umgebung nicht als unendlich angesehen werden kann, spricht man von einer Leistungszahl von 3 [32].

Mit dem eingehenden Strom wird mit einem Faktor von 2–5 Umweltwärme gefördert, die Energie wird als Wärme ausgegeben.

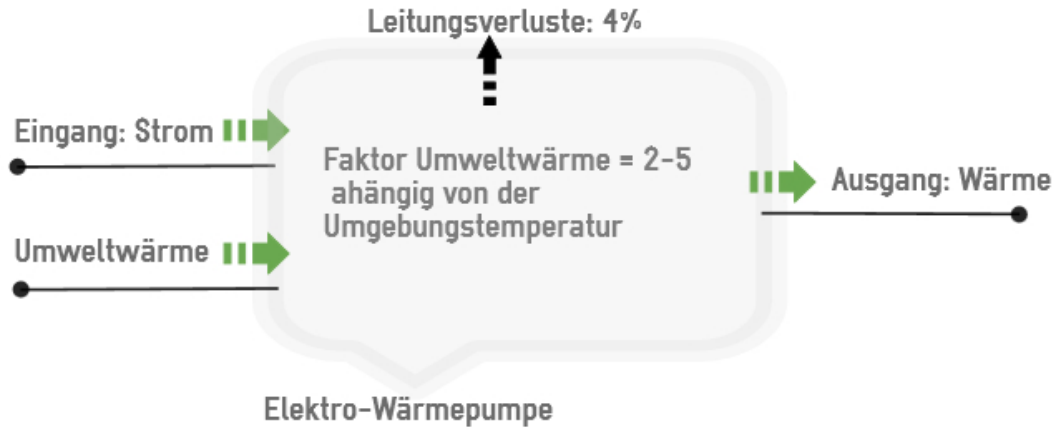

Abbildung 6: Elektro-Wärmepumpe

Der Faktor, mit dem die Umweltwärme gefördert wird, ist abhängig von der Umgebungstemperatur und wird mit der Leistungszahl angegeben.

4.9.4.2. Umsetzung in oemof

```
# create Elektro-Wärmepumpenheizung
  solph.LinearTransformer(
    label="Elektro-Wärmepumpenheizung",
    inputs={bgel: solph.Flow()},
    outputs={bwarm: solph.Flow(nominal_value=3000, variable_costs=5)}, min={300},
    conversion_factors={bwarm: data['temperaturabhängige Wirkungsgrad'] })
```

Code 17: Elektro-Wärmepumpe

Unterschiedliche Umgebungstemperaturen wirken sich auf den Umwandlungsfaktor der Wärmepumpe aus. Diese Abhängigkeit ist in oemof mit einer Einbindung einer vorab angefertigten Tabelle möglich, in der für jeden Zeitschritt ein zugehöriger Faktor des Wirkungsgrads festgelegt wird. Die minimale Teilzeitkraft wird auf 10 % der Nennleistung festgelegt und kann je nach Anlage variieren. Es lassen sich somit alle Eigenschaften einer Elektro-Wärmepumpe in oemof darstellen.

4.9.5. Nachtspeicherheizung

4.9.5.1. Allgemein

Die Nachtspeicherheizung war früher dafür vorgesehen, sich in Schwachlastzeiten zu einem günstigen Tarif mit Strom über Heizstäbe aufzuheizen und die Wärme dann kontinuierlich abzugeben. Da der dafür produzierte Strom zumeist aus großen Kraftwerken entstand, ergab sich im Vergleich zu den brennstoffbetriebenen Warmwasserheizungen ein größerer Energieverlust für die Umwelt aufgrund des geringen Wirkungsgrades der konventionellen Kraftwerke. Dadurch wurde um die Jahrtausendwende der Bestand an Nachtspeicherheizungen, auch durch finanzielle Anreize, reduziert [33]. In Zeiten der Energiewende, in der ein temporärer Überschuss an elektrischer Energie häufig vorkommen kann und vor allem elektrische Speicher noch zu teuer sind, bieten Nachtspeicherheizungen eine mögliche Alternative, Lasten zeitlich zu verschieben und günstig Energie zu speichern [34]. Dennoch sind Nachtspeicherheizungen aufgrund des ökologischen Aspekts der Ineffizienz wenig attraktiv.

Abbildung 7: Nachtspeicherheizung

Es ergibt sich ein Wirkungsgrad von annähernd 100 %.

4.9.5.2. Umsetzung in oemof

Grundsätzlich wird in einer Nachtspeicherheizung Strom in Wärme umgewandelt. Klassische Nachtspeicherheizungen gehören zu den elektrischen Lasten innerhalb eines Hauses. Wenn diese innerhalb einer Simulation also als Umwandlungsstelle aufgebaut werden soll, dann darf die Heizung nicht als elektrischer Verbraucher zählen, der im elektrischen Lastprofil eingegliedert ist, sondern muss als Wärmeverbraucher angegeben sein und in das Wärmelastprofil integriert sein. Neben der Umwandlung der Energieform muss man die Nachtspeicherheizung in oemof auch als einen Speicher ansehen. Denn nur unter der Klasse „Storage“ lässt sich ein Objekt aufbauen, welches eine Kapazität besitzt, aufgeladen

und entladen werden kann. Die Nachtspeicherheizung könnte also ebenso unter dem Begriff Speicher gelistet werden. Da in dieser Arbeit vor allem Koppelstellen relevant sind und die Nachtspeicherheizung eine weitverbreitete Anlage ist, die Energie in Form von Strom zu Wärme umwandelt, fällt sie hier in die Kategorie Umwandlungsstelle. Der Vorteil hierbei ist, dass die Energiesektoren getrennt werden.

```
# create Nachtspeicherheizung
  solph.Storage(
    label='Nachtspeicherheizung',
    inputs={bel: solph.Flow(variable_costs=10)},
    outputs={bwarm: solph.Flow()},
    capacity_loss=0, initial_capacity=0, nominal_value=3000, capacity_max=3000,
    nominal_input_capacity_ratio=1/2, nominal_output_capacity_ratio=1/2,
    inflow_conversion_factor=1, outflow_conversion_factor=0.99)
```

Code 18: Nachtspeicherheizung

Abhängig von der gewählten Dauer eines Zeitschritts spielt der maximale Durchfluss abhängig zur maximalen Kapazität eine Rolle und kann mit der *nominal_input_capacity_ratio* auf einen Wert festgelegt werden. Ebenso der abnehmende Wert, da die Heizung auch bei voller Heizkraft nicht innerhalb einer möglicherweise sehr klein gewählten Dauer der Zeitschritte (zum Beispiel 15 Minuten) die gesamte Energie abgeben kann.

Die Nachtspeicherheizung lässt sich aufgrund ihrer einfachen Eigenschaften und der einheitenlosen Darstellungsweise in oemof gut simulieren. Es werden keine Werte vernachlässigt. Eine durch eine niedrigere Umgebungstemperatur verursachte Erhöhung der Wärmelast spielt für die Anlage und deren Eigenschaften an sich keine Rolle, da der Wirkungsgrad immer gleichbleibend bei nahezu 100 % liegt.

4.9.6. Kraft-Wärme-Kopplung

4.9.6.1. Allgemein

Bei der Kraft-Wärme-Kopplung, kurz KWK, wird zu der konventionellen Stromerzeugung durch Verbrennung eines Energieträgers zusätzlich die Abwärme zur Wärmegewinnung genutzt. Somit lässt sich der Wirkungsgrad der Anlage im Vergleich ohne Nutzung der Abwärme um etwa 30 % steigern [35]. Solche Anlagen werden einerseits als Heizkraftwerk im gewerblichen Sinne genutzt, aber seit einigen Jahren gewinnen kleinere sogenannte Blockheizkraftwerke in privaten Häusern zunehmend an Bedeutung. Da hier mit weniger Anteilen an fossilen Brennstoffen die gleiche Menge an Energie gewonnen werden kann, ist die KWK im Vergleich zur konventionellen Verbrennungsmethode CO_2-sparend und umweltfreundlicher [36].

Der Energiefluss stellt sich folgendermaßen dar:

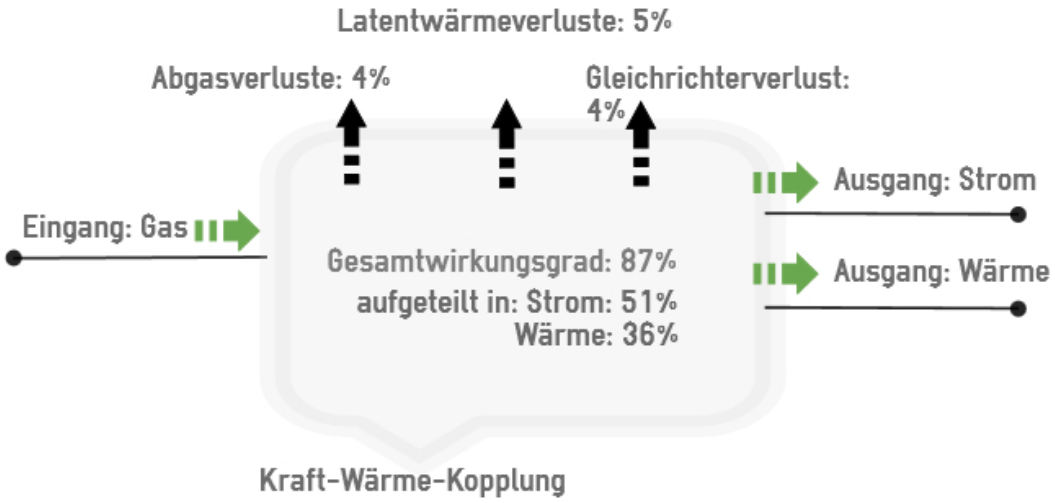

Abbildung 8: Kraft-Wärme-Kopplung

Das eingehende Gas wird verbrannt, die erwärmte Luft treibt einen Stromgenerator an. Dieser Generator wandelt 51 % der Energie in Strom um. Durch zusätzliche Nutzung der bisher im Prozess nicht umgewandelten Abwärme als Fernwärme werden weitere 36 % der Energie genutzt. Durch die Abgas-, Latentwärme- und Gleichrichterverluste ergibt sich ein Gesamtwirkungsgrad von 87 %.

4.9.6.2. Umsetzung in oemof

Der Unterschied zu den bisherigen Umwandlungsstellen ist die gleichzeitige Ausgabe der Energieeinheiten in zwei Busse. Hierfür wird einfach ein weiterer Ausgang definiert. Der Gesamtwirkungsgrad muss anschließend getrennt für die beiden ausgebenden Busse angegeben werden. Als Größe der Nennleistung wird 10 kW ausgewählt, wie es für private Haushalte üblich ist [37]. Die Mindestlast beträgt 20 % [38].

```
# create KWK
   solph.LinearTransformer(
      label="Kraft-Wärme-Kopplung",
      inputs={bgas: solph.Flow()},
      outputs={bel: solph.Flow(nominal_value=10000, variable_costs=5)
            bwarm: solph.Flow(nominal_value=10000, variable_costs=5)},
      min={2000}, startup_costs={50},
      conversion_factors={bel: 0.51, belwarm: 0.36})
```

Code 19: Kraft-Wärme-Kopplung

4.9.7. Gas- und Dampfkombikraftwerk

4.9.7.1. Allgemein

Das GUD-Kraftwerk hat im Gegensatz zu anderen konventionellen Kraftwerken die positive Eigenschaft des Kaltstarts. Es ist in der Lage innerhalb von wenigen Minuten die volle Leistung an Strom bereitzustellen. Aufgrund dieser Eigenschaften ist es wichtig, bei der immer größer werdenden stark schwankenden Einspeisung erneuerbarer Energien in das Stromversorgungsnetz, die Schwankungen in kurzer Zeit auszugleichen. Der Wirkungsgrad liegt bei moderner Bauweise bei etwa 60 %, die sich aus 40 % Wirkungsgrad der Stromerzeugung der Gasturbine und aus circa 20 % der Nutzung der Abwärme zusammensetzen [39].

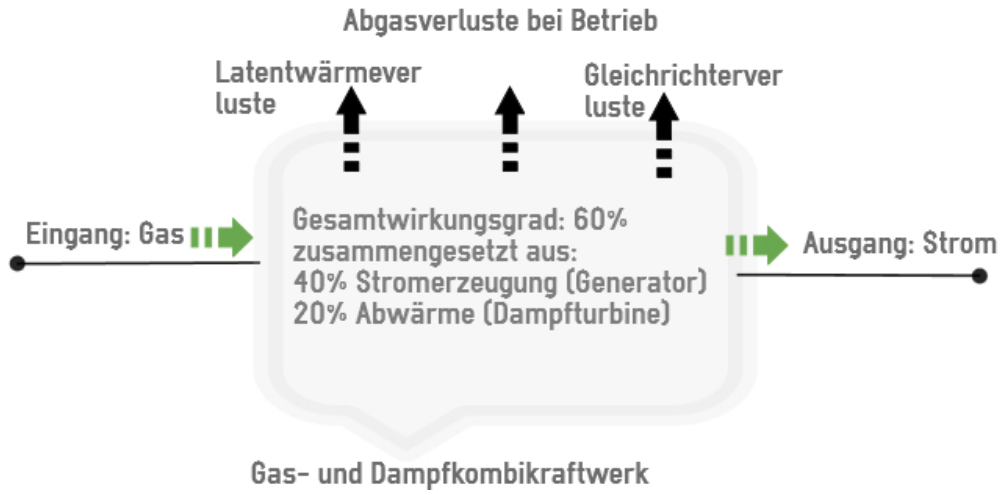

Abbildung 9: Gas- und Dampfkombikraftwerk

Gas wird mit Luft gemischt in einer Gasturbine verbrannt, hier ergibt sich bei der Stromgewinnung ein Wirkungsgrad von 40 %. Es entstehen Latentwärme- und Abgasverluste. Die Abwärme im Abhitzekessel wird genutzt, um einen Wasserkreislauf zu erhitzen. Der Wasserdampf treibt eine Dampfturbine an. Hierbei werden weitere 20 % der Energie in Form von Strom gewonnen.

4.9.7.2. Umsetzung in oemof

Es können wie bei einer KWK-Anlage problemlos zwei Ausgänge definiert werden:

```
# create Gaskraftwerk
  solph.LinearTransformer(
    label="GuD-Kraftwerk",
    inputs={bgas: solph.Flow()},
    outputs={bel: solph.Flow(nominal_value=100000, variable_costs=1)
          bwarm: solph.Flow(nominal_value=100000, variable_costs=1)},
    min={1000}, positive_gradient={0}, startup_costs={1000},
    conversion_factors={bel: 0.40, bwarm: 0.19})
```

Code 20: GuD-Kraftwerk

Die Kaltstarteigenschaft lässt sich mit *positive_gradient={}* darstellen. Hier ist auf die verwendeten Zeitschritte zu achten, um diesen auf die maximale Erhöhung pro Zeitschritt herunterrechnen zu können. Vor allem bei Zeitschritten <5 min ist hierauf zu achten. Da das Ans-Netz-Gehen bei größeren Kraftwerken mit weiteren Kosten verbunden ist, könnten Startkosten festgelegt werden. Diese Kosten unterscheiden sich von Anlage zu Anlage, sodass nur ein fiktiver Wert angegeben wird. Die Teillastfähigkeit wird durch die Festlegung *min()*= definiert. Die Wirkungsgrade der beiden ausgehenden Busse werden wie gehabt mit dem *conversion_factor* festgelegt. Alle wichtigen Eigenschaften können somit dargestellt werden.

5. Ablauf zur Berechnung und Ausgabe von Ergebnissen

5.1. Berechnung des Energieflusses

Mit den vorherigen vorgestellten Bussen und Objekten, denen verschiedene Eigenschaften zugeordnet werden, können Energieversorgungsnetze aufgebaut werden. Neben festen Werten können den Bussen und Objekten berechenbare Größen wie zum Beispiel der Durchfluss zugeordnet werden. Es können aber auch andere Größen, wie eine optimale Größe der Nennleistung, berechnet werden. Die Berechnung zur Optimierung ist mit dem Programmbefehl *solph.()* umsetzbar. Dieser Befehl ist grundsätzlich dem Flow, also der aktuellen Leistung des Objekts, zugeordnet. Dies ist die Eigenschaft, nach der das Programm das Energiesystem hin optimiert. Somit wird der optimale Energiefluss berechnet, indem die Gesamtkosten des Energiesystems am geringsten sind. Daher sind allen Objekten, die berechnet werden sollen, der Befehl *solph.()* dem *Flow* zugeordnet: *solph.Flow() [10]*.

Der optimale Energiefluss wird dann unter Berücksichtigung der zugeordneten Charakterisierungen der Objekte durchgeführt. Die Optimierung beruht also darauf, die Gesamtkosten der verbrauchten Energie möglichst gering zu halten. Für die Lösung dieser linearen Optimierung wird in oemof der CBC-Solver verwendet. Im darauffolgenden Kapitel wird auf die genaue Vorgehensweise des Solvers eingegangen.

Hier kann aber auch, statt das System auf Kosten zu optimieren, beispielsweise der CO_2-Wert optimiert werden. Dafür müssen anstelle der Kosten die Werte der variablen und fixen Kosten einfach als CO_2-Wert pro Energieeinheit definiert werden.

Der Befehl *solph.* kann ebenso auf weitere Eigenschaften wie zum Beispiel den Nennwert einer Umwandlungsstelle angewendet werden. Damit könnte man die optimale Größe einer Anlage bestimmen. Auch wenn dies nicht mit allen Umwandlungsstellen oder anderen Elementen ohne Weiteres möglich ist, nur den Nennwert als frei berechenbar anzusehen, so ließe sich hiermit etwa die benötigte Leistung einer Photovoltaikanlage bestimmen, die dann den Verbraucher mit seinen hinterlegten Lastprofilen sättigen würde.

5.2. Ausgabe von Ergebnissen

Die Ergebnisse einer jeden Berechnung lassen sich in oemof einzeln ausgeben: Mit der Klasse „outputlib“ können die gewünschten Eigenschaften, wie zum Beispiel die Summe der Leistung über alle Zeitschritte, addiert oder die maximale temporäre Leistung einer Anlage ausgegeben werden:

```
def get_result_dict(energysystem):
  logging.info('Ergebnisse anzeigen')
  myresults = outputlib.DataFramePlot(energy_system=energysystem)
  Beispielobjekt = myresults.slice_by(obj_label='Beispielobjekt', type='to_bus',
                    date_from='2016-01-01 00:00:00',date_to=2012-01-31 00:00:00)
 return {'Beispielobjekt_sum': Beispielobjekt.sum(),'Beispielobjekt_max': Beispielobjekt.max(),
      'objective': energysystem.results.objective}
```

Code 21: Ergebnisse ausgeben

Es ist darauf zu achten, dass die zeitliche Angabe mit der Anzahl der Zeitschritte der eingelesenen Leistungs- und Lastprofilen übereinstimmt. Die Berechnungsdauer, die der Solver für das Lösen benötigt, wird in Sekunden im Ausgabefeld in Python angegeben. Die Ergebnisse des optimalen Flusses für das gewählte Energiesystem lassen sich entweder als .csv-Datei in Tabellenform ausgeben oder man wählt mit dem folgenden Programmcode direkt eine graphische Auswertung:

```
def create_plots(energysystem): logging.info('Plot the results')
  cdict = {'Beispielobjekt': '#555555'}
#Farbliste hinzufügen
  colorlist = myplot.color_from_dict(cdict)
# Plotte Beispielbus an dem Beispielobjekt angeschlossen ist
  myplot = outputlib.DataFramePlot(energy_system=energysystem)
  myplot.slice_unstacked(bus_label="Beispielbus", type="to_bus",
  date_from="2012-01-01 00:00:00", date_to="2012-01-31 00:00:00")
#Graphische Darstellung bestimmen
  myplot.plot(color=colorlist, linewidth=2, title="Januar") myplot.ax.legend(loc='upper right')
  myplot.ax.set_ylabel('Leistung in kW') myplot.ax.set_xlabel('Tage')
  myplot.set_datetime_ticks(date_format='%d-%m-%Y', tick_distance=24*7)
```

Code 22: Graphische Auswertung darstellen

Für die graphische Darstellung der Ergebnisse wird dem ausgewählten Objekt zuerst eine Farbe zu geordnet. Anschließend werden die geforderten Daten für die Darstellung eingelesen. Dem Graphen können Farbe, Achsenbeschriftung, Liniendicke zugeordnet werden. Um die Ergebnisse in Tabellenform auszugeben, wird folgender Programmcode aufgerufen:

```
results = outputlib.ResultsDataFrame(energy_system=esys

    results.bus_balance_to_csv()
```

Code 22: Graphische Auswertung darstellen

6. Lösungsverfahren

Die in oemof aufgebauten Energiesysteme stellen ein mathematisches lineares Gleichungssystem dar. Zur Lösung dieser Gleichungen kann ein Solver frei ausgewählt werden. Grundsätzlich ist der CBC-Solver für das Lösen in oemof integriert.

6.1. CBC-Solver

Die in den Szenarien konstruierten Optimierungsprobleme werden mit dem CBC-Solver gelöst. Dieser Solver basiert auf Open Source und stammt aus dem COIN-Or Framework (Co15). Der Solver arbeitet nach dem Simplex-Verfahren, einem Optimierungsverfahren linearer Optimierungsprobleme [40].

6.2. Alternativer Solver: Gurobi

Der Gurobi-Solver ist ein kommerzieller Löser für mathematische Programmierung. Gurobi bietet neben matrixorientierten Schnittstellen wie C oder Matlab auch objektorientierte Schnittstellen zu C++, Java und eben auch Python. Durch die vielseitigen Schnittstellen und einer Auswahl an neuesten Algorithmen gilt Gurobi als der beste kommerzielle Solver aktuell [41]. Trotz der kommerziellen Nutzung lässt sich im akademischen Umfeld eine kostenfreie Lizenz anlegen, somit ist Gurobi als Solver für Optimierungsprobleme eines in oemof aufgebauten Energiesystems an der Technischen Hochschule Köln einsetzbar. Nachteil der kommerziellen Nutzung ist, dass die Rechenwege des Solvers nicht nachvollziehbar sind.

Zu der Einbindung des Gurobi-Solvers ist vor allem bei der Optimierung von komplexeren Energiesystemen geraten, da die Berechnungsdauer bei verschiedenen Energiesystemen gegenüber des CBC-Solvers um circa 50 % verkürzt werden konnte [42].

7. Versuchsszenarien

7.1. Einleitung

In diesem Kapitel werden verschiedene Szenarien, die zur Prüfung der Funktionalität von oemof dienen, beschrieben, erläutert, aufgebaut und die Ergebnisse anschließend diskutiert. Damit sollen bisher getroffene Annahmen bestätigt werden und oemof auf die Simulation und Optimierung von Kopplungsstellen hin überprüft werden. Dafür werden in diesem Kapitel der Aufbau von Leitungen, die Berechnungsdauer, die Optimierung des Standortes einer Kopplungsstelle und ein Vergleich zu einer Simulation einer Wärmepumpe durch ein Excel-Tool betrachtet. Bei allen durchgeführten Versuchsszenarien wird bei der Berechnungsdauer die Rechenleistung eines Intel i5-4670k Prozessors in Anspruch genommen. Bei Verwendung anderer Prozessoren kann es zu abweichenden Berechnungszeiten des Programms kommen.

7.2. Szenario 1: Leitungsaufbau

7.2.1. Beschreibung

In dem ersten Szenario wird die Darstellung von Leitungen, wie in Kapitel 4.5 beschrieben, in oemof erläutert, durchgeführt und anschließend mit einem Aufbau ohne Leitungen verglichen. Dies dient der Überprüfung der dort gemachten Annahmen, um festzustellen, dass Leitungselemente, die in oemof nicht als separates Modul verfügbar sind, dargestellt werden können.

Für das Szenario wird eine einfache Umgebung mit zwei Elektrizitäts- und einem Gas-Bus aufgebaut. In den elektrischen Bus 1 wird Energie aus einer Windkraftanlage Wind2 eingespeist und hat einen *sink,* um mögliche überschüssige Energie aus dem System abzugeben. Dieser Bus gilt als ein Standpunkt innerhalb eines einfachen Energiesystems. Der zweite Standpunkt des elektrischen Energiesystems ist der elektrische Bus 2. Er ist mit den Leitungen L12 und L21 mit dem elektrischen Bus 1 verbunden. Die Leitungselemente haben einen Nennwert von 500 und einen fiktiven Verlust von 50 %. Dieser dient nur dazu, die zu erwartenden Ergebnisse einfach darzustellen. Der elektrische Bus 2 wird gespeist von einer PV- sowie einer Windkraftanlage, ist verbunden mit einem Batteriespeicher, kann überschüssige Energie über einen *sink* abgeben und wird über ein Gaskraftwerk vom Gas-

Bus aus ebenso mit Energie versorgt. Ein Gasvorrat speist endlos in den Gas-Bus ein. Da die Kosten pro durchflossener Energieeinheit des Gaskraftwerks in Relation zu den Kosten der übrigen Energieeinspeiser hoch angesetzt sind, wird erwartet, dass der Verbraucher nur vom Gaskraftwerk versorgt wird, sofern die übrigen Erzeuger nicht genug Energie bereitstellen. Als Last- und Leistungsprofile werden fiktive stundenweise Werte zweier Tage eingelesen, um die Ergebnisse einfach interpretieren zu können.

Anschließend wird der Versuch so abgeändert, dass der elektrische Bus 1 wegfällt und das Objekt Wind2 direkt in den elektrischen Bus 2 einspeist. Dadurch entfallen die Leitungselemente und es gibt nur noch einen elektrischen Bus.

7.2.2. Übersicht mit Leitung

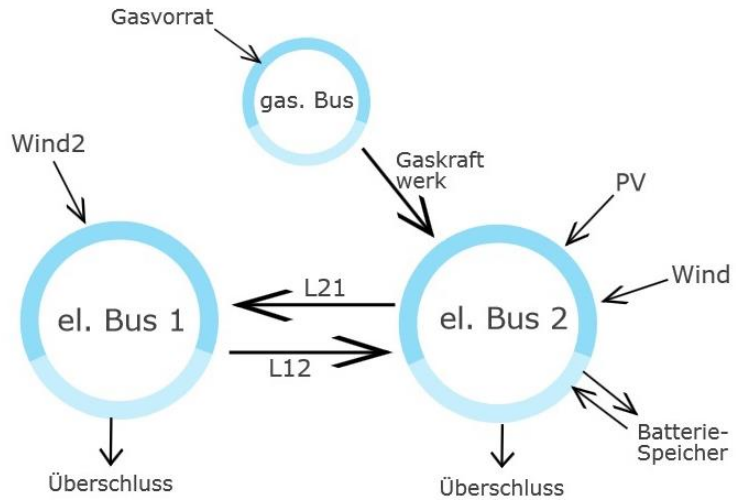

Abbildung 10: Visualisierung V1 mit Leitung

7.2.3. Übersicht ohne Leitung

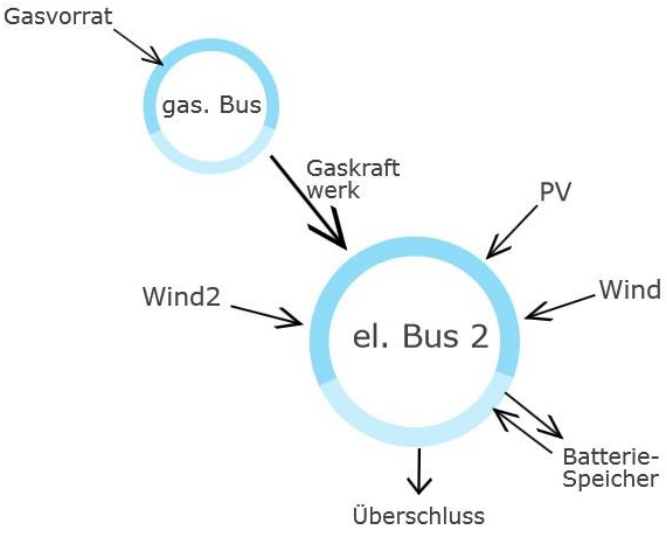

Abbildung 11: Visualisierung V2 ohne Leitung

Die vollständigen Programmcodes zu den visualisierten Übersichten befinden sich im Anhang auf S. 68–75.

7.2.4. Ablauf

Gelöst werden soll der optimale Energiefluss bezogen auf die Kosten unter Einhaltung der Eigenschaften aller Objekte. Dafür wird jeweils der Durchfluss als Lösungsvariable angegeben mit dem Befehl „solph.Flow“. Das Programm liest im Hintergrund die einzelnen Werte der Zeitschritte ein und baut damit ein Gleichungssystem auf, was vom CBC-Solver anschließend gelöst wird.

Bei einer Auswahl von 8.760 Zeitschritten, was gegenüber den ausgewählten 48 Zeitschritten stundenweise betrachtet einem Jahr entspricht, erhöht sich die Berechnungszeit von Bruchteilen einer Sekunde auf über 30 Sekunden.

7.2.5. Auswertung

Das Optimierungsproblem wurde innerhalb von Bruchteilen einer Sekunde gelöst. Verglichen mit einer Auswahl von 8.760 Zeitschritten, was stundenweise betrachtet einem Jahr entspricht, gegenüber den ausgewählten 48 Zeitschritten erhöht sich die Berechnungszeit schon auf über 40 Sekunden. In Abbildung 18 sind die Ergebnisse des Versuchsszenarios 1 mit Leitung graphisch dargestellt:

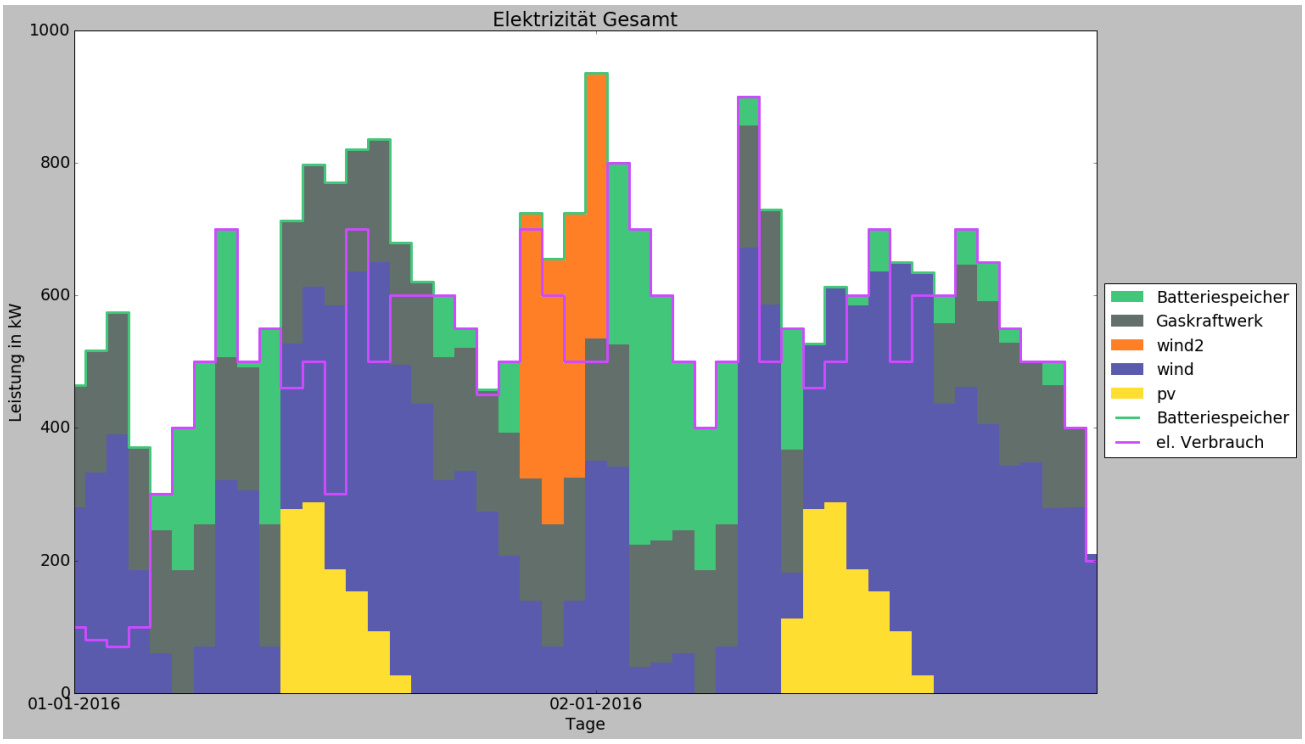

Abbildung 12: Graphische Auswertung V1 mit Leitung

Hier sind alle mit dem elektrischen System verbundenen Objekte dargestellt. Die rosa Linie zeigt den Verbrauch. Der orangene Teil ist die Energie, die über die Leitung L12 eingespeist wurde. Das Objekt Wind2, welches in den elektrischen Bus 1 einspeist, hat zwar einen Nennwert von 5.000 und hat für vier Stunden innerhalb des Zeitrahmens ein Leistungsprofil zwischen 0.3 und 0.6, wird aber aufgrund der Begrenzung der Leitung L12 mit einem Nennwert von 400 an der Weitergabe der Energie beschränkt. Damit ist die Begrenzung einer Leitung nachgewiesen.

Im Vergleich dazu die graphische Darstellung des Versuchsszenarios ohne Leitung in Abbildung 13:

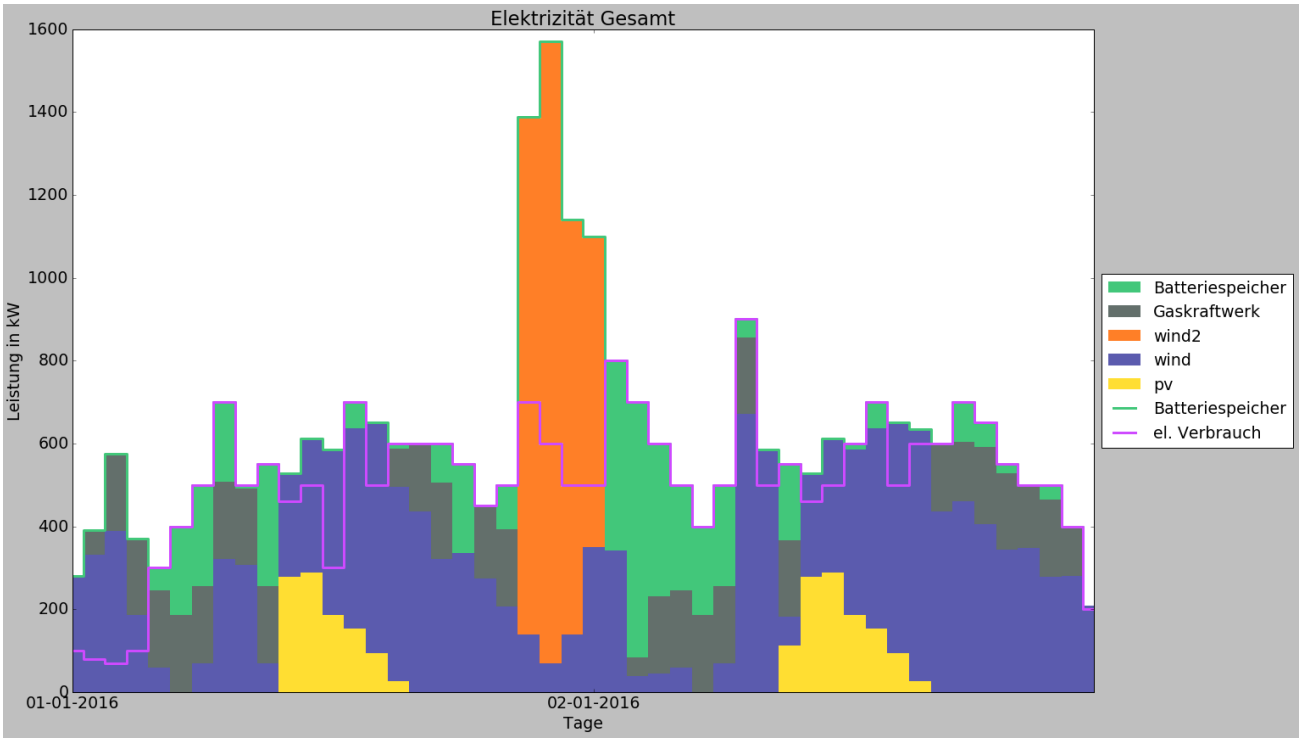

Abbildung 13: Graphische Auswertung V1 ohne Leitung

Hier wird nun die volle Leistung des Objekts Wind2 direkt in den elektrischen Bus 2 ohne Beschränkung eingespeist. Der Überschuss an Energie lädt den Batteriespeicher auf. Dieser kann die Energie in den Folgestunden an den Verbraucher abgeben. Während der Einspeisung von Wind2 lädt sich der Batteriespeicher auf, weshalb die Energie zu diesem Zeitpunkt die des elektrischen Verbrauchs übersteigt. Beim Vergleichen des maximalen Durchflusses und der Summe aller Zeitschritte der Objekte ergibt sich folgende Tabelle:

Tabelle 7: V1: Summe und Maxima der Objekte

mit Leitung	mit Leitung	ohne Leitung
Gaskraftwerk Summe	7.382	4.629
maximaler Verbrauch	900	900
Verbrauch Summe	24.270	24.270

PV maximale Leistung	288	288
PV Summe	2.163	2.163
Batteriespeicherdurchfluss	3.027	5.817
Wind2 maximale Leistung	3.000	1.500
Wind2 Summe	9.000	4.500
Wind maximale Leistung	672	672
Wind Summe	14.033	14.033
L12 maximaler Durchfluss	400	
L12 Summe	1.600	
L21 maximaler Durchfluss	0	
L21 Summe	0	

Mit Leitungsaufbau muss das Gaskraftwerk mehr Energie bereitstellen, da durch das Leitungselement die Energie des Objekts Wind2 gedrosselt und mit einem Umwandlungsfaktor von 0.5 der Leitung fließt. Die maximale Leistung und die Summe des Objekts2 weisen unterschiedliche Werte auf, da im Versuchsaufbau ohne Leitungen der Nennwert der Anlage halbiert wurde, um die Relation der Ergebnisse nicht zu weit auseinandergehen zulassen. Die Leitung L21 hat einen Durchfluss von 0, da im elektrischen Bus 1 keine Verbraucher angeschlossen sind, die eventuell mit Energie versorgt werden müssten. Die Berechnungsdauer des Versuchsszenarios lag bei jeweils 40 Sekunden. Bei den Berechnungen wird die Rechenleistung eines Intel i5-4670k Prozessors in Anspruch genommen.

7.2.6. Fazit

Leitungselemente, die grundsätzlich in oemof nicht als separate Module verfügbar sind, können mit dem Bauteil Umwandlungsstelle dargestellt werden. Damit ist ein, wenn auch vereinfachter, Aufbau von Leitungselementen in oemof wie in Kapitel 4.5 beschrieben umsetzbar. Dies ist grundlegend für den Aufbau und die Simulation von Energieversorgungssystemen.

Bei der Darstellung der Leitungen gelten bei der Auswahl der Eigenschaften die Möglichkeiten einer Umwandlungsstelle. Die Leistungsbegrenzung und ein Verlustfaktor zwischen zwei Standorten innerhalb eines Energiesystems lassen sich somit darstellen.

7.3. Versuchsszenario 2: Berechnungsdauer

7.3.1. Einleitung

Oemof nutzt zum Lösen der Optimierungsprobleme den CBC-Solver. Dieser wird im Kapitel 6.1 genauer beschrieben. Da es ein einfacher, frei verfügbarer Rechner für lineare Probleme ist, muss man von längeren Berechnungszeiten ausgehen. In diesem Versuchsszenario soll überprüft werden, in welchem Umfang bei komplexeren Aufbauten der Energiesysteme die Berechnungszeit eine Rolle spielt, ob zukünftig bei komplexeren Rechnungen dieser Solver überhaupt dem Umfang gewachsen ist und von welchen Komponenten des Energiesystems die Berechnungszeit hauptsächlich abhängig ist. Abschließend wird ein Mittelspannungsausschnitt simuliert, der einen Umfang aufweist, der innerhalb des ES-Flex-Infra Projektes relevant sein könnte.

7.3.2. Beschreibung

Dafür wird eine Umgebung mit mehreren Komponenten aufgebaut und der Umfang sowie Berechnungszeit verglichen mit dem simplen Versuchsaufbau aus dem Versuchsszenario 1. Hierfür wird nun von stündlich hinterlegten Last- und Leistungsprofilen mit einer Dauer von einem Jahr gerechnet. Diese Dauer der Simulation ist nötig, da nur so die saisonalen Schwankungen der erneuerbaren Energien vollständig dargestellt werden können, was bei zukünftigen Berechnungen eine Rolle spielt.

Die Anzahl der verwendeten Komponenten wird in diesem Versuchsszenario stetig gesteigert. Die Vergleiche der Berechnungszeiten, die für die Berechnung des Energiesystems benötigt werden, pro Anzahl an Komponenten, geben einen Aufschluss darüber, mit welchen Berechnungszeiten man bei noch mehr verwendeten Bauteilen rechnen muss.

Die in dem Szenario ausgewählten Objekte sind von ihren Eigenschaften her willkürlich, geben jedoch in der Anzahl der angeschlossenen Objekte pro Standpunkt des Energiesystems ein realitätsnahes Abbild eines Energieversorgungssystems. Da die Berechnungszeit aus dem ersten Versuchsszenario 44 Sekunden benötigte, wurde der Aufbau dahin gehend erweitert, dass das elektrische Netz nun ein Ringnetz mit sechs Standpunkten darstellt und nicht mehr nur zwei Standpunkte hat.

Die Auswahl wurde getroffen, um einerseits die Berechnungsdauer gegenüber dem Aufbau aus dem ersten Versuchsszenario mit nun mehreren Standpunkten, sprich mehr Leitungen, zuvergleichen und um die Abhängigkeit der Berechnungsdauer beim Hinzufügen von

weiteren Verbrauchern an den ringförmig angeordneten Bussen im Energiesystem zu analysieren.

7.3.3. Übersicht

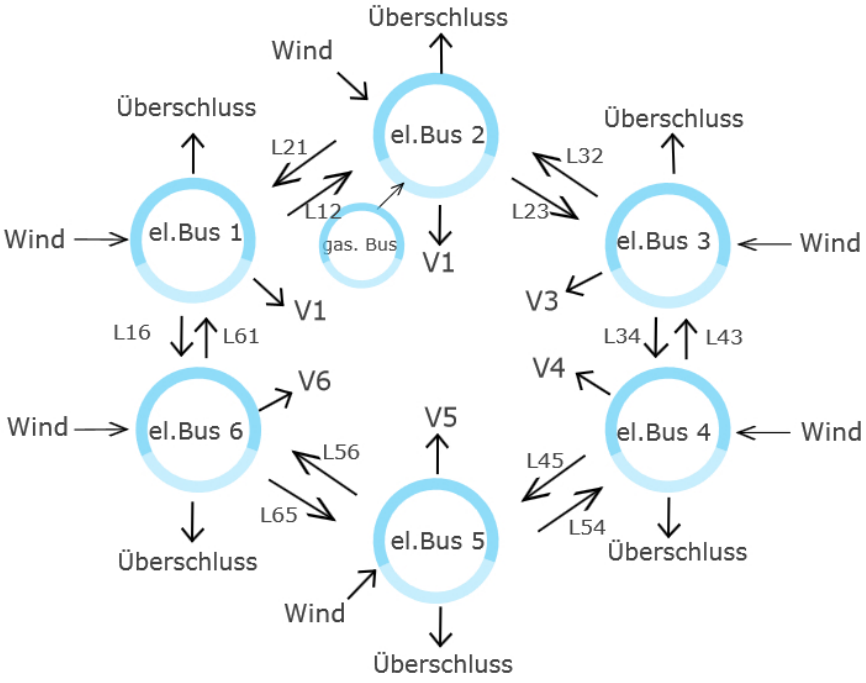

Abbildung 14: Visualisierung V2: Ringstruktur

Es wird ein Ringnetz mit sechs Standpunkten aufgebaut. Jeder Standpunkt dieses elektrischen Netzes hat einen Einspeiser, einen *sink*, um überschüssige Energie aus dem System abzugeben, und der elektrische Bus 2 wird wie im vorherigen Versuch von einer Gasquelle über ein Gaskraftwerk versorgt. Die Verbraucher V1 bis V6 sind mit dem jeweiligen elektrischen Bus verbunden. Die Anzahl der Verbraucher wird innerhalb dieses Versuchs von einem bis auf sechs gesteigert, um verschiedene Berechnungszeiten zu ermitteln. Zu Beginn ist also nur V1 angeschlossen und die restlichen Verbraucher sind nicht in das System eingegliedert. Nacheinander werden die weiteren verbraucher an die jeweiligen elektrischen Busse angeschlossen und jeweils die Berechnungsdauer ermittelt.

Es ergeben sich hieraus die Anzahl an sieben Bussen, neun energieeinspeisenden Objekten, sechs Optionen, überschüssige Energie abzugeben, zwölf Leitungselemente und eine variable Anzahl von bis zu sechs Verbrauchern, die jeweils an einem Standpunkt des elektrischen Netzes angeschlossen sind. Der vollständige Programmcode zu der visualisierten Übersicht befindet sich im Anhang auf S. 76–83.

Der vereinfachte Aufbau aus dem ersten Versuchsszenario hat eine verringerte Gesamtanzahl von 2 statt 7 Bussen, die Anzahl der Umwandlungsstellen ändert sich durch das Entfernen der Leitungen von 13 auf 1 (Gaskraftwerk bleibt erhalten) und die Anzahl an Energieeinspeisern ändert sich von 7 auf 2.

7.3.4. Auswertung

Die Berechnungszeit des vereinfachten Aufbaus aus Versuchsszenario 1 beträgt 00:44 Minuten. Mit einem angeschlossenen Verbraucher des in Abbildung 14 visualisierten ringförmigen Aufbaus ergibt sich eine Berechnungszeit von 00:45 Minuten. Die Anzahl an Objekten wie Busse, Einspeisern und Verbindungselementen in Form von Leitungen verändert die Berechnungszeit also kaum. Im Gegensatz dazu erhöht sich die Berechnungszeit beim Anschließen von weiteren Verbrauchern im elektrischen Bus:

Tabelle 8: V2: Berechnungsdauer pro Anzahl an Verbrauchern

Anzahl an Verbrauchern	Berechnungszeit in min
1	00:45
2	00:58
3	01:15
4	01:35
5	01:58
6	02:29

Die Berechnungsdauer nimmt pro weiteren angeschlossenen Verbraucher stetig zu. Zu Beginn bei nur einem angeschlossenen Verbraucher liegt die Berechnungsdauer bei 00:45 Minuten. Bei weiteren zugeschalteten Verbrauchern nimmt die Berechnungszeit schwach exponentiell zu. Bei sechs Verbrauchern ergibt sich eine Berechnungszeit von 2:29 Minuten.

Sobald mehrere Verbraucher gesättigt werden müssen, können sich die Wege zur Versorgung kreuzen und es ergeben sich für das Programm mehrere Möglichkeiten der Verteilung der Energieeinheiten. Dies führt zur Erhöhung der Berechnungszeit.

Die Anzahl der Verbraucher in einem Energiemodell in oemof ist also ausschlaggebender für die Berechnungsdauer als die Anzahl an Objekten. Die Anzahl der Busse, Leitungen und energieeinspeisenden Objekte sorgt zwar für einen leicht erhöhten Rechenaufwand, dieser ist aber sehr gering im Gegensatz zu dem Hinzufügen weiterer Verbraucher. Die Busse und Leitungselemente besitzen fest definierte Werte, die den Energiefluss begrenzen, aber erst die Verbraucher mit ihren Lasten, die auf unterschiedliche Wege gesättigt werden können, steigern die Rechenzeiten.

Beim Testen vom Hinterlegen von 15-minütigen statt stündlichen Werten für die Leistungs- und Lastprofile ergab sich jeweils eine etwa viermal so lange Berechnungsdauer.

Um die Berechnungszeit einer Simulation, die innerhalb des Forschungsprojektes ES-Flex-Infra stattfinden könnte, besser einzuschätzen, wird die Ringstruktur aus dem Versuchsszenario von 6 auf 20 Busse gesteigert und zusätzlich ein ebenfalls ringförmiges Fernwärmenetz integriert, was an jedem der 20 Standorte eine Verbindung zum Stromnetz aufweist. Dieser Aufbau steht in Referenz zu einem Strahlennetz mit einer geschlossenen Trennstelle, wie es in der Mittelspannungsebene oft vorkommt [43]. Um alle drei Ebenen der Energieversorgung – Strom, Gas und Fernwärme – darzustellen, wird noch ein Gasnetz, welches äquivalent zu den beiden bestehenden ringförmigen Netzen ist, aufgebaut. Hieraus ergibt sich eine Gesamtzahl von 60 Bussen, 120 Leitungselementen und 60 Kopplungsstellen der Energiesektoren. Da jeder Standort einen Verbraucher aufweist, ergibt sich eine Gesamtzahl von 60 Verbrauchern. Bei Hinterlegung von stündlichen Last- und Leistungsprofilen für ein Jahr ergibt sich bei der Lösung des optimalen Energieflusses basierend auf den geringsten Kosten des Energiesystems des CBC-Solvers eine Berechnungszeit von 01:25 Stunden.

7.3.5. Fazit

Die Anzahl an Verbrauchern ist entscheidend für die Berechnungsdauer des Energiesystems. Umso mehr Verbraucher in einem System integriert sind, umso mehr Möglichkeiten entstehen für den Solver diese zu lösen. Die Dauer der Suche nach dem optimalen Fluss der Energie wird somit größer. Geht man davon aus, dass man mit oemof ein realistisches Energiesystem der Größe eines Ausschnittes der Mittelspannungsebene mit etwa 20 Standpunkten und den dazugehörigen Lasten, Einspeisern und Speichern wählt, dieses auf der Strom-, Gas sowie Fernwärmeebene und miteinander verknüpft, dann ergibt sich eine Berechnungszeit von über einer Stunde. Es bietet sich an, bei der Simulation von großen Energiemodellen mehrere Verbraucher zu einem zusammenzufassen, sofern diese an einem Standpunkt des Energiesystems angeschlossen sind, der nicht für die Simulation relevant ist.

Des Weiteren fiel in diesem Versuch auf, dass das Hinzufügen von Senken, an denen überflüssige Energie abfließen kann, bei dem Aufbau von Energiesystemen immer berücksichtigt werden sollte. Wenn zu viel Energie in einem Netz vorhanden ist und diese nicht vollständig über die vorhandenen Leitungen oder Umwandlungsstellen hin zu Verbraucherlasten oder einer Senke geführt werden kann, entstehen unlösbare Probleme innerhalb des Programmablaufs.

Unlösbare Probleme werden in der Ausgabekonsole in Python angezeigt. Die Ursache der Fehler muss der Benutzer jedoch selbst finden. Ebenso kann das Gegenteil, ein Mangel an Energieeinheiten, ein unlösbares Problem ergeben. Dies kann vor allem auftreten, wenn nur in einen Bus gleichbleibend Energie eingespeist wird und dieser mit weiteren Bussen verbunden ist, die fluktuierende Einspeiser haben, und eine begrenzte Leitungskapazität zwischen den Bussen den unbegrenzten Fluss der Energieeinheiten einschränkt.

Um solch ein Problem auszuschließen, empfiehlt es sich, an jeden Bus ein energieeinspeisendes Objekt mit exorbitant hohen variablen Kosten hinzuzufügen. Diese sichern das System auf einen möglichen Energiemangel hin ab, werden aber auch nur als letzte Möglichkeit der Energieversorgung aufgrund der hohen variablen Kosten gewählt. Diesen energieeinspeisenden Objekten kann wie gehabt der optimale Fluss der Energie berechnet und ausgegeben werden; sofern dieser einen Gesamtdurchfluss von 0 aufweist, kann der Benutzer sicher sein, dass das eigentlich aufgebaute Energiesystem keinen Energiemangel aufweist.

7.4. Versuchsszenario 3: Optimierung des Standortes einer Kopplungsstelle

7.4.1. Beschreibung

In diesem Versuchsszenario wird der optimale Standort innerhalb eines Energieversorgungssystems für eine festgelegte Kopplungsstelle gesucht. Dafür wird die Kopplungsstelle jeweils an die möglichen Standorte des Energiesystems integriert und der jeweilige optimale Durchfluss der Energie innerhalb des Systems berechnet. Durch Vergleichen des Gesamtdurchflusses an Energie durch die Kopplungsstelle an den möglichen Standorten kann ein Rückschluss auf den optimalen Standort der Kopplungsstelle gezogen werden. Optimierungsgrundlage sind die Gesamtkosten des Energiesystems.

Dies gilt beispielhaft für ein bestehendes Netz, in dem nicht klar ist, an welchen Standort eine fest definierte Kopplungsstelle integriert werden soll, um die höchste Effizienz zu erreichen. Dieser Versuch gibt Aufschluss darüber, inwiefern die Optimierung des Standortes einer Kopplungsstelle in oemof durchführbar ist.

7.4.2. Übersicht

Hierfür wird ein Energieversorgungssystem auf drei Ebenen (Strom-, Gas- und Wärmenetz) mit jeweils drei Standpunkten aufgebaut. Die Knotenpunkte sind ringförmig miteinander verbunden. Ein Bus stellt jeweils einen Knotenpunkt dar. Die Verbindungen zwischen den Knotenpunkten sind Leitungselemente mit einer Nennleistung von 500 Energieeinheiten und einem Umwandlungsfaktor von 0.95. Die elektrischen Busse 1 und 2 haben jeweils einen Einspeiser (Wind), der 3. Elektrische Bus hat zwei Einspeiser (Wind, PV). Es befindet sich ein Verbraucher am elektrischen Bus 2. Die Busse haben jeweils einen *sink*, um überschüssige Energie abgeben zu können. Der Aufbau des gesamten elektrischen Energieversorgungsnetzes ist in Abbildung 15 dargestellt:

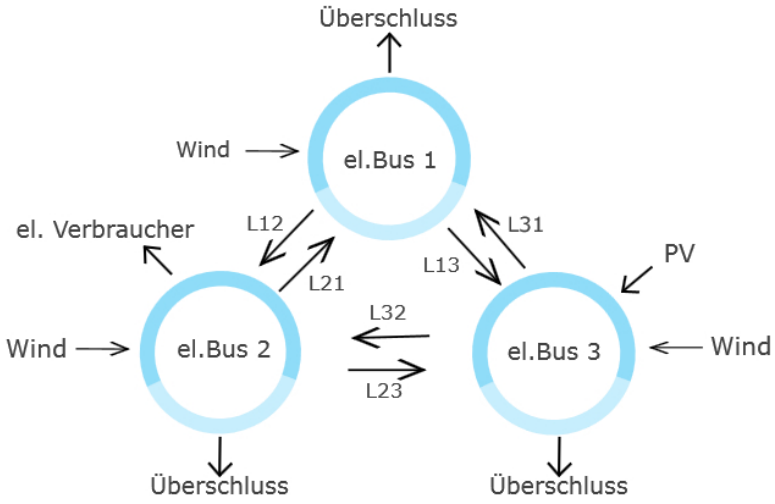

Abbildung 15: Visualisierung V3: Elektrisches Netz

Parallel dazu werden ein Gas- und ein Wärmenetz aufgebaut, welche ebenfalls drei Knotenpunkte aufweisen. Die drei durch Busse dargestellten Knotenpunkte werden mit Leitungselementen miteinander verbunden. Diese Leitungselemente sind in Abbildung 16 übersichtshalber nicht dargestellt, jedoch vorhanden. Siehe auch den Programmcode auf S. 86.

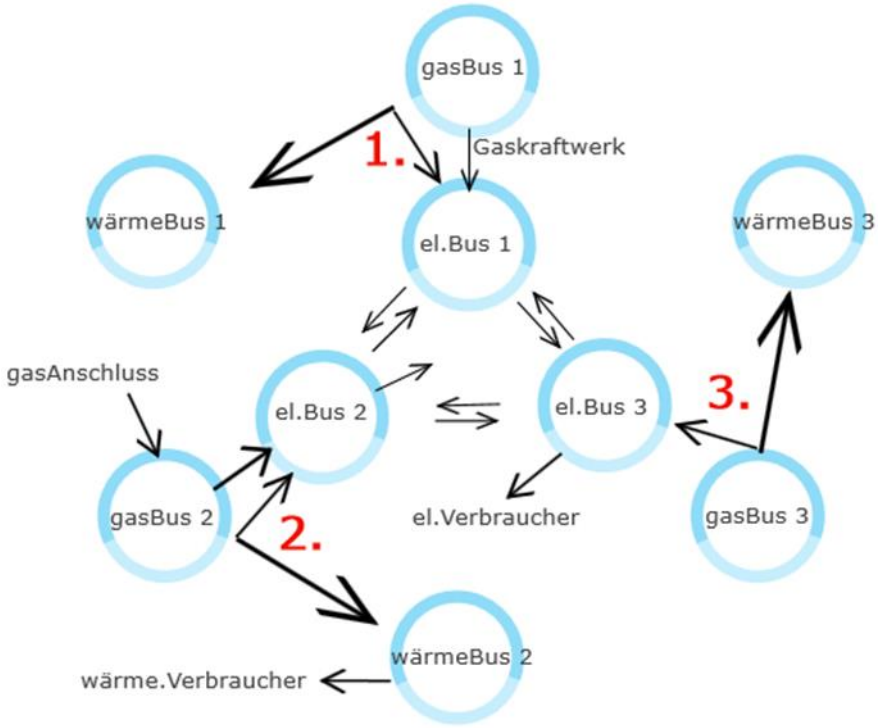

Abbildung 16: Visualisierung V3: Verknüpfung der Energiesysteme

Der Durchfluss der Energieeinheiten wird wie beim elektrischen Bus auf 500 Energieeinheiten begrenzt und der Umwandlungsfaktor von 0.95 simuliert einen Leitungsverlust von 5 %. Die Knotenpunkte 1 der drei Energienetze stellen einen

gemeinsamen Standort dar. Ebenso alle Knotenpunkte 2 sowie die Knotenpunkte 3. Damit ist ein gesamtes Energieversorgungssystem auf drei Ebenen aufgebaut, mit insgesamt drei Knotenpunkten, an denen nun Koppelstellen angebracht werden können.

Das Gasnetz wird am Gas-Bus 1 mit einem Netzanschluss verbunden, das einen unendlichen großen Gasvorrat simuliert. Es gibt am elektrischen Bus 2 einen Verbraucher und am Wärme-Bus 2 ebenfalls einen Verbraucher.

Neben dem Gaskraftwerk, das fest in das System integriert ist, ergeben sich für den Standort der Kopplungsstelle Kraft-Wärme-Kopplung drei Möglichkeiten. Diese sind mit 1., 2. und 3. rötlich im Schaubild markiert. Dabei soll herausgefunden werden, inwiefern unterschiedliche Standorte der Kopplungsstelle einen Unterschied für deren Durchfluss machen. Der vollständige Programmcode zu der visualisierten Übersicht befindet sich im Anhang auf S. 84–91.

7.4.3. Ablauf

Die KWK-Anlage wird also an jedem der drei Standorte nacheinander integriert und die drei verschiedenen Ergebnisse werden miteinander verglichen. Hier stellt sich die Frage, ob man eine Kopplungsstelle in die Nähe eines energieeinspeisenden Objekts setzt oder möglichst nahe an den gleichen Standort des Verbrauchs. Die Entscheidung hierfür ist natürlich von mehreren Faktoren abhängig: Die verwendeten Leitungen weisen Verluste auf, die elektrische Einspeisung der PV- und Windkraftanlage ist temporär unterschiedlich und die elektrischen wie auch Warmwasserverbraucher weisen ein sich änderndes Lastprofil auf. Je größer ein Aufbau des gesamten Energieversorgungsnetzes ist, desto mehr Möglichkeiten ergeben sich. Hier lassen sich aber vorab bei Betrachtung der Eigenschaften der Objekte schnell einige Standorte, an denen der Energiefluss wohl optimal ist, ausfindig machen und diese könnten dann durch oemof genau berechnet werden.

Fließt mehr Energie durch eine Kopplungsstelle, die einen höheren Wirkungsgrad besitzt als die fest installierten Kopplungsstellen des Energiesystems, dann ist mit niedrigeren Gesamtkosten der Deckung der Verbrauchslasten zu rechnen, da effizienter mit den Energieeinheiten umgegangen wird. Dies kann sich natürlich ändern, wenn eine effizientere Anlage mit einem kostspieligeren Einspeiser versorgt wird und nicht wie in diesem Beispiel eine einzige zugrunde liegende Energiequelle im Gas-Bus aufweist.

Die Kosten pro Energieeinheit der Einspeiser und Umwandlungsstellen sind auf folgende Werte festgelegt: Die regenerativen Energien haben variable Kosten von 0, die des Gaskraftwerks besitzen einen fiktiv hohen Wert von 100 und die KWK-Anlage von 50. Diese Werte sollen nur eine realistische Anordnung der Kosten geben, spiegeln aber keine realen

Werte wider. Dies ist ausreichend, um die Möglichkeit einen passenden Standort für eine Kopplungsstelle innerhalb einer in oemof aufgebauten Umgebung aufzuzeigen.

7.4.4. Auswertung

Um den optimalen Standort der KWK-Anlage zu bestimmen, werden die Ergebnisse des Gesamt-Durchflusses der Energieeinheiten der Kopplungsstellen an den drei unterschiedlichen Standorten miteinander verglichen:

Tabelle 9: Gesamtdurchfluss an Energieeinheiten an den jeweiligen KWK-Standorten

Standort:	1.	2.	3.
Gaskraftwerk Summe	4582	4582	5322
Gaskraftwerk maximal	248	248	248
KWK Summe	7167	7167	6557
KWK maximal	100	100	100

Am 2. Standort ergibt sich für die KWK-Anlage der höchste Gesamtdurchfluss in Höhe von 7.167 Energieeinheiten. An diesem gewählten Standort würde die KWK-Anlage innerhalb des Energieversorgungssystems den größten Durchfluss besitzen und man würde die geringste Leistung aus dem Gaskraftwerk beziehen. Dies wäre somit der optimale Standort bezogen auf die Gesamtkosten des Energiesystems. Die KWK-Anlage geht an allen drei Standorten temporär an das Maximum ihrer Nennleistung heran. Die Standorte 1 und 3 weisen gleiche Werte auf, da die Leitungselemente vom Gasvorrat zur Anlage und Richtung Verbraucher in beiden Fällen die gleichen Eigenschaften besitzen.

7.4.5. Fazit

Man kann mit oemof innerhalb einer aufgebauten Umgebung eine Kopplungsstelle nacheinander an verschiedenen Standorten integrieren, die Simulationen miteinander vergleichen und die Gesamtkosten des Energiesystems pro Simulation berechnen, indem man die Kosten pro Energieeinheit mit dem Gesamtdurchfluss der unterschiedlichen Objekte miteinander multipliziert. Der Standort, an dem die Kopplungsstelle den größten Gesamtdurchfluss aufweist, also am höchsten ausgelastet ist, ist somit der optimale Standort für die Kopplungsstelle, sofern diese Kopplungsstelle geringere Kosten aufweist als die übrigen Anlagen innerhalb des Energiesystems.

Bei einem realitätsgetreuen Aufbau mit einer großen Anzahl an möglichen Standorten setzt die Methodik voraus, möglichst optimale Standorte vorab ausfindig zu machen und nur diese miteinander zu vergleichen, da sonst zu viele Standorte überprüft werden müssen.

Nachteil der Methodik ist jedoch, dass die in Frage kommenden Standpunkte der Kopplungsstelle einzeln simuliert und die Gesamtkosten der Energiesysteme miteinander verglichen werden müssen.

Hierbei kann man innerhalb der Simulation gut auf mögliche Änderungen der Lasten, neue Einspeiser oder neue Leitungen eingehen und diese testweise einbauen, um gegebenenfalls neue Möglichkeiten eines effizienteren Energiesystems zu finden.

7.5. Versuchsszenario 4: Vergleich Simulation Elektro-Wärmepumpe

7.5.1. Beschreibung

Es wird explizit die Simulation einer Elektro-Wärmepumpe betrachtet. Diese wird als Wärmequelle eines Haushaltes eingesetzt. Der optimale Durchfluss wird simuliert und das Ergebnis wird mit einer Simulation nach dem Berechnungstool 170327_Lastprofil-und-Wärmepumpe, das an der Technischen Hochschule Köln durchgeführt worden ist, verglichen. Dadurch sollen Rückschlüsse auf die Simulation durch oemof im Vergleich zu dem Excel-Tool gezogen werden. Dieses Excel-Tool bietet die Möglichkeit, über eine Eingabemaske die Nennwerte eines Wärmeverbrauchs, einer Wärmepumpe sowie eines zusätzlichen Speichers zu variieren, um die Eigenbedarfsabdeckung und den Gesamtautarkiegrad eines Haushaltes mit den verwendeten Komponenten zu ermitteln. Dabei können umgebungsabhängige Variablen wie Temperatur, Ausrichtung der Photovoltaikanlage und der tatsächliche Heizbedarf, abhängig von der Gebäudegröße, verändert werden.

7.5.2. Übersicht

Für das Versuchsszenario werden ein Strom- sowie ein Wärmestandort festgelegt, die durch jeweils einen Bus dargestellt werden. Der Stromstandpunkt wird durch eine Photovoltaikanlage und einen Netzanschluss, der die höheren Kosten pro Energieeinheit aufweist als die Photovoltaikanlage, versorgt. Für den Haushaltsstromverbrauch wird ein Lastprofil nach BDEW-Norm hinterlegt [44]. Die Wärmepumpe kann je nach Bedarf des Wärmeverbrauchs Strom in Wärme umwandeln. Überflüssige Energie in Form von Wärme kann in Warmwasserspeicher gespeichert werden. Als Lastprofil des Wärmeverbrauchers wird ebenfalls ein Standardlastprofil nach BDEW ausgewählt. Das Lastprofil setzt sich aus stündlichen Werten eines Jahres zusammen. Jedem Tag ist ein Tagesverbrauch in kWh hinterlegt. Dieser Tagesverbrauch wird abhängig von der Temperatur prozentual auf die Stunden des Tages nach BDEW Norm aufgeteilt in kWh/h bzw. kW. Dem Haushaltsverbrauch an Strom wird ebenfalls ein Standardlastprofil nach BDEW hinterlegt.

Der Wärmepumpe wird zu jedem Zeitwert eine Leistungszahl durch Einbindung in eine Tabelle hinterlegt. Diese Leistungszahl ist abhängig von der Temperatur der Umgebung, die variiert, und von der Zieltemperatur der Wärmepumpe, die auf 60 Grad festgelegt ist. Zusätzlich besteht die Möglichkeit, Energie in Form von Wärme in einem Warmwasserspeicher mit einer Speichergröße von 199 Litern zu speichern, was 11.47 kWh entspricht. Dieser Speicher weist Verluste von 0.06 kW/h auf [44]. Umgerechnet entspricht das einem Verlust von 0.03 %. Die Photovoltaikanlage hat eine Leistung von 7 kWp, für

diese wird ebenfalls ein Standardlastprofil nach BDEW eingebunden. Diese Werte sind beispielhaft aus den Literaturangaben des Excel-Tools für einen möglichen Aufbau des Energiesystems entnommen worden. Daraus ergibt sich folgende Visualisierung:

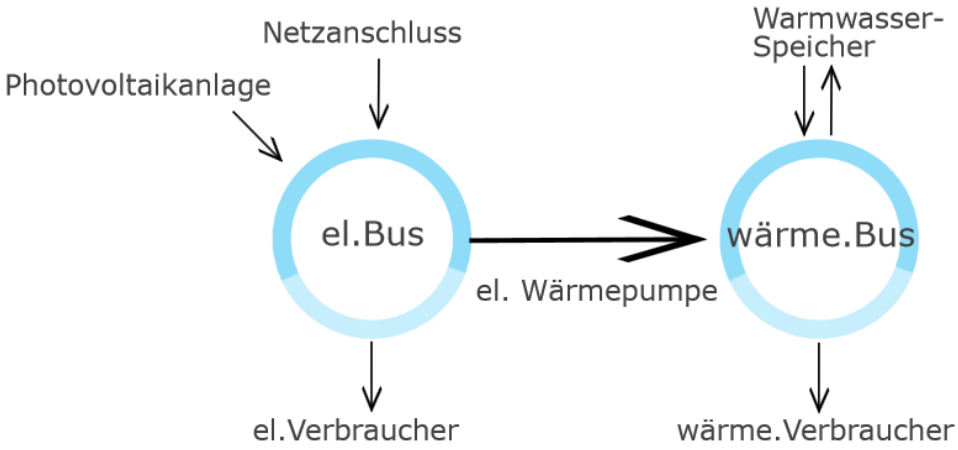

Abbildung 17: Visualisierung V4: Umfeld der Wärmepumpe

Der vollständige Programmcode zu der visualisierten Übersicht befindet sich im Anhang auf S. 92–95.

7.5.3. Ablauf

Das aufgebaute Energiesystem wird für den Zeitraum von einem Jahr simuliert. Hierbei werden nach dem optimalen Durchfluss der Wärmepumpe unter Sättigung der Verbraucher die günstigsten Kosten des Gesamtenergiesystems berechnet.

7.5.4. Auswertung

Es ergibt sich eine über das gesamte Jahr berechnete Energiebereitstellung der Wärmepumpe in Höhe von 3.996 kWh, der Wärmebedarf in diesem Zeitraum beträgt allerdings 3.891 kWh. Dies hängt damit zusammen, dass bei der Berechnung des Optimierungsproblems in oemof zuerst alle Daten eingelesen werden und danach optimiert wird. Daraus ergeben sich mehr Zeitpunkte im Gegensatz zur Simulation des Excel-Tools, an denen die Wärmepumpe aufgrund einer höheren Temperatur und damit verbundenen höheren Wirkungsgrades die Nennleistung hochfährt, um den Warmwasserspeicher zu laden. Der Speicher verliert jedoch mit der Zeit Energie. Mit der Nutzung, von besser gewählten Zeitpunkten der Einschaltung der Wärmepumpe, ergibt sich trotz der auftretenden Verluste des Warmwasserspeichers auf die Kosten bezogen ein effizienterer Ablauf.

Zur näheren Betrachtung sind die Leistungen der Objekte, die an dem Wärme-Bus angeschlossen sind, für den Zeitraum der ersten Woche graphisch in Abbildung 18 dargestellt:

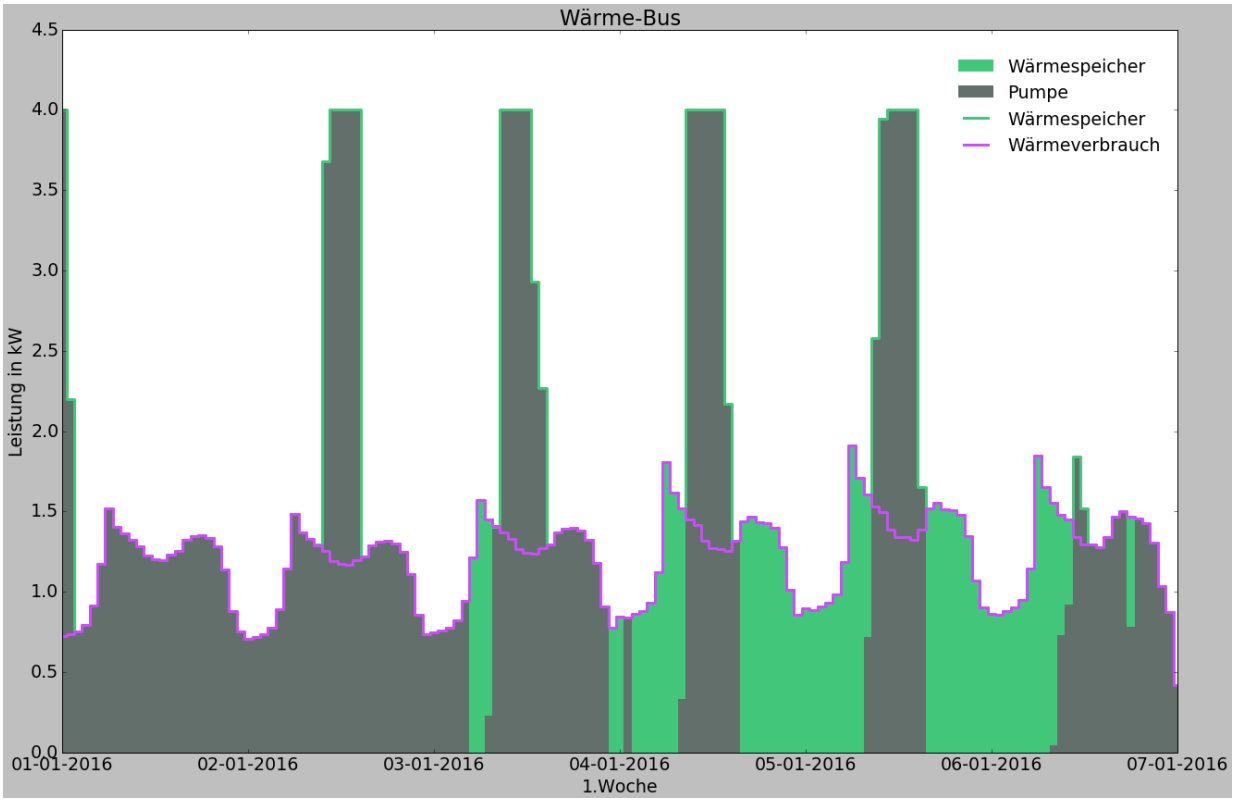

Abbildung 18: Graphische Auswertung: Leistungen am Wärme-Bus der 1.Woche

Dieser Zeitraum wurde als graphische Auswertung gewählt, da man die Änderungen der Leistungen der Objekte gut erkennen kann. Hier sieht man, dass die elektrische Wärmepumpe immer so viel Leistung bereitstellt, dass der Wärmeverbrauch gedeckt ist. Die Peaks der Wärmepumpe, an denen die volle Leistung von 4 kW abgerufen wird, sind die Zeitpunkte, an denen optimal die Wärmepumpe genutzt wird, um Energie im Warmwasserspeicher zu speichern. Dies geschieht so lange, bis der Warmwasserspeicher des Wärmebusses seine maximale Kapazität erreicht hat.

Ebenso wird die Wärmepumpe immer dann bedient, wenn die PV-Anlage Energie in den Strom-Bus einspeist. Da die Leistungszahl der Wärmepumpe immer über eins liegt, stellt die PV-Anlage immer zuerst die Energie für die Wärmepumpe bereit, da dies für das Gesamtsystem effizienter ist, als den Haushaltsstrombedarf zu decken. Dies ist gut in Abbildung 19 an zwei Tagen im Juni zu erkennen:

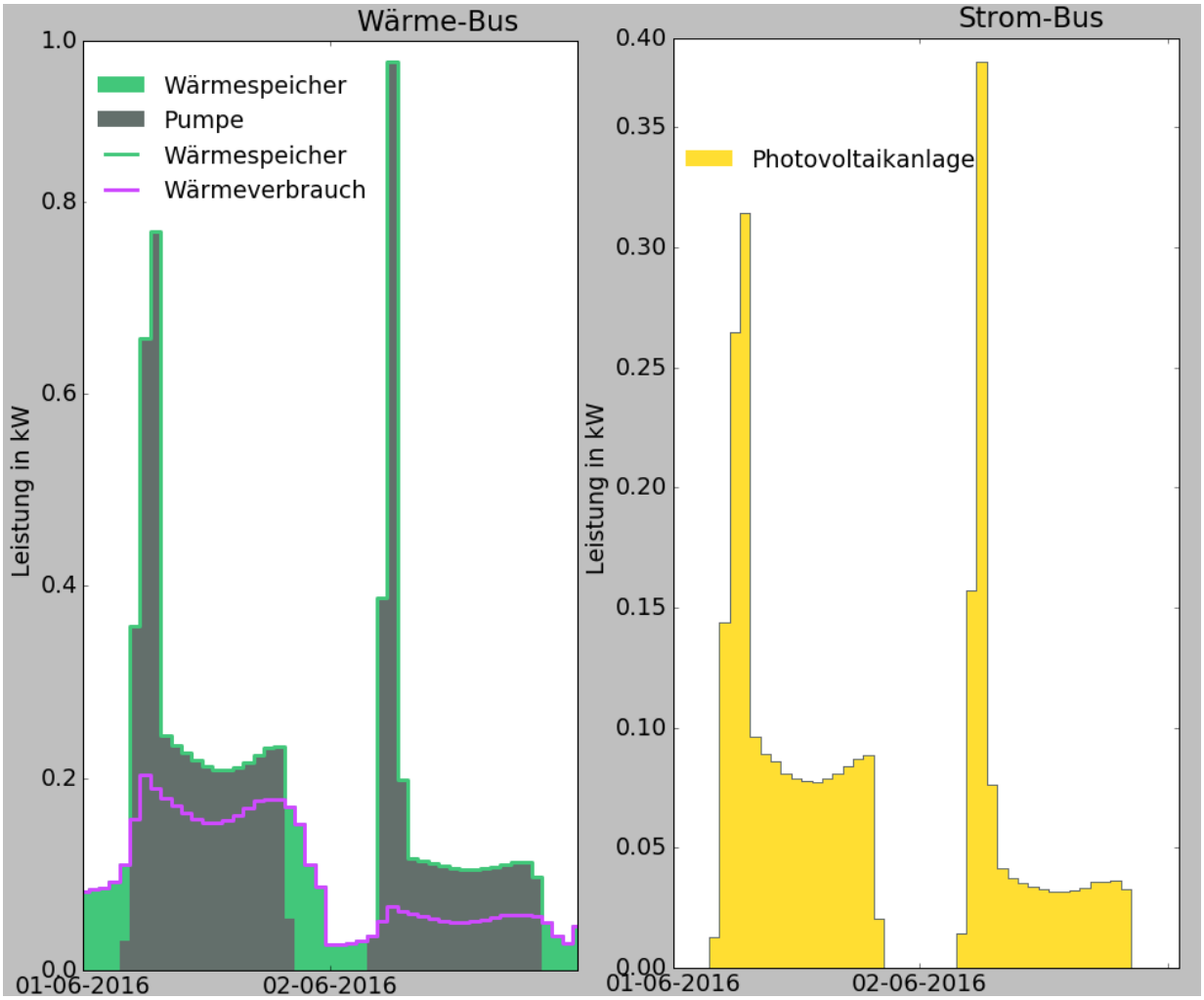

Abbildung 19: Vergleich: PV-Anlage zu Wärmepumpe an zwei Sommertagen

Die grünen Bereiche sind die Zeiträume, in denen der Wärmeverbrauch nun durch den Wärmespeicher gedeckt wird. Am nächstgünstigsten Zeitpunkt liefert die Wärmepumpe dann wieder mit voller Leistung Energie. Hieran erkennt man gut, dass oemof für die Berechnung zuerst alle Leistungs- und Lastprofile einliest und danach optimiert.

7.5.5. Fazit

Bei der Simulation in oemof ergibt sich für die Bereitstellung an Energieeinheiten der Wärmepumpe ein höherer Wert als in der Berechnung des Excel-Tools. Dies hängt damit zusammen, dass im Excel-Tool schrittweise vorgegangen wird und nicht wie bei oemof alle Daten direkt eingelesen werden und danach optimiert wird. Die einfache Darstellung einzelner Zeiträume und Ausgabe der gewünschten Ergebnisse zeigt, dass oemof ein praktisches Simulationsumfeld für Energiesysteme bietet. Die Einbindung oemofs zur Simulation verschiedener energietechnischer Probleme könnte in Projekten an der TH Köln einen Mehrwert liefern.

8. Optimierungsansätze

8.1. Einleitung

Bei der Simulation eines Energiesystemmodells muss ein Ansatz zur Optimierung gewählt werden. Diese Ansätze können verschiedene Aspekte beinhalten. Ein klassischer Ansatz besteht darin, die Kosten des gesamten Energiesystems möglichst gering zu halten. Diese Reduzierung, also ein möglichst geringer CO_2-Ausstoß, wäre ein ökologischer Optimierungsansatz. Da Energieversorgungssysteme eine hohe Priorität auf die Versorgungssicherheit legen, ist eine Optimierung auf Netzstabilität ebenfalls ein denkbarer Optimierungsansatz.

Bei der Optimierung von Umwandlungsstellen ist einerseits der optimale Standort dieser Anlagen interessant, um eine Umwandlungsstelle, deren Parameter vorgegeben sind, optimal auslasten zu können; andererseits könnte man für einen festgelegten Standort die optimale Umwandlungsstelle finden. Hierfür könnten verschiedene Parameter wie die Nennleistung oder die Art der Anlage auf die Kosten hin optimiert werden.

Oemof ist generell dafür ausgelegt, den Fluss der Energieeinheiten auf die geringsten Kosten hin zu lösen unter Berücksichtigung der Last- und Leistungsdaten der Verbraucher und Objekte. Aufgrund des einheitenlosen Aufbaus bietet oemof aber einfach weitere Möglichkeiten in der Simulation an, als nur den optimalen Energiefluss eines Energiesystems zu bestimmen. Das Kapitel gibt einen Überblick, welche der Optimierungsansätze sich in oemof umsetzen lassen, welche nicht umsetzbar sind und welche Erweiterungen oemof benötigen würde, um die fehlenden Optimierungsansätze auch in Anspruch nehmen zu können.

8.2. Optimierungsansatz: Gesamtkosten

Der ökonomische Ansatz, die Gesamtkosten eines Energiemodells zu optimieren, ist die klassische Idee bei der Simulation von Energiesystemen.

Um die Kosten des gesamten simulierten Energiesystems zu optimieren, muss jedem Objekt innerhalb des Energiesystems ein Wert für die Kosten pro Energieeinheit zugeordnet werden können. In oemof steht hierfür der Parameter *costs* bereit. Energieeinspeisenden Objekten können neben variablen Kosten, die sich auf die Kosten pro Energieeinheit beziehen, auch Fixkosten zugeordnet werden, genauso wie einer Umwandlungsstelle. Dann kann der Fluss der Energie unter Berücksichtigung der definierten Parameter der Objekte mit einem Solver gelöst werden. Oemof ist für diesen klassischen Optimierungsansatz ausgelegt.

8.3. Optimierungsansatz: CO_2-Reduzierung

Innerhalb der Energiewende wandelt sich das Bestreben der Energieversorgung von der reinen Sättigung der Lasten hin zu effizienter Nutzung der Energie und vor allem zur Nutzung von nachhaltigen Energieressourcen mit geringer Schadstoffbelastung. Ein typischer Ansatz hierfür ist ein optimaler Energiefluss eines Energiesystems mit möglichst geringem CO_2-Ausstoß. Hierbei wird in der Simulation den Objekten jeweils ein Wert für den CO_2-Ausstoß zu geordnet.

Da oemof einheitenlos aufgebaut ist, lässt sich der Parameter *costs* einfach als CO_2-Ausstoß definieren. Wenn man nun nach dem Energiefluss hin das Energiesystem löst, ergibt sich der optimale Energiefluss für den geringsten CO_2-Ausstoß. Der Optimierungsansatz ist in oemof umsetzbar. Es beinhaltet allerdings, dass die Kosten des Energiesystems somit nicht mehr betrachtet werden können. Hier wäre eine Erweiterung auf mehrere *costs*-Parameter innerhalb von oemof nötig, um direkt die Kosten und die Höhe des CO_2-Ausstoßes innerhalb einer Simulation definieren zu können. Abschließend müsste unterschieden werden, welchen der Parameter man definieren will. Dies würde die ökologische und ökonomische Optimierung in einem einzigen Simulationsaufbau möglich machen.

8.4. Optimierungsansatz: Netzstabilität

Ein Energieversorgungssystem hat neben ökonomischen und ökologischen Aspekten immer das Ziel, eine Sicherheit der Versorgung zu gewährleisten. Für die Sicherheit muss ein Parameter definiert werden, denn Versorgungssicherheit bedeutet neben der reinen Sättigung aller Verbrauchslasten auch, auf Eigenschaften des Netzes zu achten, die dafür sorgen könnten, dass ein elektrisches Netz zusammenbricht. Die Netzfrequenz ist hier entscheidend, da diese bei zu starker Abweichung vom Normalwert (in Deutschland herrscht eine Netzfrequenz von 50 Hertz) für einen totalen Versorgungsausfall führen kann [45]. Neben der Netzfrequenz müssen Leitungen Spannungskriterien einhalten. Eine Verletzung der Kriterien könnte zum Ausfall von Leitungen und somit zu einem Versorgungsabbruch innerhalb des Energiesystems führen.

Die Darstellung von Leitungen ist in oemof nicht separat vorgesehen, deshalb fallen sie unter die Kategorie Umwandlungsstellen und können auch nur die hierfür vorgesehenen Eigenschaften annehmen. Durch die Nicht-Berücksichtigung von elektrischen Eigenschaften der Leitungen, wie in Kapitel 4.5 beschrieben, können Leitungen nur vereinfacht dargestellt werden. So kann zwar mit der Begrenzung der Kapazität einer Leitung ihr Durchfluss auf einen festgelegten unkritischen Wert beschränkt werden, hier ist jedoch keine anschließende

Optimierung dieses Parameters möglich. Für eine wissenschaftliche Betrachtung des Problems in oemof fehlt eine Möglichkeit der vollständigen Darstellung von elektrischen Leitungen. Eine Verknüpfung mit einem weiteren Programm, das sich auf elektrische Netze spezialisiert hat, wäre eine Möglichkeit. Die andere Möglichkeit wäre, eine Klasse von Leitungen mitsamt der Definition der fehlenden Parameter in oemof einzubetten. Dies erscheint aber unwahrscheinlich, da oemof sich so der Einfachheit durch die einheitenlose Darstellungsweise entledigen würde.

8.5. Optimierungsansatz: Standort der Kopplungsstelle

Wird in ein vorhandenes Energieversorgungssystem eine neue Koppelstelle integriert, so ergeben sich verschiedene Möglichkeiten für den Standort dieser Koppelstelle. Hier wird der optimale Standort gesucht, an dem die Koppelstelle im vorhandenen System den höchsten Gesamtdurchfluss der Simulation aufweist. Dies ist in oemof, wie in Kapitel 7.4 erläutert, möglich, sofern man die möglichen Standorte der Kopplungsstelle einzeln simuliert und die Gesamtdurchflüsse der Kopplungsstelle miteinander vergleicht. Hier wäre es wünschenswert, den Prozess automatisch durchführen zu können, um bei vielen möglichen Standorten der Kopplungsstelle nicht die gleiche hohe Anzahl an Berechnungen durchführen zu müssen. So eine Option ist in oemof nicht gegeben.

8.6. Optimierungsansatz: Variation der Eigenschaften einer Kopplungsstelle

In diesem Fall geht man davon aus, dass eine Kopplungsstelle innerhalb eines Energieversorgungssystems einen festgelegten Standort besitzt, aber eine Auswahl an verschiedenen Ausführungen der Kopplungsstelle auswählbar ist. Das könnte zum Beispiel die Nennleistung der Kopplungsstelle sein. Eventuell würden sich auch weitere Parameter der Kopplungsstelle dadurch ändern. Hier bietet oemof ähnlich wie bei der Optimierung des Standortes einer Kopplungsstelle nur die Möglichkeit, die in Frage kommenden Ausführungen der Koppelstelle nacheinander ins System zu integrieren und jeweils die Gesamtkosten für die unterschiedlichen Fälle zu ermitteln.

9. Zusammenfassung

Mit der Programmbibliothek oemof lassen sich auf einfache Weise Energiesysteme mit energieeinspeisenden Objekten, Verbrauchern und Koppelstellen darstellen sowie deren Energieflüsse berechnen beziehungsweise optimieren. Die einheitenlose Darstellung der Objekte eines Energiesystems gibt dem Benutzer eine Vielzahl an Möglichkeiten der Darstellung und erlaubt einen einfachen Aufbau. Bei dem Aufbau von energieeinspeisenden Objekten und von Energiespeichern lassen sich aufgrund der vielfältigen Charakterisierungsmöglichkeiten alle Eigenschaften darstellen. Verbraucher beziehungsweise Lasten lassen sich in gewünschten Zeitschritten problemlos einbinden. Auch die Darstellung der Umwandlungsstellen, die ein zentrales Thema dieser Arbeit sind, bereitet keine Probleme.

Der Nachteil der einheitenlosen Darstellungsweise ist, dass wichtige Eigenschaften der energieführenden Objekte, die Auswirkungen auf den gesamten Energiefluss innerhalb des Energiesystems haben, nicht dargestellt werden können. Dieser Nachteil zeigt sich vor allem in der Darstellung von elektrischen Leitungen innerhalb von oemof.

Hier ist vor allem die Darstellung von elektrischen Leitungen ein Problem, da wichtige Parameter wie Strom und Spannung, die im Rahmen einer Lastflussberechnung ermittelt werden müssten, nicht eingelesen werden können. Hier müssen vorab die Eigenschaften vereinfacht werden, um sie darstellen zu können, und zwar in Form einer Begrenzung des zulässigen Durchflusses einer Leitung.

Dies hat eine nicht ausreichend wissenschaftliche Betrachtung der Leitungen zur Folge. Da die Kopplung von elektrischen Netzen, Fernwärmenetzen sowie von Gasnetzen im Forschungsprojekt ES-Flex-Infra betrachtet wird, darf der Fokus nicht allein auf die Darstellung der elektrischen Komponenten fallen. Die Verbindungen zwischen zwei Punkten des Fernwärmenetzes wie auch eines dargestellten Gasnetzes erfordern ebenso eine komplexere Darstellung, als sie aktuell in oemof möglich sind.

Die Versuchsszenarien zeigen, dass mit oemof Energiesysteme aufgebaut werden können und diese, sofern sie nicht zu viele Objekte beinhalten, innerhalb von einfachen Berechnungszeiten simuliert und nach dem optimalen Durchfluss hin berechnet werden können. Die formulierten Optimierungsansätze lassen sich abgesehen von der Netzstabilität mit oemof durchführen. Hierfür fehlen Eingabeparameter von elektrischen Eigenschaften der Objekte und vor allem der Leitungselemente.

Positiv herauszustellen ist der simple Aufbau von Energiesystemen in oemof. Die einfache Programmierung und problemlose Ausgabe von Ergebnissen setzt keine lange Einarbeitung für neue Benutzer voraus.

10. Ausblick

Trotz der Vereinfachungen in der Darstellung bietet oemof dem Benutzer durch die verschiedenen variablen Eigenschaften der Objekte eines Energiesystems eine gute Basis, ein Energiesystem mitsamt den energieeinspeisenden Objekte, Verbraucher und Kopplungsstellen, die innerhalb des Energiesystems auftreten, darzustellen und zu charakterisieren.

Die aufgebauten Energiesysteme bieten die Grundlage für die Simulation und Optimierung von Energiekopplungsstellen. Deshalb bietet sich oemof als Programm für die Darstellung und Berechnung innerhalb der Technischen Hochschule für das Forschungsprojekt ES-Flex-Infra sowie für weitere Projekten an. Die vereinfachte Darstellung von Leitungen muss hierbei allerdings in Kauf genommen werden.

Die Simulation der elektrischen Wärmepumpe aus den Vorgaben des Excel-Tools in Kapitel 7.5 hat gezeigt, dass oemof nicht nur für Optimierungen von Kopplungsstellen innerhalb des Forschungsprojekts ES-Flex-Infra für die TH Köln interessant ist, sondern für jede Art der Simulation von Energiesystemen, da ein energietechnischer Sachverhalt einfach und präzise mit oemof simuliert werden kann.

Es bietet sich an, die zukünftige Entwicklung zur Integration eines Programms, das die Eigenschaften der elektrischen Netze miteinbezieht, in oemof zu verfolgen und daran mitzuwirken. Hier bietet PyPSA, das sich auf die Darstellung von elektrischen Netzen spezialisiert hat und wie oemof auf eine open-source-basierte Programmierung setzt, eine gute Möglichkeit. Durch die Integration ließen sich die Vorteile der beiden Programme vereinen.

11. Literaturverzeichnis

[1] M. F. Peter Hennicke, *Erneuerbare Energien: mit Energieeffizienz zur Energiewende*, 2nd ed.: C.H.Beck, 2007.

[2] Matthias Popp, Ed., *Speicherbedarf bei einer Stromversorgung mit erneuerbaren Energien*, 2010.

[3] Europa-Universität Flensburg, *oemof Kurzbeschreibung.* [Online] Available: https://www.uni-flensburg.de/eum/forschung/laufende-projekte/oemof/.

[4] Reiner Lemoine Institut gGmbH, *Das Reiner Lemoine Institut.* [Online] Available: http://reiner-lemoine-institut.de/ueber-uns/institut/. Accessed on: Feb. 01 2017.

[5] *oemof organisation · GitHub: Community.* [Online] Available: https://github.com/oemof. Accessed on: Feb. 05 2017.

[6] Holger Trapp, *Die Programmiersprache Python: 2. Programmierparadigmen.* [Online] Available: https://www-user.tu-chemnitz.de/~hot/PYTHON/#paradig.

[7] *Python Software Foundation.* [Online] Available: https://www.python.org/psf/.

[8] TH Köln, *Forschungsprojekt ES-Flex-Infra.* [Online] Available: https://www.th-koeln.de/informations-medien-und-elektrotechnik/forschungsprojekt-es-flex-infra_37260.php. Accessed on: Mar. 01 2017.

[9] Andrea Vedovo, *Energy scientists must show their workings.* [Online] Available: http://www.nature.com/news/energy-scientists-must-show-their-workings-1.21517. Accessed on: Mar. 17 2017.

[10] oemof-Team, *oemof-solph: Add your components to the energy system.* [Online] Available: http://oemof.readthedocs.io/en/v0.1.1/oemof_solph.html#add-your-components-to-the-energy-system. Accessed on: Jan. 06 2017.

[11] Fares Farah, *Wirtschaftlichkeitsszenarien von Speichermöglichkeiten als Grundlage für Geschäftsmodelle von Energieversorgern*, 1st ed.: Diplomica Verlag, 2014.

[12] Dr. Harry Wirth, Fraunhofer ISE, "Aktuelle Fakten zur Photovoltaik in Deutschland,"

[13] Bundesverband WindEnergie e.V., *Kleinwind | Bundesverband WindEnergie e.V.* [Online] Available: https://www.wind-energie.de/themen/kleinwind. Accessed on: Feb. 04 2017.

[14] Dr. Markus Reichel, *Markteinführung von erneuerbaren Energien: Lock-Out-Effekte und innovationspolitische Konsequenzen für die elektrische Wind- und Solarenergienutzung*: Springer, 1998.

[15] U. Eicker, *Solare Technologien für Gebäude*. S.82, 2nd ed.: Vieweg+Teubner Verlag.

[16] E. Waffenschmidt, "Vorlesung Elektrische Netze: Netzwerke berechnen,"

[17] E. Waffenschmidt, “Vorlesung: Elektrische Netze: Leitungen,”

[18] Diss. A. Gassel, “Fernwärme: Energieverteilung mittels Fernwärme,” S.51.

[19] Dietrich Stein, *Der begehbare Leitungsgang*, 2002.

[20] Knut Håkansson, *Lexikon der Gasinstallation: (Gasinstallation von A - Z)*, 2nd ed.: Vulkan-Verlag, 1996.

[21] BEM-Fachgruppe, “Strategischer Energieeinkauf,”

[22] M. Sterner and I. Stadler, “Definition und Klassifizierung von Energiespeichern,” in *Energiespeicher - Bedarf, Technologien, Integration*, M. Sterner and I. Stadler, Eds., 2nd ed., Berlin, s.l.: Springer Berlin, 2016, pp. 25–46.

[23] S. Eckstein, “Lithium-Ionen-Batterie,” S.43.

[24] Statista, *Preisentwicklung von Lithium-Ionen-Batterien | Prognose.* [Online] Available: https://de.statista.com/statistik/daten/studie/534429/umfrage/weltweite-preise-fuer-lithium-ionen-akkus/. Accessed on: Feb. 08 2017.

[25] Christian Märtel, *Biogasspeicher.* [Online] Available: https://www.heizungsfinder.de/bhkw/biogasanlage/biogasspeicher.

[26] Arbeitsgemeinschaft Neue Energie Deutschland, *Heizung heute: Gastherme, Gasheizung mit Brennwerttechnik.* [Online] Available: http://www.deutsches-energieportal.de/heizung-heute/heizung-gasheizung.html. Accessed on: Feb. 28 2017.

[27] Sven Geitmann, *Energiewende 3.0: 3.5 Stadtgas*. S.70, 3rd ed.: Hydrogeit Verlag, 2012.

[28] S. Eckstein, “Virtuelles Institut: Strom zu Gas und Waerme: Technologiecharakterisierungen: Allgemein,”

[29] Max Blatter, *Atlas der erneuerbaren Energien*, 4th ed.: Books on Demand, 2015.

[30] MVG Mainzer Verkehrsgesellschaft, *Wasserstoff-Vorzeigeprojekt läuft erfolgreich.* [Online] Available: http://www.energiepark-mainz.de/artikel-detailseite/article/wasserstoff-vorzeigeprojekt-laeuft-erfolgreich/. Accessed on: Jan. 20 2017.

[31] Udo Leuschner, *Energie Wissen - Wärmepumpe.* [Online] Available: http://www.udo-leuschner.de/basiswissen/SB127-05.htm.

[32] R. Paschotta, *Elektrowärmepumpe.* [Online] Available: https://www.energie-lexikon.info/elektrowaermepumpe.html. Accessed on: Jan. 15 2017.

[33] Robert Doelling, *Aktuelle Gesetze zum Verbot der Nachtspeicherheizung.* [Online] Available: http://www.energie-experten.org/heizung/elektroheizung/nachtspeicherheizung/verbot.html.

[34] Matthias Günther, *Energieeffizienz durch Erneuerbare Energien: Möglichkeiten, Potenziale, Systeme: Möglichkeiten, Potenziale, Systeme*: Springer, 2015.

[35] A. Rötzel, *Praxiswissen umweltfreundliches Bauen: Kraft-Wärme-Kopplung S.59*, 1st ed. Wiesbaden: Vieweg+Teubner Verlag, 2005.

[36] Institut für Energiewirtschaft und Rationelle Energieanwendung Universität Stuttgart, "Kraft-Wärme-Kopplung," S.2, 2009.

[37] Bundesamt für Wirtschaft und Ausfuhrkontrolle, *Mini-KWK Förderung.* [Online] Available: http://www.bafa.de/DE/Energie/Energieeffizienz/Kraft_Waerme_Kopplung/Mini_KWK/mini_kwk_node.html;jsessionid=AB64E8E61E6F92FFF19CD1863A20F3D7.1_cid387. Accessed on: Feb. 21 2017.

[38] Universität Duisburg, "Adaptiver Energie- und Lastmanager: Für Mikro-KWK-Anlagen," S.8-S.9.

[39] *GuD-Kraftwerk | Energieumwandlung | KWK.* [Online] Available: http://energie-strom.com/energie/kraftwerke/gud-kraftwerk.html. Accessed on: Mar. 06 2017.

[40] John Forrest, *CBC User Guide.* [Online] Available: https://www.coin-or.org/Cbc/cbcuserguide.html. Accessed on: Mar. 10 2017.

[41] Peter Becker, *Operations Research II: Einführung in die kombinatorische Optimierung.* [Online] Available: http://www2.inf.fh-rhein-sieg.de/~pbecke2m/or2/.

[42] Uwe Krien.

[43] EnArgus - Fraunhofer FIT. [Online] Available: https://enargus.fit.fraunhofer.de/pub/bscw.cgi/d4841495-2/*/*/Ringnetz.html?op=Wiki.getwiki.

[44] BDEW Bundesverband der Energie- und Wasserwirtschaft e. V., "BDEW/VKU/GEODELeitfaden," vol. 2016.

[45] E. Waffenschmidt, "Vorlesung Regelleistung und Netzfrequenz: Frequenzgrenzen und 5-Stufen-Plan," S.38.

12. Anhang

12.1. Versuchsszenario 1:Programmcode „mit Leitung“

```
# Outputlib
from oemof import outputlib
# Default logger of oemof
from oemof.tools import logger
from oemof.tools import helpers
# import oemof base classes to create energy system objects
import logging
import os
import pandas as pd
import matplotlib.pyplot as plt
import oemof.solph as solph


def optimise_storage_size(filename="EinTagTEST2.csv", solvername='cbc', debug=True, number_timesteps=48, tee_switch=True):
    logging.info('Initialize the energy system')
    date_time_index = pd.date_range('1/1/2016', periods=number_timesteps, freq='H')


    energysystem = solph.EnergySystem(timeindex=date_time_index)
# Read data file
    full_filename = os.path.join(os.path.dirname(__file__), filename)
    data = pd.read_csv(full_filename, sep=",")
# Create oemof object
    logging.info('Create oemof objects')
# create Gas_bus
    bgas = solph.Bus(label="Gasnetz")
# create Elektrizitäts_Bus
    bel1 = solph.Bus(label="Gesamt_Elektrizität_1")
# create Elektrizitäts_Bus
    bel2 = solph.Bus(label="Gesamt_Elektrizität_2")
# create L12
    solph.LinearTransformer(
        label="L12",
        inputs={bel1: solph.Flow()}, outputs={bel2: solph.Flow(nominal_value=400, variable_costs=100)}, conversion_factors={bel2: 0.5})
# create L21
```

```
    solph.LinearTransformer(
        label="L21",
        inputs={bel2: solph.Flow()}, outputs={bel1: solph.Flow(nominal_value=400, variable_costs=100)}, conversion_factors={bel1: 0.5})
# create Überschuss_El.2
    solph.Sink(label='Überschuss_El.2', inputs={bel2: solph.Flow()})
# create Überschuss_El.
    solph.Sink(label='Überschuss_El.1', inputs={bel1: solph.Flow()})
# create Gasvorrat
    solph.Source(label='Gasvorrat', outputs={bgas: solph.Flow(nominal_value=194397000 * number_timesteps / 8760, summed_max=1)})
# create wind
    solph.Source(label='wind', outputs={bel2: solph.Flow( actual_value=data['wind'], nominal_value=700, fixed=True, fixed_costs=20)})
# create PV
    solph.Source(label='pv', outputs={bel2: solph.Flow(, actual_value=data['pv'], nominal_value=1800, fixed=True, fixed_costs=0)})
# create Wind2
    solph.Source(label='wind2', outputs={bel1: solph.Flow( actual_value=data['wind2'], nominal_value=5000, fixed=True, fixed_costs=0)})
# create el.Verbrauch
    solph.Sink(label='el. Verbrauch', inputs={bel2: solph.Flow( actual_value=data['Verbrauch'], fixed=True, nominal_value=1)})
# create Gaskraftwerk
    solph.LinearTransformer( label="Gaskraftwerk", inputs={bgas: solph.Flow()},
        outputs={bel2: solph.Flow(nominal_value=185, variable_costs=100000)}, conversion_factors={bel2: 0.62})
# create Batteriespeicher
    solph.Storage(
        label='Batteriespeicher',
        inputs={bel2: solph.Flow(variable_costs=200)}, outputs={bel2: solph.Flow(variable_costs=200)},
        capacity_loss=0.00, initial_capacity=0, nominal_value=2000, nominal_input_capacity_ratio=1/6,
        nominal_output_capacity_ratio=1/6, inflow_conversion_factor=1, outflow_conversion_factor=0.8,
        fixed_costs=35, investment=solph.Investment(ep_costs=100),)


# Energiesystem berechnen und Ergebnisse plotten
logging.info('Optimise the energy system')
    om = solph.OperationalModel(energysystem)
    if debug:
        filename = os.path.join(
            helpers.extend_basic_path('lp_files'), 'storage_invest.lp')
        logging.info('Store lp-file in {0}.'.format(filename))
        om.write(filename, io_options={'symbolic_solver_labels': True})
```

```
logging.info('Solve the optimization problem')
    om.solve(solver=solvername, solve_kwargs={'tee': tee_switch})
    return energysystem
def get_result_dict(energysystem):
    logging.info('Check the results')
    storage = energysystem.groups['Batteriespeicher']
    myresults = outputlib.DataFramePlot(energy_system=energysystem)

    Gaskraftwerk = myresults.slice_by(obj_label='Gaskraftwerk', type='to_bus',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L12 = myresults.slice_by(obj_label='L12', type='to_bus',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L21 = myresults.slice_by(obj_label='L21', type='to_bus',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    demand = myresults.slice_by(obj_label='el. Verbrauch',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    wind = myresults.slice_by(obj_label='wind',
                         date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    pv = myresults.slice_by(obj_label='pv',
                        date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    wind2 = myresults.slice_by(obj_label='wind2',
                        date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')

    return {'Gaskraftwerk_sum': Gaskraftwerk.sum(),
          'demand_sum': demand.sum(), 'demand_max': demand.max(),
          'L12_sum': L12.sum(), 'L12_max': L12.max(),
          'L21_sum': L21.sum() ,'L21_max': L21.max(),
          'wind_sum': wind.sum(), 'wind_inst': wind.max(),
          'pv_sum': pv.sum(), 'pv_max': pv.max(),
          'wind2_sum': wind2.sum(), 'wind2_max': wind2.max(),
          'storage_cap': energysystem.results[storage][storage].invest,
          'objective': energysystem.results.objective}
def create_plots(energysystem):
logging.info('Plot the results')
    cdict = {'wind': '#5b5bae', 'pv': '#ffde32', 'wind2': '#ffde32', 'Batteriespeicher': '#42c77a', 'Gaskraftwerk': '#636f6b', 'el. Verbrauch':
'#ce4aff', 'Überschuss_El.': '#555555'}
# Plotte die eingehende Energie in elektrischen Bus für Januar
```

```
    myplot = outputlib.DataFramePlot(energy_system=energysystem)
    myplot.slice_unstacked(bus_label="Gesamt_Elektrizität_2", type="to_bus",
                           date_from="2016-01-01 00:00:00", date_to="2016-03-01 00:00:00")
#Farbliste hinzufügen
    colorlist = myplot.color_from_dict(cdict)
#El. Gesamt anzeigen
    myplot.plot(color=colorlist, linewidth=2, title="Ein Tag"), myplot.ax.legend(loc='upper right'), myplot.ax.set_ylabel('Leistung in W'),
    myplot.ax.set_xlabel('Tage'), myplot.set_datetime_ticks(date_format='%d-%m-%Y', tick_distance=24*7)
# Plotte Eigenschaften des Graphen
    fig = plt.figure(figsize=(24, 14)), plt.rc('legend', **{'fontsize': 19}), plt.rcParams.update({'font.size': 19})
    plt.style.use('grayscale'), handles, labels = myplot.io_plot(
      bus_label='Gesamt_Elektrizität_2', cdict=cdict, barorder=['pv','wind', 'wind2', 'Gaskraftwerk', 'Batteriespeicher'],
      lineorder=['el. Verbrauch', 'Batteriespeicher'], line_kwa={'linewidth': 3}, ax=fig.add_subplot(1, 1, 1),
      date_from="2016-01-01 00:00:00", date_to="2016-03-01 00:00:00",
)
    myplot.ax.set_ylabel('Leistung in kW'), myplot.ax.set_xlabel('Tage'), myplot.ax.set_title("Elektrizität Gesamt")
    myplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m-%Y'), myplot.outside_legend(handles=handles, labels=labels)
# Plotte Eigenschaften des Graphen
    fig = plt.figure(figsize=(24, 14)) plt.rc('legend', **{'fontsize': 19}) plt.rcParams.update({'font.size': 19}) plt.style.use('grayscale')

    handles, labels = myplot.io_plot(
       bus_label='Gesamt_Elektrizität_1',
       cdict=cdict, barorder=['wind2'], lineorder=['wind2'], line_kwa={'linewidth': 4}, ax=fig.add_subplot(1, 1, 1),
       date_from="2016-01-01 00:00:00", date_to="2016-03-01 00:00:00",)
    myplot.ax.set_ylabel('Leistung in W') myplot.ax.set_xlabel('Tage')  myplot.ax.set_title("Elektrizität 1")
    myplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m-%Y')  myplot.outside_legend(handles=handles, labels=labels)
    plt.show()
def run_storage_invest_example():
    logger.define_logging()
    esys = optimise_storage_size()
# esys.dump()
# esys.restore()
    import pprint as pp
    pp.pprint(get_result_dict(esys)) create_plots(esys)
if __name__ == "__main__": run_storage_invest_example()
```

12.2. Versuchsszenario 1:Programmcode „ohne Leitung“

```
# Outputlib

from oemof import outputlib

# Default logger of oemof

from oemof.tools import logger

from oemof.tools import helpers

# import oemof base classes to create energy system objects

import logging

import os

import pandas as pd

import matplotlib.pyplot as plt

import oemof.solph as solph

def optimise_storage_size(filename="EinTagTEST2.csv", solvername='cbc',

                          debug=True, number_timesteps=48, tee_switch=True):

    logging.info('Initialize the energy system')

    date_time_index = pd.date_range('1/1/2016', periods=number_timesteps, freq='H')

    energysystem = solph.EnergySystem(timeindex=date_time_index)

# Read data file

    full_filename = os.path.join(os.path.dirname(__file__), filename)

    data = pd.read_csv(full_filename, sep=",")


# Create oemof object

    logging.info('Create oemof objects')

# create Gas_bus

    bgas = solph.Bus(label="Gasnetz")


# create Elektrizitäts_Bus

    bel2 = solph.Bus(label="Gesamt_Elektrizität_2")

# create Überschuss_El.

    solph.Sink(label='Überschuss_El.2', inputs={bel2: solph.Flow()})

# create Gasvorrat

    solph.Source(label='Gasvorrat', outputs={bgas: solph.Flow( nominal_value=194397000 * number_timesteps / 8760, summed_max=1)})

# create wind

    solph.Source(label='wind', outputs={bel2: solph.Flow( actual_value=data['wind'], nominal_value=700, fixed=True, fixed_costs=20)})

# create PV

    solph.Source(label='pv', outputs={bel2: solph.Flow( actual_value=data['pv'], nominal_value=1800, fixed=True, fixed_costs=0)})
```

```
# create Wind2
    solph.Source(label='wind2', outputs={bel2: solph.Flow( actual_value=data['wind2'], nominal_value=2500, fixed=True, fixed_costs=0)})
# create el.Verbrauch
    solph.Sink(label='el. Verbrauch', inputs={bel2: solph.Flow( actual_value=data['Verbrauch'], fixed=True, nominal_value=1)})


# create KWK
    solph.LinearTransformer(
        label="KWK",
        inputs={bgas: solph.Flow()},
        outputs={bel2: solph.Flow(nominal_value=185, variable_costs=100000)}, conversion_factors={bel2: 0.62})


# create Batteriespeicher
    solph.Storage(
        label='Batteriespeicher',
        inputs={bel2: solph.Flow(variable_costs=200)},
        outputs={bel2: solph.Flow(variable_costs=200)},
        capacity_loss=0.00, initial_capacity=0, nominal_value=2000, nominal_input_capacity_ratio=1/6,
        nominal_output_capacity_ratio=1/6, inflow_conversion_factor=1, outflow_conversion_factor=0.8,
        fixed_costs=35, investment=solph.Investment(ep_costs=100),
    )
    # Optimise the energy system and plot the results
    logging.info('Optimise the energy system')
    om = solph.OperationalModel(energysystem)
    if debug:
        filename = os.path.join(
            helpers.extend_basic_path('lp_files'), 'storage_invest.lp')
        logging.info('Store lp-file in {0}.'.format(filename))
        om.write(filename, io_options={'symbolic_solver_labels': True})


    logging.info('Solve the optimization problem')
    om.solve(solver=solvername, solve_kwargs={'tee': tee_switch})
    return energysystem


def get_result_dict(energysystem):
    logging.info('Check the results')
    storage = energysystem.groups['Batteriespeicher']
```

```
    myresults = outputlib.DataFramePlot(energy_system=energysystem)
    KWK = myresults.slice_by(obj_label='KWK', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    demand = myresults.slice_by(obj_label='el. Verbrauch',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    wind = myresults.slice_by(obj_label='wind',
                            date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    pv = myresults.slice_by(obj_label='pv',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    wind2 = myresults.slice_by(obj_label='wind2',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')


    return {'KWK_sum': KWK.sum(),
            'demand_sum': demand.sum(), 'demand_max': demand.max(),
            'wind_sum': wind.sum(), 'wind_inst': wind.max(),
            'pv_sum': pv.sum(), 'pv_max': pv.max(),
            'wind2_sum': wind2.sum(), 'wind2_max': wind2.max(),
            'storage_cap': energysystem.results[storage][storage].invest,
            'objective': energysystem.results.objective
            }
def create_plots(energysystem):
    logging.info('Plot the results')
    cdict = {'wind': '#5b5bae','pv': '#ffde32','wind2': '#ffde32','Batteriespeicher': '#42c77a',
             'KWK': '#636f6b','el. Verbrauch': '#ce4aff','Überschuss_El.': '#555555'}


# Plotte die eingehende Energie in elektrischen Bus für Januar
    myplot = outputlib.DataFramePlot(energy_system=energysystem)
    myplot.slice_unstacked(bus_label="Gesamt_Elektrizität_2", type="to_bus",
                           date_from="2016-01-01 00:00:00", date_to="2016-03-01 00:00:00")
#Farbliste hinzufügen
    colorlist = myplot.color_from_dict(cdict)
    #El. Gesamt anzeigen
    myplot.plot(color=colorlist, linewidth=2, title="Ein Tag"), myplot.ax.legend(loc='upper right'), myplot.ax.set_ylabel('Leistung in kW'),
    myplot.ax.set_xlabel('Tage'), myplot.set_datetime_ticks(date_format='%d-%m-%Y', tick_distance=24*7)
# Plotte Eigenschaften des Graphen
    fig = plt.figure(figsize=(24, 14)) plt.rc('legend', **{'fontsize': 19}), plt.rcParams.update({'font.size': 19}), plt.style.use('grayscale')
```

```
    handles, labels = myplot.io_plot(
        bus_label='Gesamt_Elektrizität_2', cdict=cdict, barorder=['pv','wind','wind2', 'Gaskraftwerk', 'Batteriespeicher'],
        lineorder=['el. Verbrauch', 'Batteriespeicher'], line_kwa={'linewidth': 3}, ax=fig.add_subplot(1, 1, 1),
        date_from="2016-01-01 00:00:00", date_to="2016-03-01 00:00:00",
        )
    myplot.ax.set_ylabel('Leistung in kW'), myplot.ax.set_xlabel('Tage'), myplot.ax.set_title("Elektrizität Gesamt")
    myplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m-%Y'), myplot.outside_legend(handles=handles, labels=labels)


# Plotte Eigenschaften des Graphen
    fig = plt.figure(figsize=(24, 14)), plt.rc('legend', **{'fontsize': 19}) plt.rcParams.update({'font.size': 19}), plt.style.use('grayscale')
    handles, labels = myplot.io_plot(
        bus_label='Gesamt_Elektrizität_2', cdict=cdict,
        barorder=['wind2'], lineorder=['wind2'], line_kwa={'linewidth': 4}, ax=fig.add_subplot(1, 1, 1),
        date_from="2016-01-01 00:00:00", date_to="2016-03-01 00:00:00",
        )
    myplot.ax.set_ylabel('Leistung in W'), myplot.ax.set_xlabel('Tage'), myplot.ax.set_title("Elektrizität 1")
    myplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m-%Y'), myplot.outside_legend(handles=handles, labels=labels)
    plt.show()
def run_storage_invest_example():
    logger.define_logging()
    esys = optimise_storage_size()
# esys.dump()
# esys.restore()
    import pprint as pp
    pp.pprint(get_result_dict(esys))
    create_plots(esys)


if __name__ == "__main__":
    run_storage_invest_example()
```

12.3. Versuchsszenario 2: Berechnungsdauer

```
# imports

# Outputlib

from oemof import outputlib

# Default logger of oemof

from oemof.tools import logger

from oemof.tools import helpers

# import oemof base classes to create energy system objects

import logging

import os

import pandas as pd

import matplotlib.pyplot as plt

import oemof.solph as solph

def optimise_storage_size(filename="S2.csv", solvername='cbc',

                          debug=True, number_timesteps=8760, tee_switch=True):

    logging.info('Initialize the energy system')

    date_time_index = pd.date_range('1/1/2016', periods=number_timesteps,

                                    freq='H')

    energysystem = solph.EnergySystem(timeindex=date_time_index)

# Read data file

    full_filename = os.path.join(os.path.dirname(__file__), filename)

    data = pd.read_csv(full_filename, sep=",")

# Create oemof object

    logging.info('Create oemof objects')

# create Gas_bus

    bgas = solph.Bus(label="Gasnetz")

# create Elektrizitäts_Bus

    bel1 = solph.Bus(label="Gesamt_Elektrizität_1")

# create Elektrizitäts_Bus

    bel2 = solph.Bus(label="Gesamt_Elektrizität_2")

# create Elektrizitäts_Bus

    bel3 = solph.Bus(label="Gesamt_Elektrizität_3")

# create Elektrizitäts_Bus

    bel4 = solph.Bus(label="Gesamt_Elektrizität_4")

# create Elektrizitäts_Bus

    bel5 = solph.Bus(label="Gesamt_Elektrizität_5")
```

```
# create Elektrizitäts_Bus
    bel6 = solph.Bus(label="Gesamt_Elektrizität_6")
# create L12
    solph.LinearTransformer( label="L12", inputs={bel1: solph.Flow()}, outputs={bel2: solph.Flow(nominal_value=4000, variable_costs=100)},
        conversion_factors={bel2: 0.9})
# create L21
    solph.LinearTransformer( label="L21", inputs={bel2: solph.Flow()}, outputs={bel1: solph.Flow(nominal_value=4000, variable_costs=100)},
        conversion_factors={bel1: 0.9})
# create L23
    solph.LinearTransformer( label="L23", inputs={bel2: solph.Flow()}, outputs={bel3: solph.Flow(nominal_value=4000, variable_costs=100)},
        conversion_factors={bel3: 0.9})
# create L32
    solph.LinearTransformer( label="L32", inputs={bel3: solph.Flow()}, outputs={bel2: solph.Flow(nominal_value=4000, variable_costs=100)},
        conversion_factors={bel2: 0.9})
# create L34
    solph.LinearTransformer( label="L34", inputs={bel3: solph.Flow()}, outputs={bel4: solph.Flow(nominal_value=4000, variable_costs=100)},
        conversion_factors={bel4: 0.9})
# create L43
    solph.LinearTransformer( label="L43", inputs={bel4: solph.Flow()}, outputs={bel3: solph.Flow(nominal_value=4000, variable_costs=100)},
        conversion_factors={bel3: 0.9})
# create L45
    solph.LinearTransformer( label="L45", inputs={bel4: solph.Flow()}, outputs={bel5: solph.Flow(nominal_value=4000, variable_costs=100)},
        conversion_factors={bel5: 0.9})
# create L54
    solph.LinearTransformer( label="L54", inputs={bel5: solph.Flow()}, outputs={bel4: solph.Flow(nominal_value=4000, variable_costs=100)},
        conversion_factors={bel4: 0.9})
# create L56
    solph.LinearTransformer(
        label="L56", inputs={bel5: solph.Flow()}, outputs={bel6: solph.Flow(nominal_value=4000, variable_costs=100)},
        conversion_factors={bel6: 0.9})
# create L65
    solph.LinearTransformer(
        label="L65", inputs={bel6: solph.Flow()}, outputs={bel5: solph.Flow(nominal_value=4000, variable_costs=100)},
        conversion_factors={bel5: 0.9})
# create L61
    solph.LinearTransformer( label="L61", inputs={bel6: solph.Flow()}, outputs={bel1: solph.Flow(nominal_value=4000, variable_costs=100)},
```

```
        conversion_factors={bel1: 0.9})
# create L16
    solph.LinearTransformer( label="L16", inputs={bel1: solph.Flow()}, outputs={bel6: solph.Flow(nominal_value=4000, variable_costs=100)},
        conversion_factors={bel6: data['wind']})
# Shortage bel1 shortage
    solph.Source(label='shortage1', outputs={bel1: solph.Flow(variable_costs=50000)})
# Shortage bel2
    solph.Source(label='shortage2', outputs={bel2: solph.Flow(variable_costs=50000)})
# Shortage bel3
    solph.Source(label='shortage3', outputs={bel3: solph.Flow(variable_costs=50000)})
# Shortage bel4
    solph.Source(label='shortage4', outputs={bel4: solph.Flow(variable_costs=50000)})
# Shortage bel5
    solph.Source(label='shortage5', outputs={bel5: solph.Flow(variable_costs=50000)})
# Shortage bel6
    solph.Source(label='shortage6', outputs={bel6: solph.Flow(variable_costs=50000)})
# create Überschuss_El.1
    solph.Sink(label='Überschuss_El.1', inputs={bel1: solph.Flow()})
# create Überschuss_El.2
    solph.Sink(label='Überschuss_El.2', inputs={bel2: solph.Flow()})
# create Überschuss_El.3
    solph.Sink(label='Überschuss_El.3', inputs={bel3: solph.Flow()})
# create Überschuss_El.4
    solph.Sink(label='Überschuss_El.4', inputs={bel4: solph.Flow()})
# create Überschuss_El.5
    solph.Sink(label='Überschuss_El.5', inputs={bel5: solph.Flow()})
# create Überschuss_El.6
    solph.Sink(label='Überschuss_El.6', inputs={bel6: solph.Flow()})
# create Überschuss_gas
    solph.Sink(label='Überschuss_gas', inputs={bgas: solph.Flow()})
# create Gasvorrat
    solph.Source(label='Gasvorrat', outputs={bgas: solph.Flow(variable_costs=100)})
# create wind
    solph.Source(label='wind', outputs={bel2: solph.Flow(actual_value=data['wind'], nominal_value=700, fixed=True, fixed_costs=20)})
# create Wind2
    solph.Source(label='wind2', outputs={bel1: solph.Flow(actual_value=data['wind'], nominal_value=5000, fixed=True, fixed_costs=20)})
```

```
# create Wind3
   solph.Source(label='wind3', outputs={bel3: solph.Flow(actual_value=data['wind'], nominal_value=5000, fixed=True, fixed_costs=20)})
# create Wind4
   solph.Source(label='wind4', outputs={bel4: solph.Flow(actual_value=data['wind'], nominal_value=5000, fixed=True, fixed_costs=20)})
# create Wind5
   solph.Source(label='wind5', outputs={bel5: solph.Flow(actual_value=data['wind'], nominal_value=5000, fixed=True, fixed_costs=20)})
# create Wind6
   solph.Source(label='wind6', outputs={bel6: solph.Flow(actual_value=data['wind'], nominal_value=5000, fixed=True, fixed_costs=20)})
# create PV
   solph.Source(label='pv', outputs={bel2: solph.Flow( actual_value=data['pv'], nominal_value=1800, fixed=True, fixed_costs=20)})
# create el.Verbrauch
   solph.Sink(label='el. Verbrauch', inputs={bel2: solph.Flow( actual_value=data['Verbrauch'] / 250, fixed=True, nominal_value=1)})
# create el.Verbrauch
   solph.Sink(label='el. Verbrauch1', inputs={bel1: solph.Flow( actual_value=data['Verbrauch'] / 250, fixed=True, nominal_value=1)})
# create el.Verbrauch
   solph.Sink(label='el. Verbrauch3', inputs={bel3: solph.Flow( actual_value=data['Verbrauch'] / 250, fixed=True, nominal_value=1)})
# create el.Verbrauch
   solph.Sink(label='el. Verbrauch4', inputs={bel4: solph.Flow( actual_value=data['Verbrauch'] / 250, fixed=True, nominal_value=1)})
# create el.Verbrauch
   solph.Sink(label='el. Verbrauch5', inputs={bel5: solph.Flow( actual_value=data['Verbrauch'] / 250, fixed=True, nominal_value=1)})
# create el.Verbrauch
   solph.Sink(label='el. Verbrauch6', inputs={bel6: solph.Flow( actual_value=data['Verbrauch'] / 250, fixed=True, nominal_value=1)})
# create Gaskraftwerk
   solph.LinearTransformer(
      label="Gaskraftwerk", inputs={bgas: solph.Flow()}, outputs={bel2: solph.Flow(nominal_value=185)}, conversion_factors={bel2: 0.62})
# create Batteriespeicher
   solph.Storage(
      label='Batteriespeicher',
      inputs={bel2: solph.Flow(variable_costs=200)}, outputs={bel2: solph.Flow(variable_costs=200)},
      capacity_loss=0.00, initial_capacity=0, nominal_value=2000, nominal_input_capacity_ratio=1/6,
      nominal_output_capacity_ratio=1/6, inflow_conversion_factor=1, outflow_conversion_factor=0.8,
      fixed_costs=35, investment=solph.Investment(ep_costs=100),
   )
# Optimiere das Energiesystem und berechne die Durchflüsse
   logging.info('Optimise the energy system')
   om = solph.OperationalModel(energysystem)
```

```
    if debug:
        filename = os.path.join(
            helpers.extend_basic_path('lp_files'), 'storage_invest.lp')
        logging.info('Store lp-file in {0}.'.format(filename))
        om.write(filename, io_options={'symbolic_solver_labels': True})

    logging.info('Solve the optimization problem')
    om.solve(solver=solvername, solve_kwargs={'tee': tee_switch})
    return energysystem
def get_result_dict(energysystem):
    logging.info('Check the results')
    storage = energysystem.groups['Batteriespeicher']
    myresults = outputlib.DataFramePlot(energy_system=energysystem)

    Gaskraftwerk = myresults.slice_by( obj_label='Gaskraftwerk', type='to_bus',
        date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L12 = myresults.slice_by(obj_label='L12', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L21 = myresults.slice_by(obj_label='L21', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L23 = myresults.slice_by(obj_label='L23', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L32 = myresults.slice_by(obj_label='L32', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L34 = myresults.slice_by(obj_label='L34', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L43 = myresults.slice_by(obj_label='L43', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L45 = myresults.slice_by(obj_label='L45', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L54 = myresults.slice_by(obj_label='L54', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L56 = myresults.slice_by(obj_label='L56', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L65 = myresults.slice_by(obj_label='L65', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
```

```
    L61 = myresults.slice_by(obj_label='L61', type='to_bus',
                             date_from='2016-01-01 00:00:00',date_to='2016-12-31 23:00:00')
    L16 = myresults.slice_by(obj_label='L16', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    demand = myresults.slice_by(obj_label='el. Verbrauch',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    wind = myresults.slice_by(obj_label='wind',
                            date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    pv = myresults.slice_by(obj_label='pv',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    wind2 = myresults.slice_by(obj_label='wind2',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    wind3 = myresults.slice_by(obj_label='wind3',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    wind4 = myresults.slice_by(obj_label='wind4',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    wind5 = myresults.slice_by(obj_label='wind5',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    wind6 = myresults.slice_by(obj_label='wind6',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    return {'Gaskraftwerk_sum': Gaskraftwerk.sum()['val'],'demand_sum': demand.sum()['val'], 'demand_max': demand.max()['val'],
            'L12_sum': L12.sum()['val'], 'L12_max': L12.max()['val'], 'L21_sum': L21.sum()['val'], 'L21_max': L21.max()['val'],
            'L23_sum': L23.sum()['val'], 'L23_max': L23.max()['val'], 'L32_sum': L32.sum()['val'], 'L32_max': L32.max()['val'],
            'L34_sum': L34.sum()['val'], 'L34_max': L34.max()['val'], 'L43_sum': L43.sum()['val'], 'L43_max': L43.max()['val'],
            'L45_sum': L45.sum()['val'], 'L45_max': L45.max()['val'] ,'L54_sum': L54.sum()['val'], 'L54_max': L54.max()['val'],
            'L56_sum': L56.sum()['val'], 'L56_max': L56.max()['val'], 'L65_sum': L65.sum()['val'], 'L65_max': L65.max()['val'],
            'L61_sum': L61.sum()['val'], 'L61_max': L61.max()['val'], 'L16_sum': L16.sum()['val'], 'L16_max': L16.max()['val'],
            'wind_sum': wind.sum()['val'], 'wind_inst': wind.max()['val'], 'pv_sum': pv.sum()['val'], 'pv_max': pv.max()['val'],
            'wind2_sum': wind2.sum()['val'], 'wind2_max': wind2.max()['val'], 'wind3_sum': wind3.sum()['val'], 'wind3_max': wind3.max()['val'],
            'wind4_sum': wind4.sum()['val'], 'wind4_max': wind4.max()['val'], 'wind5_sum': wind5.sum()['val'], 'wind5_max': wind5.max()['val'],
            'wind6_sum': wind6.sum()['val'], 'wind6_max': wind6.max()['val'], 'storage_cap': energysystem.results[storage][storage].invest,
            'objective': energysystem.results.objective
            }
def create_plots(energysystem):
    logging.info('Plot the results')
    cdict = {'wind': '#5b5bae', 'pv': '#ffca00', 'wind2': '#5b5bae', 'wind3': '#5b5bae', 'Batteriespeicher': '#42c77a',
```

```
        'Gaskraftwerk': '#636f6b', 'el. Verbrauch': '#ce4aff', 'Überschuss_El.': '#555555',

        'L34': '#aebfff', 'L43': '#aebfff', 'L32': '#85d5ff', 'L23': '#85d5ff', 'L12': '#aebfff','L21': '#aebfff',

        'shortage2': '#ebff00','shortage3': '#ebff00',}

# Plotte den Energiefluss des elektrischen Bus 2 für zwei Tage

  myplot = outputlib.DataFramePlot(energy_system=energysystem) myplot.slice_unstacked(bus_label="Gesamt_Elektrizität_2",
type="to_bus", date_from="2016-01-01 00:00:00", date_to="2016-03-01 00:00:00")

# Farbliste hinzufügen

  colorlist = myplot.color_from_dict(cdict)

# El. Gesamt anzeigen

  myplot.plot(color=colorlist, linewidth=2, title="Ein Tag") myplot.ax.legend(loc='upper right')

  myplot.ax.set_ylabel('Leistung in W') myplot.ax.set_xlabel('Tage')

  myplot.set_datetime_ticks(date_format='%d-%m-%Y', tick_distance=24*7)

  plt.show()

  fig = plt.figure()

  handles, labels = myplot.io_plot(

    bus_label='Gesamt_Elektrizität_2', cdict=cdict,

    barorder=['pv', 'wind', 'Gaskraftwerk', 'Batteriespeicher', 'L12', 'L32', 'shortage2'],

    lineorder=['el. Verbrauch', 'Batteriespeicher', 'L21', 'L23', 'Überschuss_El.2'],

    ax=fig.add_subplot(1, 1, 1),

    line_kwa={'linewidth': 3},

    date_from="2016-02-01 00:00:00", date_to="2016-03-01 00:00:00",

    )

  myplot.ax.set_ylabel('Leistung in kW')

  myplot.ax.set_xlabel('Tage')

  myplot.ax.set_title("Elektrizität Gesamt")

  myplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m-%Y')

  myplot.outside_legend(handles=handles, labels=labels)

  plt.show()

  fig = plt.figure()

  handles, labels = myplot.io_plot(

    bus_label='Gesamt_Elektrizität_3', cdict=cdict,

    barorder=['pv', 'wind3', 'Gaskraftwerk', 'Batteriespeicher', 'L43', 'L23', 'shortage3'],

    lineorder=['el. Verbrauch3', 'L34', 'L32','Überschuss_El.3'],

    ax=fig.add_subplot(1, 1, 1),

    line_kwa={'linewidth': 3},

    date_from="2016-02-01 00:00:00", date_to="2016-03-01 00:00:00",

    )
```

```
    myplot.ax.set_ylabel('Leistung in kW')
    myplot.ax.set_xlabel('Tage')
    myplot.ax.set_title("Elektrizität Gesamt")
    myplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m-%Y')
    myplot.outside_legend(handles=handles, labels=labels)
    plt.show()
def run_storage_invest_example():
    logger.define_logging()
    esys = optimise_storage_size(solvername='cbc', debug=False)
# esys.dump()
# esys.restore()
    import pprint as pp
    pp.pprint(get_result_dict(esys))
    create_plots(esys)
if __name__ == "__main__":
    run_storage_invest_example()
```

12.4. Versuchsszenario 3: Optimierung von Koppelstellen

```
# Outputlib
from oemof import outputlib
# Default logger of oemof
from oemof.tools import logger
from oemof.tools import helpers
# import oemof base classes to create energy system objects
import logging
import os
import pandas as pd
import matplotlib.pyplot as plt
import oemof.solph as solph
def optimise_storage_size(filename="EinTagTEST2.csv", solvername='cbc', debug=True, number_timesteps=48, tee_switch=True):
    logging.info('Initialize the energy system')
    date_time_index = pd.date_range('1/1/2016', periods=number_timesteps, freq='H')
    energysystem = solph.EnergySystem(timeindex=date_time_index)

# Read data file
    full_filename = os.path.join(os.path.dirname(__file__), filename)
    data = pd.read_csv(full_filename, sep=",")
# Create oemof object
    logging.info('Create oemof objects')
# create Gas_bus
    bgas1 = solph.Bus(label="Gasnetz1")
# create Gas_bus2
    bgas2 = solph.Bus(label="Gasnetz2")
# create Gas_bus3
    bgas3 = solph.Bus(label="Gasnetz3")
# create Elektrizitäts_Bus
    bel1 = solph.Bus(label="Gesamt_Elektrizität_1")
# create Elektrizitäts_Bus
    bel2 = solph.Bus(label="Gesamt_Elektrizität_2")
# create Elektrizitäts_Bus
    bel3 = solph.Bus(label="Gesamt_Elektrizität_3")
# create Wärme_Bus
    bwarm1 = solph.Bus(label="Wärmenetz1")
```

```
# create Gas_bus2
   bwarm2 = solph.Bus(label="Wärmenetz2")
# create Gas_bus3
   bwarm3 = solph.Bus(label="Wärmenetz3")
# Elektrische Leitungen erstellen
# create L12
   solph.LinearTransformer(
      label="L12",inputs={bel1: solph.Flow()}, outputs={bel2: solph.Flow(nominal_value=400, variable_costs=100)},
      conversion_factors={bel2: 0.8})
# create L21
   solph.LinearTransformer(
      label="L21", inputs={bel2: solph.Flow()}, outputs={bel1: solph.Flow(nominal_value=400, variable_costs=100)},
      conversion_factors={bel1: 0.8})
# create L23
   solph.LinearTransformer(
      label="L23", inputs={bel2: solph.Flow()}, outputs={bel3: solph.Flow(nominal_value=400, variable_costs=100)},
      conversion_factors={bel3: 0.8})
# create L32
   solph.LinearTransformer(
      label="L32", inputs={bel3: solph.Flow()}, outputs={bel2: solph.Flow(nominal_value=400, variable_costs=100)},
      conversion_factors={bel2: 0.8})
# create L31
   solph.LinearTransformer(
      label="L31", inputs={bel3: solph.Flow()}, outputs={bel1: solph.Flow(nominal_value=400, variable_costs=100)},
      conversion_factors={bel1: 0.8})
# create L13
   solph.LinearTransformer(
      label="L13", inputs={bel1: solph.Flow()}, outputs={bel3: solph.Flow(nominal_value=400, variable_costs=100)},
      conversion_factors={bel3: 0.8})
# Gas Leitungen
# create g12
   solph.LinearTransformer(
      label="g12", inputs={bgas1: solph.Flow()}, outputs={bgas2: solph.Flow(nominal_value=300, variable_costs=100)},
      conversion_factors={bgas2: 0.8})
# create g21
   solph.LinearTransformer(
```

```
        label="g21", inputs={bgas2: solph.Flow()}, outputs={bgas1: solph.Flow(nominal_value=400, variable_costs=100)},
        conversion_factors={bgas1: 0.8})
# create g23
    solph.LinearTransformer(
        label="g23", inputs={bgas2: solph.Flow()}, outputs={bgas3: solph.Flow(nominal_value=400, variable_costs=100)},
        conversion_factors={bgas3: 0.8})
# create g32
    solph.LinearTransformer(
        label="g32", inputs={bgas3: solph.Flow()}, outputs={bgas2: solph.Flow(nominal_value=400, variable_costs=100)},
        conversion_factors={bgas2: 0.8})
# create g31
    solph.LinearTransformer(
        label="g31", inputs={bgas3: solph.Flow()}, outputs={bgas1: solph.Flow(nominal_value=400, variable_costs=100)},
        conversion_factors={bgas1: 0.8})
# create g13
    solph.LinearTransformer(
        label="g13", inputs={bgas1: solph.Flow()}, outputs={bgas3: solph.Flow(nominal_value=400, variable_costs=100)},
        conversion_factors={bgas3: 0.8})
# Wärmenetz Leitungen erstellen
# create w12
    solph.LinearTransformer(
        label="w12",
        inputs={bwarm1: solph.Flow()}, outputs={bwarm2: solph.Flow(nominal_value=300, variable_costs=100)},
        conversion_factors={bwarm2: 0.8})
# create w21
    solph.LinearTransformer(
        label="w21", inputs={bwarm2: solph.Flow()}, outputs={bwarm1: solph.Flow(nominal_value=400, variable_costs=100)},
        conversion_factors={bwarm1: 0.8})
# create w23
    solph.LinearTransformer(
        label="w23", inputs={bwarm2: solph.Flow()}, outputs={bwarm3: solph.Flow(nominal_value=400, variable_costs=100)},
        conversion_factors={bwarm3: 0.8})


# create w32
    solph.LinearTransformer(
        label="w32", inputs={bwarm3: solph.Flow()}, outputs={bwarm2: solph.Flow(nominal_value=400, variable_costs=100)},
```

```
        conversion_factors={bwarm2: 0.8})
# create w31
    solph.LinearTransformer(
        label="w31", inputs={bwarm3: solph.Flow()}, outputs={bwarm1: solph.Flow(nominal_value=400, variable_costs=100)},
        conversion_factors={bwarm1: 0.8})
# create w13
    solph.LinearTransformer(
        label="w13", inputs={bwarm1: solph.Flow()}, outputs={bwarm3: solph.Flow(nominal_value=400, variable_costs=100)},
        conversion_factors={bwarm3: 0.8})
# create Überschuss_El.
    solph.Sink(label='Überschuss_El.1', inputs={bel1: solph.Flow()})
# create Überschuss_El.
    solph.Sink(label='Überschuss_El.2', inputs={bel2: solph.Flow()})
# create Überschuss_El.
    solph.Sink(label='Überschuss_El.3', inputs={bel3: solph.Flow()})
# create Gasvorrat
    solph.Source(label='Gasvorrat', outputs={bgas1: solph.Flow( nominal_value=194397000 * number_timesteps / 8760, summed_max=1)})
# create wind
    solph.Source(label='wind', outputs={bel2: solph.Flow( actual_value=data['wind'], nominal_value=700, fixed=True, fixed_costs=20)})
# create Wind2
    solph.Source(label='wind2', outputs={bel1: solph.Flow( actual_value=data['wind2'], nominal_value=5000, fixed=True, fixed_costs=0)})
# create Wind3
    solph.Source(label='wind3', outputs={bel3: solph.Flow( actual_value=data['wind2'], nominal_value=5000, fixed=True, fixed_costs=0)})
# create PV
    solph.Source(label='pv', outputs={bel3: solph.Flow( actual_value=data['pv'], nominal_value=1800, fixed=True, fixed_costs=0)})
# create el.Verbrauch
    solph.Sink(label='el. Verbrauch', inputs={bel2: solph.Flow( actual_value=data['Verbrauch'], fixed=True, nominal_value=1)})
# create Gaskraftwerk
    solph.LinearTransformer(
        label="Gaskraftwerk", inputs={bgas2: solph.Flow()}, outputs={bel2: solph.Flow(nominal_value=2000, variable_costs=10000)},
        conversion_factors={bel2: 0.62})
# create KWK
    solph.LinearTransformer(
        label="KraftWärmeKopplung",
        inputs={bgas1: solph.Flow()},
        outputs={bel1: solph.Flow(nominal_value=100, variable_costs=50), bwarm1: solph.Flow(nominal_value=100, variable_costs=50)},
```

```
        conversion_factors={bel1: 0.21, bwarm1: 0.40})
# create Batteriespeicher
    solph.Storage(
        label='Batteriespeicher',
        inputs={bel2: solph.Flow(variable_costs=200)}, outputs={bel2: solph.Flow(variable_costs=200)},
        capacity_loss=0.00, initial_capacity=0, nominal_value=2000, nominal_input_capacity_ratio=1/6,
        nominal_output_capacity_ratio=1/6, inflow_conversion_factor=1, outflow_conversion_factor=0.8, fixed_costs=35,
        investment=solph.Investment(ep_costs=100),
    )
# Optimiere das Energiesystem und Ergebnisse angeben
    logging.info('Optimise the energy system')
    om = solph.OperationalModel(energysystem)
    if debug:
        filename = os.path.join(
            helpers.extend_basic_path('lp_files'), 'storage_invest.lp')
        logging.info('Store lp-file in {0}.'.format(filename))
        om.write(filename, io_options={'symbolic_solver_labels': True})

    logging.info('Solve the optimization problem')
    om.solve(solver=solvername, solve_kwargs={'tee': tee_switch})
    return energysystem
def get_result_dict(energysystem):
    logging.info('Check the results')
    storage = energysystem.groups['Batteriespeicher']
    myresults = outputlib.DataFramePlot(energy_system=energysystem)

    Gaskraftwerk = myresults.slice_by(obj_label='Gaskraftwerk', type='to_bus',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    KraftWärmeKopplung = myresults.slice_by(obj_label='KraftWärmeKopplung', type='to_bus',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L12 = myresults.slice_by(obj_label='L12', type='to_bus',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L21 = myresults.slice_by(obj_label='L21', type='to_bus',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L23 = myresults.slice_by(obj_label='L23', type='to_bus',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
```

```
    L32 = myresults.slice_by(obj_label='L32', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L31 = myresults.slice_by(obj_label='L31', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    L13 = myresults.slice_by(obj_label='L13', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
#Gasnetz
    g12 = myresults.slice_by(obj_label='g12', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    g21 = myresults.slice_by(obj_label='g21', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
       g23 = myresults.slice_by(obj_label='g23', type='to_bus',
                             date_from='2016-01-01 00:00:00',date_to='2016-12-31 23:00:00')
    g32 = myresults.slice_by(obj_label='g32', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
       g31 = myresults.slice_by(obj_label='g31', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    g13 = myresults.slice_by(obj_label='g13', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
#Wärmenetz
       w12 = myresults.slice_by(obj_label='w12', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    w21 = myresults.slice_by(obj_label='w21', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    w23 = myresults.slice_by(obj_label='w23', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    w32 = myresults.slice_by(obj_label='w32', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    w31 = myresults.slice_by(obj_label='w31', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    w13 = myresults.slice_by(obj_label='w13', type='to_bus',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    demand = myresults.slice_by(obj_label='el. Verbrauch',
                             date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    wind = myresults.slice_by(obj_label='wind',
                          date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
```

```
    pv = myresults.slice_by(obj_label='pv',
                        date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    wind2 = myresults.slice_by(obj_label='wind2',
                        date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    wind3 = myresults.slice_by(obj_label='wind3',
                        date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
#Durchflüsse der Objekte angeben
    return {'Gaskraftwerk_sum': Gaskraftwerk.sum(),
            'KraftWärmeKopplung_sum': KraftWärmeKopplung.sum(),
            'demand_sum': demand.sum(), 'demand_max': demand.max(),
            'L12_sum': L12.sum(), 'L12_max': L12.max(), 'L21_sum': L21.sum(), 'L21_max': L21.max(),
            'L23_sum': L23.sum(), 'L23_max': L23.max(), 'L32_sum': L32.sum(), 'L32_max': L32.max(),
            'L31_sum': L31.sum(), 'L31_max': L31.max(), 'L13_sum': L13.sum(),'L13_max': L13.max(),
            'g12_sum': g12.sum(), 'g12_max': g12.max(), 'g21_sum': g21.sum(), 'g21_max': g21.max(),
            'g23_sum': g23.sum(),'g23_max': g23.max(),'g32_sum': g32.sum(),'g32_max': g32.max(),
            'g31_sum': g31.sum(), 'g31_max': g31.max(), 'g13_sum': g13.sum(), 'g13_max': g13.max(),
            'w12_sum': w12.sum(), 'w12_max': w12.max(), 'w21_sum': w21.sum(), 'w21_max': w21.max(),
            'w23_sum': w23.sum(), 'w23_max': w23.max(),'w32_sum': w32.sum(), 'w32_max': w32.max(),
            'w31_sum': w31.sum(), 'w31_max': w31.max(), 'w13_sum': w13.sum(),'w13_max': w13.max(),
           'wind_sum': wind.sum(), 'wind_inst': wind.max(), 'pv_sum': pv.sum(), 'pv_max': pv.max(),
            'wind2_sum': wind2.sum(),'wind2_max': wind2.max(), 'wind3_sum': wind3.sum(), 'wind3_max': wind3.max(),
            'storage_cap': energysystem.results[storage][storage].invest, 'objective': energysystem.results.objective }
def create_plots(energysystem):
    logging.info('Plot the results')
    cdict = {'wind': '#5b5bae', 'pv': '#ffde32', 'wind2': '#ffde32', 'wind3': '#ffde32', 'Batteriespeicher': '#42c77a', 'Gaskraftwerk': '#636f6b',
'el. Verbrauch': '#ce4aff', 'Überschuss_El.': '#555555'}
# Plotte die Ergebnise des elektrischen Busses für zwei Tage
    myplot = outputlib.DataFramePlot(energy_system=energysystem)
    myplot.slice_unstacked(bus_label="Gesamt_Elektrizität_2", type="to_bus",
                        date_from="2016-01-01 00:00:00", date_to="2016-03-01 00:00:00")

#Farbliste hinzufügen
    colorlist = myplot.color_from_dict(cdict)
#Elektrischen Bus anzeigen
    myplot.plot(color=colorlist, linewidth=2, title="Ein Tag") myplot.ax.legend(loc='upper right')
    myplot.ax.set_ylabel('Leistung in W') myplot.ax.set_xlabel('Tage')
```

```
    myplot.set_datetime_ticks(date_format='%d-%m-%Y', tick_distance=24*7)
# Plotting a combined stacked plot
    fig = plt.figure(figsize=(24, 14))
    plt.rc('legend', **{'fontsize': 19})
    plt.rcParams.update({'font.size': 19})
    plt.style.use('grayscale')
handles, labels = myplot.io_plot(
        bus_label='Gesamt_Elektrizität_2', cdict=cdict,
        barorder=['pv','wind', 'wind2', 'wind2', 'Gaskraftwerk', 'Batteriespeicher'],
        lineorder=['el. Verbrauch', 'Batteriespeicher'],
        line_kwa={'linewidth': 3}, ax=fig.add_subplot(1, 1, 1),
        date_from="2016-01-01 00:00:00", date_to="2016-03-01 00:00:00", )
myplot.ax.set_ylabel('Leistung in kW') myplot.ax.set_xlabel('Tage') myplot.ax.set_title("Elektrizität Gesamt")
myplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m-%Y') myplot.outside_legend(handles=handles, labels=labels)
# Plotting a combined stacked plot
    fig = plt.figure(figsize=(24, 14)) plt.rc('legend', **{'fontsize': 19}) plt.rcParams.update({'font.size': 19}) plt.style.use('grayscale')
    handles, labels = myplot.io_plot(
        bus_label='Gesamt_Elektrizität_1', cdict=cdict, barorder=['wind2'], lineorder=['wind2'], line_kwa={'linewidth': 4},
        ax=fig.add_subplot(1, 1, 1),
        date_from="2016-01-01 00:00:00", date_to="2016-03-01 00:00:00",)
    myplot.ax.set_ylabel('Leistung in W') myplot.ax.set_xlabel('Tage') myplot.ax.set_title("Elektrizität 1")
    myplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m-%Y') myplot.outside_legend(handles=handles, labels=labels)
    plt.show()
def run_storage_invest_example():
    logger.define_logging()
    esys = optimise_storage_size()
# esys.dump()
# esys.restore()
    import pprint as pp
    pp.pprint(get_result_dict(esys))
    create_plots(esys)
if __name__ == "__main__":
    run_storage_invest_example()
```

12.5. Versuchsszenario 4: Vergleich Wärmepumpe

```
# imports
# Outputlib
from oemof import outputlib
# Default logger of oemof
from oemof.tools import logger
from oemof.tools import helpers
# import oemof base classes to create energy system objects
import logging
import os
import pandas as pd
import matplotlib.pyplot as plt
import oemof.solph as solph
def optimise_storage_size(filename="WS5.csv", solvername='cbc', debug=True, number_timesteps=8760, tee_switch=True):
    logging.info('Initialize the energy system')
    date_time_index = pd.date_range('1/1/2016', periods=number_timesteps, freq='H')
    energysystem = solph.EnergySystem(timeindex=date_time_index)
# Read data file
    full_filename = os.path.join(os.path.dirname(__file__), filename)
    data = pd.read_csv(full_filename, sep=",")
# Create oemof object
    logging.info('Create oemof objects')
# create Strombus
    bel = solph.Bus(label="Stromnetz")
# create Wärme-Bus
    bwarm = solph.Bus(label="Gesamt_Wärme")
# create Überschuss_warm
    solph.Sink(label='Überschuss_warm', inputs={bwarm: solph.Flow()})
# create Überschuss_el
    solph.Sink(label='Überschuss_el', inputs={bel: solph.Flow()})
# create Stromnetzanschluss
    solph.Source(label='Stromnetzanschluss',outputs={bel: solph.Flow(variable_costs=1000)})
# create Wärmeverbrauch
    solph.Sink(label='Wärmeverbrauch', inputs={bwarm: solph.Flow(actual_value=data['Verbrauch'], fixed=True, nominal_value=1)})
# create Stromverbrauch
    solph.Sink(label='Stromverbrauch', inputs={bwarm: solph.Flow(actual_value=data['Verbrauch'], fixed=True, nominal_value=1)})
```

```
# create Elektro-Wärmepumpenheizung
    solph.LinearTransformer(label="Pumpe", inputs={bel: solph.Flow()}, outputs={bwarm: solph.Flow(nominal_value=4, variable_costs=5)},
        conversion_factors={bwarm:data['Faktor']})
# create Wärmespeicher
    solph.Storage(
        label='Wärmespeicher', inputs={bwarm: solph.Flow(variable_costs=20)}, outputs={bwarm: solph.Flow(variable_costs=20)},
        capacity_loss=0.00, initial_capacity=0, nominal_value=11.47, inflow_conversion_factor=1, outflow_conversion_factor=0.95,
        fixed_costs=35, investment=solph.Investment(ep_costs=100),)
# create Photovoltaikanlage
    solph.Source( label='Photovoltaikanlage', outputs={bel: solph.Flow(actual_value=data['PV'], nominal_value=1, fixed=True,
        fixed_costs=20)})
# Optimiere das Energiesystem und plotte die Ergebnisse
    logging.info('Optimise the energy system')
    om = solph.OperationalModel(energysystem)
    if debug:
        filename = os.path.join(
        helpers.extend_basic_path('lp_files'), 'storage_invest.lp')
        logging.info('Store lp-file in {0}.'.format(filename))
        om.write(filename, io_options={'symbolic_solver_labels': True})
    logging.info('Solve the optimization problem')
    om.solve(solver=solvername, solve_kwargs={'tee': tee_switch})
    return energysystem
def get_result_dict(energysystem):
    logging.info('Check the results')
    storage = energysystem.groups['Wärmespeicher']
    myresults = outputlib.DataFramePlot(energy_system=energysystem)
    Pumpe = myresults.slice_by(
        obj_label='Pumpe', type='to_bus', date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    demand = myresults.slice_by(obj_label='Wärmeverbrauch', date_from='2016-01-01 00:00:00',date_to='2016-12-31 23:00:00')
    demand = myresults.slice_by(obj_label='Wärmeverbrauch', date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    Photovoltaikanlage = myresults.slice_by(obj_label='Photovoltaikanlage', type='to_bus',
                date_from='2016-01-01 00:00:00', date_to='2016-12-31 23:00:00')
    return {'Pumpe_sum': Pumpe.sum()['val'], 'Pumpe_max': Pumpe.max()['val'],
            'Photovoltaikanlage_sum': Photovoltaikanlage.sum()['val'], 'Photovoltaikanlage_max': Photovoltaikanlage.max()['val'],
            'demand_sum': demand.sum()['val'], 'demand_max': demand.max()['val'],
            'storage_cap': energysystem.results[storage][storage].invest,'objective': energysystem.results.objective}
```

```
def create_plots(energysystem):

    logging.info('Plot the results')

    cdict = {'Wärmespeicher': '#42c77a', 'Pumpe': '#636f6b', 'Photovoltaikanlage': '#ffde32', 'Wärmeverbrauch': '#ce4aff', 'Überschuss_warm': '#555555',}

# Plotting the input flows of the electricity bus for January

    myplot = outputlib.DataFramePlot(energy_system=energysystem)

    myplot.slice_unstacked(bus_label="Gesamt_Wärme", type="to_bus", date_from="2016-01-01 00:00:00", date_to="2016-12-01 00:00:00")

# Farbliste hinzufügen

    colorlist = myplot.color_from_dict(cdict)

# Ein Tag

    myplot.plot(color=colorlist, linewidth=2, title="Ein Tag")

    myplot.ax.legend(loc='upper right')

    myplot.ax.set_ylabel('Leistung in W')

    myplot.ax.set_xlabel('Tage')

    myplot.set_datetime_ticks(date_format='%d-%m-%Y', tick_distance=24*7)

    plt.show()

    fig = plt.figure(figsize=(24, 14))

    plt.rc('legend', **{'fontsize': 19})

    plt.rcParams.update({'font.size': 19})

    plt.style.use('grayscale')

    handles, labels = myplot.io_plot( bus_label='Gesamt_Wärme', cdict=cdict, barorder=['Pumpe', 'Wärmespeicher', 'Wärmeverbrauch'],

        lineorder=['Wärmeverbrauch', 'Wärmespeicher'], ax=fig.add_subplot(1, 1, 1), line_kwa={'linewidth': 3},

        date_from="2016-06-01 00:00:00", date_to="2016-06-07 00:00:00", )

    myplot.ax.set_ylabel('Leistung in kW')

    myplot.ax.set_xlabel('1.Woche')

    myplot.ax.set_title("Wärme-Bus")

    myplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m-%Y')

    myplot.outside_legend(handles=handles, labels=labels)

    plt.show()

    fig = plt.figure(figsize=(24, 14))

    plt.rc('legend', **{'fontsize': 19})

    plt.rcParams.update({'font.size': 19})

    plt.style.use('grayscale')

    handles, labels = myplot.io_plot( bus_label='Stromnetz', cdict=cdict, barorder=['Pumpe', 'Photovoltaikanlage'],
ax=fig.add_subplot(1, 1, 1),

     line_kwa={'linewidth': 1}, date_from="2016-06-01 00:00:00", date_to="2016-06-07 00:00:00",)
```

```
    myplot.ax.set_ylabel('Leistung in kW')
    myplot.ax.set_xlabel('Tage')
    myplot.ax.set_title("Strom-Bus")
    myplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m-%Y')
    myplot.outside_legend(handles=handles, labels=labels)
    plt.show()
def run_storage_invest_example():
    logger.define_logging()
    esys = optimise_storage_size(solvername='cbc', debug=False)
# esys.dump()
# esys.restore()
    import pprint as pp
    pp.pprint(get_result_dict(esys))
    create_plots(esys)
if __name__ == "__main__":
    run_storage_invest_example()
```