Modeling and Simulation of Energy Grids Under Transactive Energy Markets

By
Dan Alberto Rosa de Jesús
A thesis submitted in partial fulfillment of the requirements for the degree
of
MASTER OF SCIENCE
in
COMPUTER ENGINEERING
UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS
2018

Approved by:

Wilson Rivera Gallego, Ph.D. President, Graduate Committee	Date
Manuel Rodríguez Martínez, Ph.D. Member, Graduate Committee	Date
Emmanuel Arzuaga, Ph.D. Member, Graduate Committee	Date
Hilton Alers Valentín Graduate School Representative	Date
José Colom Ustariz, Ph.D. Department Chairperson	Date

Abstract of Thesis Presented to the Graduate School
of the University of Puerto Rico in Partial Fulfillment of the
Requirements for the Degree of Master of Science in Computer Engineering

Modeling and Simulation of Energy Grids Under Transactive Energy Markets

Electric grids are an important aspect of the civil infrastructure of our society. Traditional energy grids are being modernized with the introduction of smart grids and transactive energy concepts. In a smart grid sensors, computers and communication networks are integrated into the power generation, transmission, distribution, and load elements. Transactive energy is a novel conceptual model in which distributed generators are coordinated through software, creating a type of "software-defined" electric grid and featuring a market-based mechanism to establish prices.

The main research question in this work is which methodologies and tools allow the modeling and simulation of electric grids including both physical components and energy consumption and market elements. This work then focuses on developing an optimization and simulation framework to facilitate the analysis of energy grids.

The contributions of this thesis are fourfold. First, a framework to evaluate the performance of evolutionary algorithms in the context of solving smart grid related problems. It includes a set of procedures that carry out multi-objective optimization through evolutionary algorithms and a set of metrics to measure their performance. Second, demonstrate how multiple evolutionary algorithms can be applied to a demand response case study for day-ahead load forecasting. Third, a framework to evaluate how smart grid entities behave when market prices are established under transactive energy strategies. Finally, a demonstration of how multiple multi-agent systems can be applied to simulate these strategies under high-demand smart grid scenarios.

Resumen de tesis presentada a la Escuela Graduada
de la Universidad de Puerto Rico como requisito parcial de los
requerimientos para el grado de Maestría en Ciencias de Ingeniería de Computadora

Modelamiento y Simulación de Redes de Energía Bajo Mercados de Transacción

Las redes eléctricas son un aspecto importante de la infraestructura civil de nuestra sociedad. Con el paso del tiempo, las mismas han sido modernizadas con la introducción de redes inteligentes y energía transactiva. En una red inteligente sensores, computadoras y redes de comunicación se integran en los elementos de generación, transmisión, distribución y carga de energía. La energía transactiva es un modelo conceptual novedoso en el que los generadores distribuidos se coordinan mediante software, creando un tipo de red eléctrica "definida por software" y presentando un mecanismo basado en el mercado para establecer precios.

La pregunta de investigación principal en este trabajo es cuáles metodologías y herramientas permiten modelar y simular redes de energía que incluyan tanto componentes físicos como elementos de consumo y mercado. Este trabajo se centra en el desarrollo de un marco de optimización y simulación para facilitar el análisis de las redes de energía.

Esta tesis provee cuatro contribuciones. Primero, un marco para evaluar el rendimiento de algoritmos evolutivos en el contexto de problemas relacionados a redes inteligentes. El mismo incluye un conjunto de algoritmos evolutivos y métricas para medir su rendimiento. En segundo lugar, demostrar cómo se pueden aplicar múltiples algoritmos evolutivos a un estudio de caso de respuesta a la demanda para predecir cargas. En tercer lugar, un marco para evaluar el comportamiento de entidades conectadas a redes inteligentes cuando los precios del mercado se establecen bajo estrategias de energía transactiva. Finalmente, una demostración de cómo se pueden aplicar múltiples agentes para simular estas estrategias bajo escenarios de alta demanda.

To my son, Sebastián M. Rosa Rivera and my father, Luis A. Rosa Ríos.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor, professor Wilson Rivera, for the continuous support of my Master of Science studies and related research and for his patience, motivation, and immense knowledge. His guidance helped me in all the aspects of this long way researching and writing of this thesis. I could not have imagined having a better advisor and mentor.

I would also like to thank the OASIS project team for making this research possible and for assisting me in the completion of my degree and research. Last but not least, I would like to thank my family and friends for always believing in me.

This project is funded in part by the National Science Foundation (NFS), under grant #ACI-1541106. Any opinions, findings, conclusions, or recommendations expressed in this material are those mine and do not necessarily reflect the views of NFS. I also would like to acknowledge the Chameleon Cloud team for making available such a good environment for running computing experiments on the cloud.

Contents

Abstract ii

Abstract (Spanish Version) iii

Acknowledgment vi

List of Figures xiii

List of Tables xv

1 Introduction 1

1.1 Motivation 1

1.2 Objectives 2

1.3 Contributions 3

1.4 Outline 4

2 The Optimization Framework 5

2.1 Multi-Objective Optimization 6

2.2 Smart Grid Policies 6

2.3 Evolutionary Algorithms 7

2.3.1 Multi-Objective Evolutionary Algorithm Based on Decomposition . 8

2.3.2 Indicator-Based Evolutionary Algorithm 9

2.3.3 Epsilon Domination Based Multi-Objective Evolutionary Algorithm 9
2.3.4 Covariance Matrix Adaptation Evolution Strategy 10
2.3.5 Third Evolution Step of Generalized Differential Evolution 10
2.3.6 Speed-constrained Multi-Objective Particle Swarm Optimization . . 11
2.3.7 Non-dominated Sorting Genetic Algorithm II 12
2.3.8 Strength Pareto Evolutionary Algorithm 2 12
2.4 Evolutionary Algorithm Performance Metrics 13
2.4.1 Hypervolume Indicator 14
2.4.2 Generational Distance 15
2.4.3 Epsilon Indicator 16
2.4.4 Inverted Generational Distance 17
2.4.5 Spacing 18
2.4.6 Running Time 19
2.5 Conclusion 19

3 Energy Demand Side Optimization 20
3.1 Model 20
3.1.1 Problem Formulation 21
3.2 *Platypus* Evolutionary Computing Framework 22
3.3 Scenario Implementation 24
3.4 Experimental Results 25
3.5 Conclusion 28

4 Transactive Energy Optimization 29
4.1 Transactive Energy Multi-Agent Systems 29
4.2 The Distribution-Communication Grid Models 31
4.2.1 Distributed Generation and One-way Communication 31
4.2.2 Centralized Generation and One-way Communication 32

4.2.3 Distributed Generation and Two-way Communication 34
4.2.4 Centralized Generation and Two-way Communication 35
4.3 Problem Formulations 35
4.3.1 Distributed Generation and One-way Communication 35
4.3.2 Centralized Generation and One-way Communication 36
4.3.3 Distributed Generation and Two-way Communication 37
4.3.4 Centralized Generation and Two-way Communication 37
4.4 Scenario Implementation 37
4.4.1 Mosaik Simulators and Control Mechanisms 38
4.5 Experimental Results 41
4.6 Conclusion 43

5 Conclusion and Future Work **44**

References **46**

Appendices **49**

A Optimization Platform Documentation **50**
A.1 Installing *Platypus* 50
A.1.1 Miniconda 50
A.1.2 Virtual Environment 50
A.1.3 Platypus 51
A.2 Getting the Optimization Platform 52
A.3 Using the Optimization Platform 52
A.3.1 Adding Optimization Problems 53
A.3.2 Adding Evolutionary Algorithms 54
A.3.3 Adding Performance Metrics 54
A.3.4 Entity Data 54

A.3.5 Definitions 55
A.3.6 Logs 55
A.3.7 Putting Everything Together 55

B GitHub Repositories **56**
B.1 Optimization Platform 56
B.2 Transactive Energy Multi-Agent System Simulations 56

C Virtual Machine Disk Images **57**

List of Abbreviations

CA Cost-Availability

CMA-ES Covariance Matrix Adaptation Evolution Strategy

CSV Comma Separated Values

CU Cost-Utility

DCG Distribution-Communication Grid

DER Distributed Energy Resource

DR Demand Response

DTLZ2 Deb Thiele Laumanns and Zitzler 2

EA Evolutionary Algorithm

ϵ-indicator Epsilon Indicator

ϵ-MOEA Epsilon Domination Based Multi-Objective Evolutionary Algorithm

GD Generational Distance

GDE3 Third Evolution Step of Generalized Differential Evolution

IaaS Infrastructure as a Service

IBEA Indicator-Based Evolutionary Algorithm

IGD Inverted Generational Distance

JSON JavaScript Object Notation

MAS Multi-Agent System

MOEA Multi-Objective Evolutionary Algorithm

MOEA/D Multi-Objective Evolutionary Algorithm Based on Decomposition

MOO Multi-Objective Optimization

NSGA-II Non-dominated Sorting Genetic Algorithm II

OASIS Open Access Smart Grids Services

O&M Operation & Maintenance

PAES Pareto Archived Evolution Strategy Algorithm

PF Pareto Frontier

PV Photovoltaic

QoS Quality of Service

RES Renewable Energy Source

SG Smart Grid

SLA Service Level Agreement

SMPSO Speed-constrained Multi-objective Particle Swarm Optimization

SPEA2 Strength Pareto Evolutionary Algorithm 2

TE Transactive Energy

List of Figures

2.1 Optimization framework for SGs. 5

2.2 SG policies and the SLA framework. 7

2.3 ϵ-MOEA procedure. 10

2.4 NSGA-II top level flowchart. 12

2.5 Graphic representation of the Hypervolume for two objective functions. . . 15

2.6 Graphic representation of the GD for two objective functions. 16

2.7 Graphic representation of the ϵ-indicator for two objective functions. . . 17

2.8 Graphic representation of the IGD for two objective functions. 17

2.9 Graphic representation of the spacing for two objective functions. 18

3.1 Example of a consumer load in 24 time intervals of 1 hour obtained from the dSpace. 21

3.2 Decision space of the non-convex optimization problem example. 22

3.3 Median values for the performance metrics obtained in the experiments. . . 26

3.4 IBEA and NSGA-II solution sets obtained in their first run. 27

3.5 Consumer load obtained from the dSpace and the corresponding load obtained running NSGA-II on the CU problem. 27

4.1 Representation of the proposed TE approach. 30

4.2 The Distribution-Communication Grid. 31

4.3 An example of a distributed SG topology. 32
4.4 Example of a PV energy generation profile. 33
4.5 An example of a centralized SG topology. 34
4.6 Comparison between a house load profile and its associated PV energy generation profile in the distributed one-way scenario. 41
4.7 Comparison between the total house load profile and total PV energy generation profile in the centralized one-way scenario. 42
4.8 Comparison between the total house load profile and total PV energy generation profile in the centralized two-way scenario. 42
4.9 Comparison between the renewable and non-renewable load profiles in the distributed two-way scenario for House 11. 43
4.10 Comparison of the cost per kilo-watt of house 11 as if it consumed the energy without incentives (left) and with incentives (right). 43

A.1 Optimization framework directory tree. 53

List of Tables

2.1 The performance metrics by number of citations and classification. 14

3.1 Values of the relevant variables in the CU problem. 24

3.2 Best mean values for each performance metric by EA. 26

4.1 Values of the relevant variables in the CU problem. 36

4.2 Values of the relevant variables in the CA problem. 36

4.3 Values of the constraints in the CA problem. 36

Listings

3.1 Simple example source code. 23

4.1 Partial example of a dictionary with information of a house entity. 38

4.2 A partial example of a JSON file to specify entity connections for *PYPOWER*. 39

A.1 Downloading and installing Miniconda latest version. 50

A.2 Creating and activating a *Python* 3.4.5 virtual environment. 51

A.3 Installing git. 51

A.4 Downloading and installing *Platypus* in the virtual environment. 51

A.5 *Platypus* post installation example. 51

A.6 Downloading the optimization platform and installing its dependencies. . . 52

Chapter 1

Introduction

1.1 Motivation

Electric grids constitute the cornerstone of the civil infrastructure of our society, essentially to carry out daily operations in education, health care, commerce, entertainment, defense, and government. Traditional energy grids are being modernized with the proliferation of Renewable Energy Sources (RESs) and Distributed Energy Resources (DERs), and particularly with the introduction of two new technologies: Smart Grids (SGs) and Transactive Energy (TE).

In a SG sensors, computers and communication networks are integrated into the power generation, transmission, distribution, and load elements. This enables a mechanism to gather information to control generation and demand and make decisions on the electric grid operation. A SG enables bidirectional flows of energy and control capabilities, which is a departure from traditional power grids which exhibits only one-way communication and limited control.

TE is a novel conceptual model in which distributed generators are coordinated through software, creating a type of "software-defined" electric grid and featuring a market-based mechanism to establish prices. The energy producers are independent agents, and connect to the grid to sell their electric services. For example, independent generators might be

common citizens with renewable energy systems. Using a common software platform to control the operation of the system, the independent generators are carefully coordinated to inject energy into the system without causing operational disruptions.

The convergence of the trends in energy grids described above generates new challenges in terms of modeling and simulation of energy grid infrastructures. This work focuses on developing an optimization and simulation framework to facilitate the analysis of energy grids.

1.2 Objectives

- **To investigate optimization techniques suitable for SG modeling:** The proposed approach includes a set of procedures that carry out Multi-Objective Optimizations (MOOs) through Evolutionary Algorithms (EAs) on SG models. It also includes a set of metrics that measure their performance in terms of their accuracy, diversity, and cardinality. An optimization framework is proposed where SG custom optimization problems, EAs, and performance metrics can be created and instantiated. It also allows the specification of custom constraints, data, and other parameters important to the optimization problems.

- **To implement and research TE simulation scenarios:** Several energy grid scenarios are modeled through a co-simulation framework, based on the Distribution-Communication Grid (DCG), that integrates various models at different SG architecture levels. Intelligence is added to some of the entities that are present in SGs through EAs. In this way, Multi-Agent System (MAS) or controllers can be developed to exchange data between the entities for intelligent decision-making on the parameters specified by the SG producers and consumers under high-demand scenarios.

1.3 Contributions

- **A framework to evaluate the performance of EAs in the context of solving SG related problems:** Besides containing the procedures for MOO problems in SGs, this framework also includes the implementation of performance metrics to validate the results of the optimization presented here. Although there are other researches that present the implementation of individual EAs and/or performance metrics to solve SG related problems. This is the first time that a framework with many EAs and performance metrics is developed and implemented to compare their performance for such application.

- **A demonstration of how multiple EAs can be applied to a Demand Response (DR) case study for day-ahead load forecasting:** The scenario presented here serves as a demonstration of how the optimization framework can be implemented as an optimization platform. Consumers connected to a SG have their policies taken into account and a DR strategy through the EAs is executed to curtail their loads while considering the maximum capacity of the utility. This can be used for day-ahead load forecasting of consumers.

- **A framework to evaluate how SG entities behave under different TE distribution and communication strategies:** The framework is based on the DCG that includes centralized and distributed generation of energy and one-way and two-way communication of information between consumers and producers. Four scenarios derive from it including the distributed generation and one-way communication; centralized generation and one-way communication; distributed generation and two-way communication; and centralized generation and one-way communication to evaluate the scenarios in terms of the consumer and producer energy profiles.

- **A demonstration of how multiple MASs can be applied to simulate TE strategies under high-demand SG scenarios:** The scenarios that derive from the DCG are implemented in *Mosaik*, a co-simulator framework for SGs, to demonstrate how DR strategies can be done on SG entities with MASs powered by EAs. This is the first time SG entities are provided with EA MAS intelligence for DR strategies that improves the cost, utility, and availability of SG entities.

1.4 Outline

The outline of this thesis is as follows. Chapter 2 includes a brief description of the optimization framework including the optimization problem formulation, definition of SG policies, optimization methods, and performance metrics. A DR case study is presented in Chapter 3. This Chapter also elaborates on the technical aspects of the SG model, problem formulation, implementation, and results. In Chapter 4, the TE optimization approach and its implementation is described. It also elaborates on the implementation of MASs for SG simulations and the results obtained. Finally, the conclusions and future work is presented in Chapter 5.

Chapter 2

The Optimization Framework

The definition and implementation of SG models in terms of MOO problems is currently carried out in ways that makes it hard to make changes to them to derive new ones, since the MOO problems vary from model to model and the implementation of EAs is difficult [1, 2, 3] due to the great amount of tacit knowledge related to their implementation and deployment in large-scale systems. The optimization framework presented here serves as an abstraction that allows the definition of SG models in the context of solving SG related problems without reinventing the wheel. As illustrated in Figure 2.1, this framework is composed of EAs and performance metrics that can be integrated with custom optimization problems and smart grid policies to define new SG model scenarios. The next sections in this chapter describe the framework components.

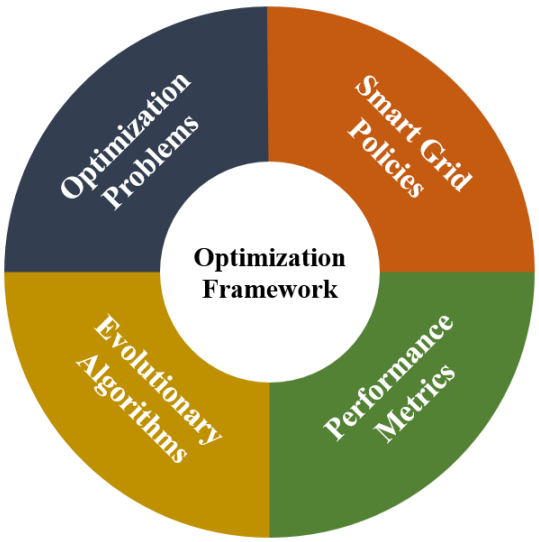

Fig 2.1: Optimization framework for SGs.

2.1 Multi-Objective Optimization

A MOO problem is an optimization problem that has two or more objectives. It can be defined as:

$$\begin{aligned} \min \quad & F(\vec{x}) = \{f_1(\vec{x}), ..., f_n(\vec{x})\} \\ \text{s.t.} \quad & G(\vec{x}) \leq 0, H(\vec{x}) = 0, \vec{x} \in \Omega, \end{aligned} \tag{2.1}$$

where $\vec{x} = (x_1, ..., x_d)$ are the decision variables, Ω is the decision space, $\mathbb{R}^n$ is the objective space, , and $G(\vec{x})$ and $H(\vec{x})$ are constraints [4]. Other definitions are:

- **Pareto Set:** the solutions in the decision space that non-dominate each other defined as $PS = \{\vec{x} \in \Omega | \nexists \vec{y} \in \Omega : F(\vec{y}) \preceq F(\vec{x})\}$
- **Pareto Frontier (PF):** it is in the objective space defined as $PF = \{F(\vec{x}) | \vec{x} \in PS\}$
- **Reference Set:** contains predefined non-dominated solutions and it is defined as RPF.

Within the scope of this research, MOO problems include objectives related to the Quality of Service (QoS) provided to customers or energy resources of producers connected to SGs. The constraints include policies like the willingness of load curtailment of consumers and the maximum generation capacity of producers. These policies are described in detail in the next section.

2.2 Smart Grid Policies

The SG policies represent the interests of the stakeholders, customers and providers, that participate in the transactions specified by a certain Service Level Agreement (SLA) framework. See Figure 2.2. These policies include, but are not limited to, the maximum or minimum energy demand and offer, percentages of load willing to curtail, etc. In [5] a SLA framework is proposed that specifies the SLA management in Open Access Smart Grids Services (OASIS) in relation to the stakeholder roles, responsibilities, and limitations. The life cycle of this SLA framework spans several phases:

- **Generation Module:** Stakeholders are authenticated and authorized to publish, modify, retrieve, or suppress service offers.

- **Negotiation Module:** Stakeholders evaluate the services available establishing, for example, the price and energy type they provide or consume.
- **Deployment Module:** The energy services are deployed considering the policies specified by the stakeholders in during negotiation.
- **Monitoring Module:** The services are checked for SLA violations and the infrastructure is improved by optimizing its services considering the level of service and policies of the stakeholders. It is here where EAs become relevant.

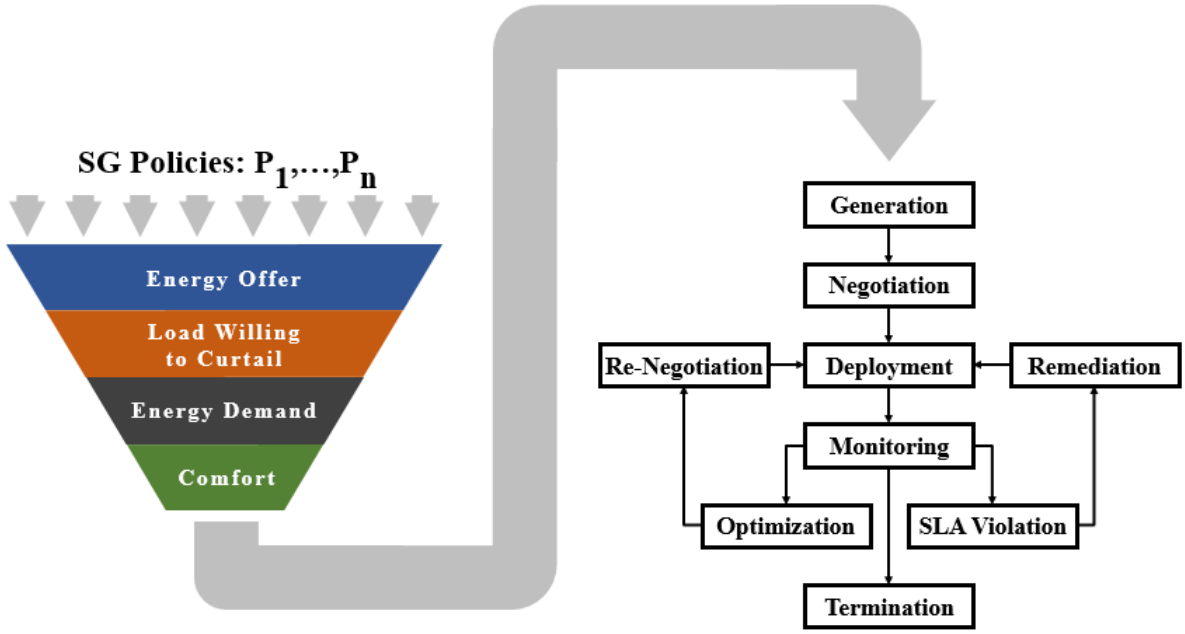

Fig 2.2: SG policies and the SLA framework.

It is important to mention that "exact" optimization methods such as linear and integer programming are not appropriate to solve current SG optimization problems, since most of them have multiple conflicting objectives, the dimension of the decision space is high, and the set of solutions within the decision space is not necessarily convex.

2.3 Evolutionary Algorithms

During the last two decades, new EAs have been developed and others have been modified for improvements. EAs are generic population-based metaheuristic optimization algorithms based on biological phenomenon such as mutation and selection. Although EAs do not

guarantee exact solutions, they are useful when approximating the true PF of MOO problems. For instance, in [1] the energy flow of a system is optimized implementing a combination of load forecasting genetic algorithm and Adaptive Neuro-Fuzzy Inference Systems to satisfy the energy demand. The results show benefits in terms of energy consumption, operation cost, and generated CO_2 emissions. In [2], a differential EA is applied on a mixed-integer optimization problem for optimal energy generation while applying DR through energy demand curtailment. In other work [3], an EA is developed to solve a MOO problem for load scheduling while minimizing cost and maximizing utility. Although these studies aim at problems with two or more conflicting objectives, they do not provide day-ahead load forecasting and fall short to evaluate the implemented EAs in terms of performance metrics like the hypervolume indicator.

2.3.1 Multi-Objective Evolutionary Algorithm Based on Decomposition

The Multi-Objective Evolutionary Algorithm Based on Decomposition (MOEA/D) [6] decomposes MOO problems into subproblems and optimizes them concurrently. The decomposition can be done using the weighted sum approach where a convex combination of the objectives in an optimization problem are considered. Let $(\lambda_1, ..., \lambda_m)^T$ be a weight vector where $\lambda_i \geq 0 \; \forall \; i \in [1, m]$ and $\sum_{i=1}^{m} \lambda_i = 1$. Thus, the solution of the optimization problem:

$$\begin{aligned} \max \quad & \left\{ g(x|\lambda) = \sum_{i=1}^{m} \lambda_i f_i(x) \right\} \\ \text{s.t.} \quad & x \in \Omega, \end{aligned} \tag{2.2}$$

is Pareto optimal to the MOO problem:

$$\begin{aligned} \max \quad & \left\{ F(x) = (f_1(x), ..., f_m(x))^T \right\} \\ \text{s.t.} \quad & x \in \Omega, \end{aligned} \tag{2.3}$$

where f_i is the *i-th* objective of the MOO problem, and Ω is the decision space. In this case, as MOEA/D solves N scalar problems at the same time. However, not all the Pareto optimal solutions can be obtained by this approach in cases where the PF is not concave. A specific implementation of MOEA/D in [7] obtains comparable results to Non-dominated Sorting Genetic Algorithm II (NSGA-II) when optimizing cost and utility for optimal

energy consumption.

2.3.2 Indicator-Based Evolutionary Algorithm

The Indicator-Based Evolutionary Algorithm (IBEA) [8] adapts the fitness of its population according to indicators like the hypervolume. One way of doing this is summing up the indicator values of the population members while giving more importance to the dominating population members over the dominated ones:

$$F(p_1) = \sum_{p_2 \in P\{p_1\}} -e^{-I_H(p_1,p_2)/\kappa}, \tag{2.4}$$

where p_1 and p_2 are decision vectors of the population P, I_H is the hypervolume indicator which is to be maximized, $F(p_1)$ is a measure for the loss in quality if p_1 is removed from the population, and κ is a scaling factor that depends on I_H and the optimization problem. Since the hypervolume is a Pareto dominance preserving algorithm ($I_H(p_1, p_2) < I_H(p_2, p_1)$ if $p_1 \succ p_2$), the importance of small values of I_H contribute much more to the overall fitness than the large ones. In other words, population members with low values of $F(p_1)$ are ranked higher than the ones with high values of $F(p_1)$. This makes the IBEA a flexible EA as any Pareto dominance compliant indicator can be implemented for its fitness assignment scheme. However, it can be computationally complex depending on the indicator.

2.3.3 Epsilon Domination Based Multi-Objective Evolutionary Algorithm

The Epsilon Domination Based Multi-Objective Evolutionary Algorithm (ϵ-MOEA) [9] uses two separate co-evolving populations, an EA population and an archive population. See Figure 2.3. Two solutions, one from each populations, are selected and from them an offspring is created. The offspring is added to the EA and archive populations using a tournament procedure and an archive acceptance procedure, respectively. This is repeated until a maximum number of iterations is reached. The tournament procedure consists of several tournaments among some of the solutions in EA that are chosen at randomly from its population. The winner of each tournament is selected and put back into the population. The archive acceptance procedure consists of not allowing two solutions with a difference of less than ϵ in the archive. In this way, good diversity is maintained in EA and convergence is achieved in an efficient manner.

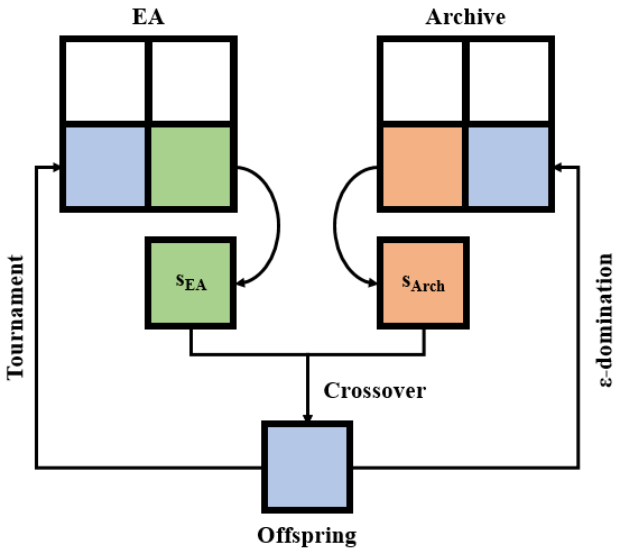

Fig 2.3: ϵ-MOEA procedure.

2.3.4 Covariance Matrix Adaptation Evolution Strategy

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [10] implements an evolution strategy to sample solutions according to a normal distribution in $\mathbb{R}^n$. A recombination technique is executed on the population members of the current iteration to obtain an new mean for the next iteration. The variables of the optimization problem are stored in a covariance matrix that is updated according to a covariance matrix adaptation method that guides the step at which the algorithm converges to the optimum. This method follows the maximum-likelihood principle in which the mean of the distribution is updated so that the likelihood of the selected solutions for the next iteration is maximized. This is done repeatedly until a maximum number of iterations is reached. Although this EA is good for ill-conditioned MOO problems, it presents issues handling MOO problems with a larger number of decision variables.

2.3.5 Third Evolution Step of Generalized Differential Evolution

The Third Evolution Step of Generalized Differential Evolution (GDE3) [11] is an extension of DE that improves its predecessors, the GDE and GDE2. Similar to the NSGA-II, in GDE3, solutions are sorted based on non-dominance and crowdedness while considering the constraints of the optimization problem and the selection based on crowding distance.

Although this provides GDE3 with a measure of the density of solutions in a PF, the sorting scheme presents issues when solutions share the same fitness.

2.3.6 Speed-constrained Multi-Objective Particle Swarm Optimization

New approaches extend the PSO to handle MOOPs but they make the particles, go off their lower and upper bounds when their velocity becomes too high [12]. Speed-constrained Multi-objective Particle Swarm Optimization (SMPSO) implements a constriction coefficient to prevent this from happening:

$$\chi = \frac{2}{2 - \varphi - \sqrt{\varphi^2 - 4\varphi}}, \tag{2.5}$$

where

$$\varphi = \begin{cases} C_1 + C_2 & \text{if } C_1 + C_2 > 4 \\ 1 & \text{if } C_1 + C_2 \leq 4 \end{cases}, \tag{2.6}$$

being C_1 and C_2, the values that control the effect of the *i-th* best particle and the global best particle over the *i-th* velocity. In addition, the accumulated velocity is bounded by a delta that is calculated considering the velocity.

2.3.7 Non-dominated Sorting Genetic Algorithm II

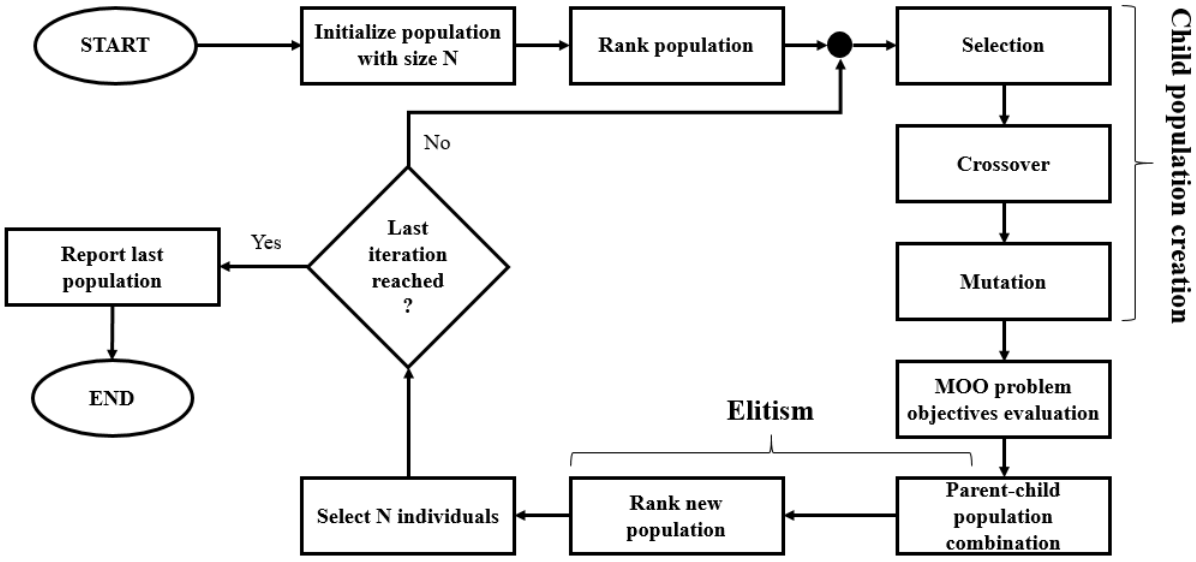

Fig 2.4: NSGA-II top level flowchart.

The NSGA-II [13] is based on the non-dominated sorting approach. It improves its predecessor, the NSGA, with elitism storing all the non-dominated solutions of the *i-th* iteration in an archive for later selection in the subsequent iterations. The algorithm first create a child population from an initial randomly generated population through selection, crossover, and mutation methods. See Figure 2.4. Then, the objectives of the optimization problem are evaluated using the children population. The parent population is combined with the child population and the resulting population is ranked. Finally, The non-dominated solutions go to the archive and N of them are selected. If the maximum number of iterations has been reached, the last N individuals are the solutions of the optimization problem.

2.3.8 Strength Pareto Evolutionary Algorithm 2

The Strength Pareto Evolutionary Algorithm 2 (SPEA2) [14] improves its previous version, the SPEA, with an improved fitness assignment scheme. This scheme avoids the situation where solutions dominated by the same archive members have similar fitness values by

assigning a strength value to each member in the archive:

$$R(i) = \sum_{j \in P_t \bigcup \bar{P}_t, i \succ j} S(j), \tag{2.7}$$

where

$$S(i) = |\{j | j \in P_t \bigcup \bar{P}_t \wedge i \succ j\}|, \tag{2.8}$$

being $R(i)$, the raw fitness of the *i-th* solution in $j \in P_t \bigcup \bar{P}_t, i \succ j$; $S(i)$, the strength of the *i-th* solution; P_t, the population of solutions of the current iteration; and $\bar{P}_t$, the archive population. Then, the raw fitness is added to the density $D(i)$ which is calculated through the *k-th* nearest neighbor method:

$$F(i) = R(i) + D(i), \tag{2.9}$$

where

$$D(i) = \frac{1}{\sigma_i^k + 2}, \tag{2.10}$$

being $F(i)$, the fitness; $D(i)$, the density; and σ_i^k, the distance of the *i-th* solution.

2.4 Evolutionary Algorithm Performance Metrics

As EAs prove useful for many applications in science and engineering, establishing a set of comparison methods is necessary. These methods are called performance metrics and they measure three main characteristics of the solution sets obtained by EAs:

- **Accuracy:** proximity to the true PF,
- **Diversity:** distribution and dispersion, and
- **Cardinality:** number of solutions.

In [15] various performance metrics are compared in terms of their number of citations. The selection of performance metrics for this research is based on this and their classification. Table 2.1 shows the performance implemented in this research along with their number of citations and classification. They are described in detail in Sections 2.4.1 to 2.4.6.

Table 2.1: The performance metrics by number of citations and classification.

Number of Citations	**Performance Metric**	**Classification**
91	Hypervolume Indicator	Accuracy Diversity
26	Generational Distance	Accuracy
23	Epsilon Indicator	Accuracy Diversity Cardinality
17	Inverted Generational Distance	Accuracy Diversity
6	Spacing	Diversity

2.4.1 Hypervolume Indicator

The hypervolume indicator $I_H(P)$ [16] is defined as the Lebesgue measure $\lambda(S)$ of a set S whose elements consist of all the solutions weakly dominated by a solution p but not weakly dominated by a reference point $r = (r_1, \ldots, r_i) \in \mathbb{R}^n$. More precisely,

$$I_H(P) = \lambda\left(\bigcup_{p \in P} [f_1(p), r_1] \times \cdots \times [f_k(p), r_i]\right), \tag{2.11}$$

where $P \subseteq \Omega$, being Ω, the decision space and $[f_1(p), r_1] \times \cdots \times [f_k(p), r_i]$, an i-dimensional hypercuboid. The hypervolume indicator is Pareto-dominance compliant. For example, when a PF $P^a \subseteq \Omega$ is strictly better than another PF $P^b \subseteq \Omega$, the hypervolume of P^a is also strictly better than the one of P^b ($I_H(P^a) > I_H(P^b)$). Figure 2.5 shows the graphic representation of this metric for two objectives. This can be interpreted as the area obtained by merging the areas of the rectangles containing the solutions and the reference point. For more than two objectives, I_H value represents the volume enclosed within the resulted attained surfaces.

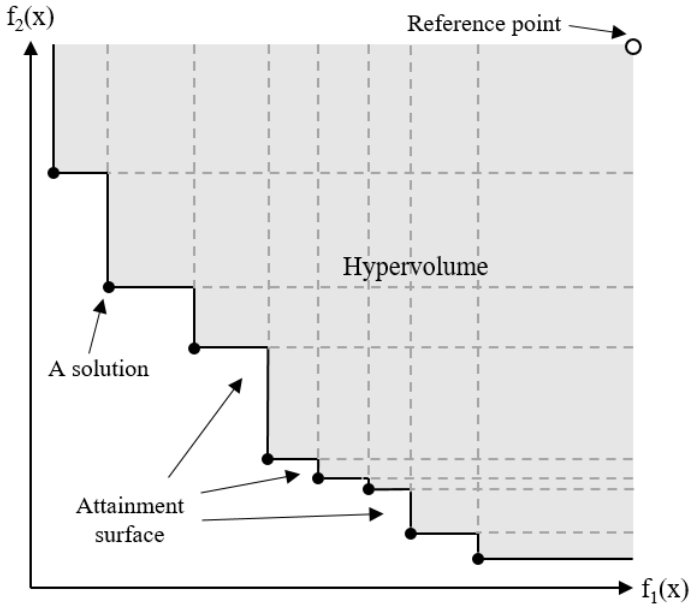

Fig 2.5: Graphic representation of the Hypervolume for two objective functions.

Although great effort have been put in decreasing the time complexity of the Hypervolume, its calculation is still done in exponential time [17]. Thus, EAs like the IBEA are affected negatively as they take more time than other EAs, based on other schemes, to converge.

2.4.2 Generational Distance

The Generational Distance (GD) $I_{GD}(RPF, P)$ [16] is defined as the average distance between a reference PF $RPF \subseteq \Omega$ and a PF $P \subseteq \Omega$. Specifically,

$$I_{GD}(RPF, P) = \frac{\sqrt{\sum_{i=1}^{|P|} d_i^2}}{|P|}, \tag{2.12}$$

where d_i^2 is the Euclidean distance between each solution in P and the nearest solution in RPF. In other words, for each solution in the PF obtained by an EA, the distance of the closest solution in the reference PF is calculated. See Figure 2.6. This metric is not Pareto-dominance compliant ($I_{GD}(P^a) < I_{GD}(P^b)$).

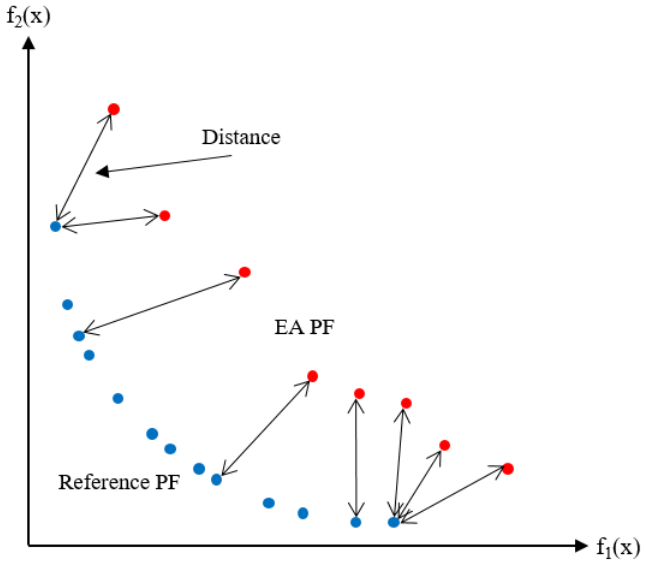

Fig 2.6: Graphic representation of the GD for two objective functions.

In terms of time complexity, the calculation of the GD can be done in linear time [17]. This makes the GD one of the fastest metrics to calculate while giving initial insights about the solutions set obtained by EAs.

2.4.3 Epsilon Indicator

The Epsilon Indicator (ϵ-indicator) $I_\epsilon(P, RPF)$ [18] is defined as the minimum factor ϵ such that for any solution in RPF there is at least one solution in P that is not worse by a factor of ϵ in all the objectives. Similarly,

$$I_\epsilon(P, RPF) = \max_{rpf \in RPF} \min_{p \in P} \min_{1 \leq i \leq n} \frac{p_i}{rpf_i}, \tag{2.13}$$

where $p_i \in P$, $rpf_i \in RPF \subseteq \Omega$, and n is the number of objectives. The ϵ-indicator is not Pareto-dominance compliant ($I_\epsilon(P^a) < I_\epsilon(P^b)$) as shown in Figure 2.7 where the ϵ-indicator gives the factor by which an EA PF is worse than another in terms of the objectives. The time complexity of this metric is linear [17].

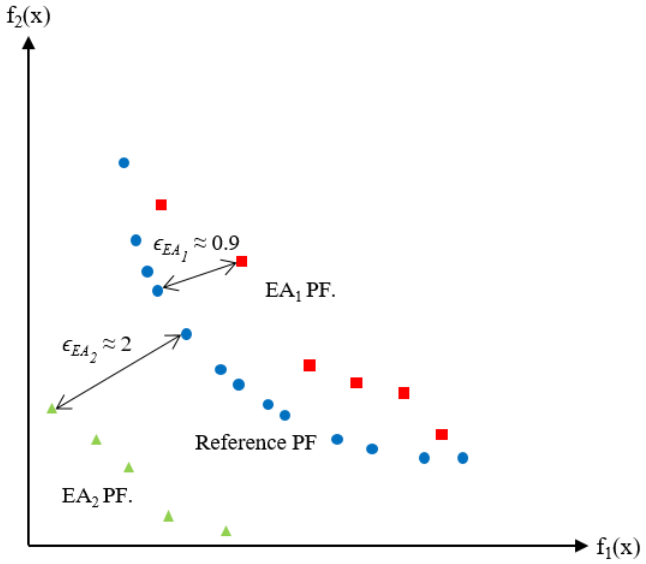

Fig 2.7: Graphic representation of the ϵ-indicator for two objective functions.

2.4.4 Inverted Generational Distance

The Inverted Generational Distance (IGD) $I_{IGD}(RPF, P)$ [4] is defined as the average distance between each reference solution $rpf_i \in RPF \subseteq \Omega$ to its nearest solution $p_j \in P \subseteq \Omega$.

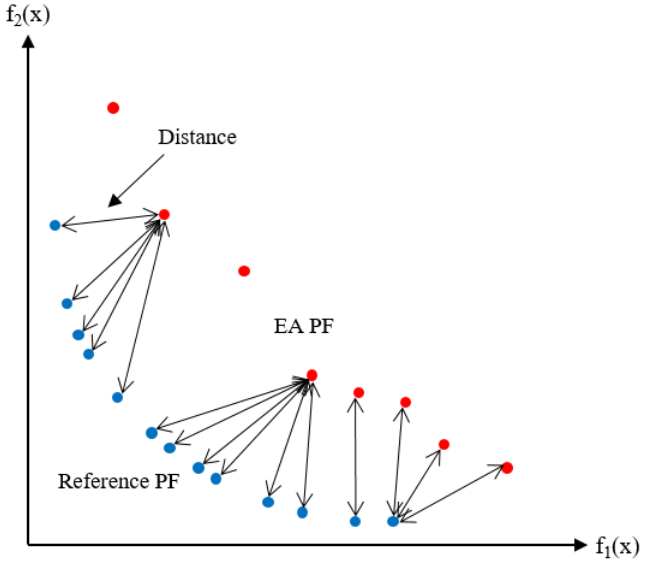

Fig 2.8: Graphic representation of the IGD for two objective functions.

Similarly,

$$I_{IGD}(RPF, P) = \frac{\sqrt{\sum_{i=1}^{|RPF|} d_i^2}}{|RPF|}, \tag{2.14}$$

where d_i is the smallest Euclidean distance between each solution in RPF and the nearest solution in P. See Figure 2.8. This metric is not Pareto-dominance compliant ($I_{IGD}(P^a) < I_{IGD}(P^b)$). The time complexity of this metric is linear [17].

2.4.5 Spacing

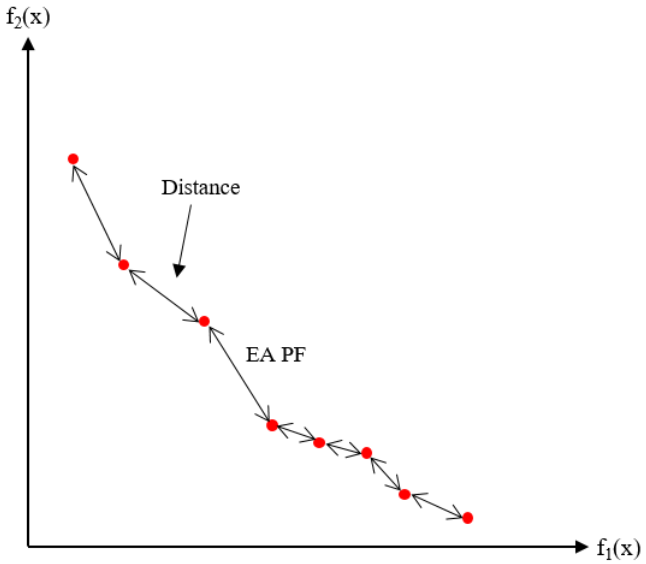

Fig 2.9: Graphic representation of the spacing for two objective functions.

The spacing (I_S) [16] is defined as the distance variance of neighboring solutions in $P \subseteq \Omega$. More precisely,

$$I_S(P) = \sqrt{\frac{1}{|P| - 1} \sum_{i=1}^{|P|} (d_i - \bar{d})^2}, \tag{2.15}$$

where $d_i = \min_{p_i \neq p_j} ||F(p_i) - F(p_j)||$, $p_i, p_j \in P \subseteq \Omega$, and $\bar{d}$ is the mean of all d_i. This metric is not Pareto-dominance compliant ($I_S(P^a) > I_S(P^b)$) and its time complexity is quadratic [17].

2.4.6 Running Time

Additionally to the EA metrics described above, the running time I_{RT} is implemented to measure the performance of EAs in terms of how long it take them to optimize MOO problems:

$$I_{RT} = t_f - t_i, \tag{2.16}$$

where t_i is the initial time and t_f is the final time, both in seconds. This metric is not Pareto-dominance compliant ($I_{RT}(EA^a) < I_{RT}(EA^b)$) and its calculation is constant. With the optimization framework components defined, the optimization platform can be developed and energy demand side optimizations can be carried out on SG models.

2.5 Conclusion

In this Chapter the components of a Multi Objective Optimization framework for Smart Grids modeling and simulation have been discussed. Several Evolutionary Algorithms and performance evaluation metrics have been presented. The application of the proposed framework on energy demand side optimization is demonstrated in the next chapter.

Chapter 3

Energy Demand Side Optimization

Leveraging on the knowledge acquired in the OASIS project with regards to DR strategies, a scenario of generation and consume of energy is modeled and simulated using the optimization platform presented in Chapter 2. The performance of these EAs is evaluated using the metrics described in that chapter too. To my knowledge this is the first time that a comprehensive evaluation of the performance of these algorithms in the context of DR is presented.

3.1 Model

Using the values associated with the policies that include the maximum generation capacity of the producer and the loads of the consumers, the cost can be minimized and utility can be maximized with the goal of performing day-ahead load forecasting. The cost is the amount of money paid per unit of energy consumed. It can be defined with the following function [3]:

$$\begin{aligned} Cost &= a_c E_t^2 + b_c E_t + c_c, \\ E_t &= \sum_{c=1}^{C} dv_c^t, \end{aligned} \tag{3.1}$$

where a_c, b_c, and c_c are constant coefficients, E_t is the total energy used by the consumers up to a time interval $t \in [1, T]$, and $dv_c^t \in DV$. DV is a $C \times T$ matrix where C is the number of consumers and T is the number of time intervals. Each row represents the loads of the consumers from 1 to T while each column represents the loads of all the consumers in a time interval t. The values in DV that yield to non-dominated solutions are used to obtain the next day loads for the consumers assuming a steady energy generation and

that the consumption profiles of the consumers do not change drastically within two days. The utility is the comfort associated with the consumption of energy. It can be defined with the following concave increasing function [3]:

$$Utility = log_{10}(E_t), \tag{3.2}$$

The values of a_c, b_c, c_c, C, and T are defined in Section 3.3.

3.1.1 Problem Formulation

With the objective functions of cost and utility defined, the MOOP is as follows:

$$\begin{aligned} \min \quad & \left\{ \sum_{t=1}^{T} \Big(Cost\,(E_t) \Big), - \sum_{t=1}^{T} \Big(Utility\,(E_t) \Big) \right\} \\ \text{s.t.} \quad & ds_c^t - \tau_l \leq dv_c^t \leq ds_c^t + \tau_u \forall c \in [1, C] \wedge \forall t \in [1, T], \\ & 0 \leq E_t \leq E_{max} \forall t \in [1, T], \end{aligned} \tag{3.3}$$

where $ds_c^t \in DS$, and E_{max} is the total energy generation capacity of the producer. DS is a matrix with equal dimensions to DV. Its values are obtained from a hardware device, called the dSpace, that models the energy use of consumers connected to a SG powered by photovoltaic cells (see Figure 3.1).

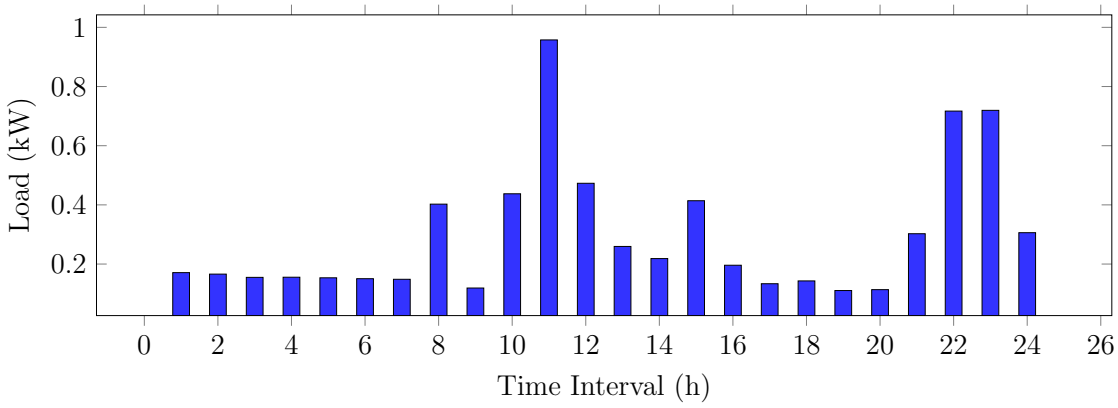

Fig 3.1: Example of a consumer load in 24 time intervals of 1 hour obtained from the dSpace.

The constants τ_l and τ_u set the bounds for the energy that consumers are willing to curtail and the amount of energy that they may use when exceeding their policy values. This provides flexibility when the total generation capacity of the producer is reached or

when individual consumer loads are greater than expected. The first constraint sets lower and upper bounds of the decision space. The second guarantees the security of the energy generation systems of the producer by not allowing the total energy consumed, up to a time interval t, to exceed the total energy generation capacity of the producer. The values of τ_l and τ_u are defined in Section 3.3.

3.2 *Platypus* Evolutionary Computing Framework

Before discussing the implementation of the DR case study, let me introduce *Platypus*. It is an open source framework for evolutionary computing ported to *Python* from the Multi-Objective Evolutionary Algorithm (MOEA) Framework in Java [19]. It provides the tools necessary to design, develop, execute, and test EAs like the NSGA-II and Pareto Archived Evolution Strategy Algorithm (PAES). It also supports the execution of EAs and performance metrics in parallel. *Platypus* works well on *Python* versions 2.7 and 3.4, and in both Windows 10 and Linux 16.04 LTS. The following is a MOO problem example and what *Platypus* can do with it.

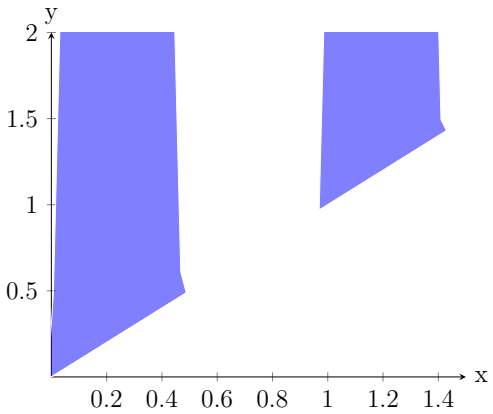

Fig 3.2: Decision space of the non-convex optimization problem example.

This a non-convex problem that is defined with one decision variable, two objective functions, and four constraints:

$$\begin{aligned} \min \quad & \{y = 10sin(120\pi x), y = x\} \\ \text{s.t.} \quad & 0 \leq x \leq 1.5, 0 \leq y \leq 2, \\ & x \leq y, y \leq 10sin(120\pi x) \end{aligned} \tag{3.4}$$

Figure 3.2 shows the graphic representation of this optimization problem. The shaded area in blue is the decision space. Having a look at this optimization problem, it is easy to verify that the optimum is at $x = 0, y = 0$. The *Python* program that defines this problem and solves it is shown in Listing 3.1. In the first two lines, the necessary modules classes and variables are imported. In line 4, the *example* class is defined as a subclass of *Platypus' Problem* class. When instantiating the *example* class, the *super()* method must be called. This is done in line 6 by passing the *example* class and the number of decision variables, objectives, and constraints as arguments to *super()*. The rest of the constraints are defined in line 7. There, the variable types are also set as continuous. Then, the constraint operators are defined in line 8.

In line 10, the *evaluate* function is overwritten. The objectives and constraints are evaluated according to the position of the population members in the decision space as the EA progresses in its iterations. In line 16, the problem is instantiated and passed to the algorithm that will optimize it as an argument. In this case, the NSGA-II is implemented. The EA solves the problem with a total number of 10000 iterations in line 7. Finally, the x and y values of each solution is printed to the terminal. All the solutions yielded $x = 3.40 \times 10^{-17}$, $y = 1.28 \times 10^{-13}$, which is close to the exact optimum.

```
1  from platypus import NSGAII, Problem, Real
2  from math import sin, pi
3
4  class example(Problem):
5      def __init__(self):
6          super(example, self).__init__(2, 2, 2)
7          self.types[:] = [Real(0, 1.5), Real(0, 2)]
8          self.constraints[:] = "<=0"
9
10     def evaluate(self, solution):
11         x = solution.variables[0]
12         y = solution.variables[1]
13         solution.objectives[:] = [10 * sin(120 * pi * x), x]
14         solution.constraints[:] = [x - y, y - 10 * sin(120 * pi * x)]
15
16 algorithm = NSGAII(example())
17 algorithm.run(10000)
18
19 print([(result.variables[0], result.objectives[0]) for result in algorithm.
       result])
```

Listing 3.1: Simple example source code.

Platypus also provides the *experimenter* module for testing multiple EAs on multiple MOO problems with different parameters and comparing performance metrics. Besides the *Problem* class, it also provides a set of abstract data types for implementing custom EAs and performance metrics. Finally, parallelization is supported through *Platypus' ProcessPoolEvaluator* class. This is very convenient as this research benefits from running the experiments concurrently on large-scale systems.

3.3 Scenario Implementation

The Cost-Utility (CU) problem is defined as a class that inherits from the *Problem* class. Any instance of the *CU* class can be instantiated with the parameters listed in Table 3.1. The values of a_c, b_c, c_c, and E_{max} are based on [3], while the values of τ_l and τ_u are based on the standard deviation of the values in DS.

Table 3.1: Values of the relevant variables in the CU problem.

$\mathbf{a_c}$	$\mathbf{b_c}$	$\mathbf{c_c}$	**C**	**T**	$\mathbf{E_{max}}$	$\tau_\mathbf{l}$, $\tau_\mathbf{u}$
0.2	0.3	0.05	200	24 hrs	3200 kW	0.08 kW

The performance metrics are calculated using a reference set of non-dominated solutions generated with *Platypus*, since the true solution set of the CU problem is unknown. This set is created using an epsilon archive, similar to the one used in the ϵ-MOEA. In order to use this archive for measuring the performance of different EAs, the archive object is serialized as a stream and stored in a file within the source code of the project. When needed, the content of the file is deserialized back to an epsilon archive object and passed as an argument to the performance metric objects to calculate their values.

The algorithm for day-ahead load forecasting optimizing the formulated problem is shown in Algorithm 1. In line 1 the algorithm that will be tasked with the optimization, the sets of metrics to measure its performance, the optimization problem, and the number of rounds are set. A counter for the current round is set in line 2. In line 3, a loop iterates over each round. In lines 4 and 6, the initial and final times are recorder for calculating the running time of the EA optimizing the CU problem. Lines 5 and 7 are executed concurrently through the *ProcessPoolEvaluator* and the results are saved accordingly. In lines 8 and 9, the running time is calculated and added to the result of the other metrics respectively. The algorithm and metric results are added to a log variable in line 10 and the values stored there are saved to a file in line 12. Finally, the counter is incremented by

one in line 12. In case the maximum number of rounds is not reached, the instructions from line 4 to 12 are repeated.

Algorithm 1 Load forecasting optimizing the CU problem.

```
1:  procedure LFCU(algorithm, metrics, CU, rounds)
2:      round ← 1
3:      while round ≤ rounds do
4:          ti ← start time
5:          algo_res ← run algorithm on CU
6:          tf ← end time
7:          met_res ← calculate metrics using algo_res
8:          running_time ← tf - ti
9:          met_res ← met_res ∪ {running_time}
10:         log ← algo_res and met_res
11:         save log to a file
12:         round ← round + 1
```

The EAs were put to optimize the CU problem a total of 30 rounds for obtaining statistical information about their performance with a total of 1000 function evaluations and a population size of 100. The experiments were done on a Linux 14.04 LTS compute node running on Chameleon Cloud [20], a configurable experimental Infrastructure as a Service (IaaS) environment for large-scale cloud research powered by OpenStack. In this case, the node included 48 cores Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30 GHz and 128 GB of RAM. An image of this node is available at Chameleon Cloud with the environment requirements installed for future research in this area (See Appendix C).

3.4 Experimental Results

Table 3.2 shows that the EAs achieved the best mean values for the performance metrics. Although MOEA/D presents the best hypervolume, it has the lowest spacing among the EAs (see Figure 3.3). This suggests that its non-dominated solutions are too close to each other in comparison to other EAs with higher spacing. Solution sets with low spacing include solutions that are too similar to each other. This could be a disadvantage as solution sets like this one would present decision-makers with few options for SG SLA deployment.

Table 3.2: Best mean values for each performance metric by EA.

Performance Metric	Best EA	Mean Value
Hypervolume	MOEA/D	18.886138
GD	ϵ-MOEA	1.233662
ϵ-indicator	SMPSO	7.849621
IGD	IBEA	18.670577
Spacing	IBEA	2.650885
Running Time	GDE3	325.628541 s

ϵ-MOEA achieved the best GD and the other EAs performed similarly among themselves. This means that the solutions sets obtained by this EA are closer, in average, to the reference solution set than the solution sets obtained by the other EAs. However, the solutions sets obtained by ϵ-MOEA are not as spread as the ones obtained by IBEA and NSGA-II, the EAs with the best spacing.

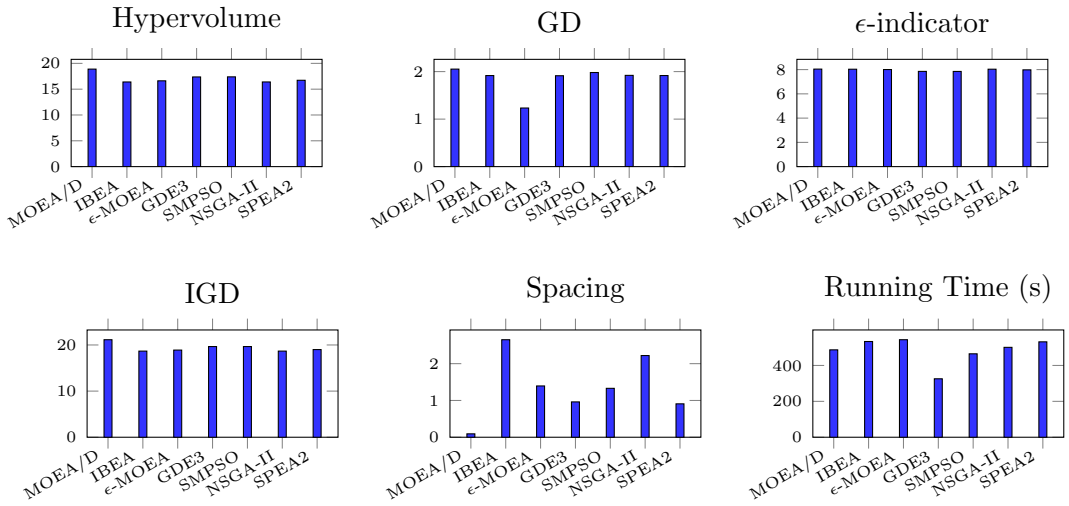

Fig 3.3: Median values for the performance metrics obtained in the experiments.

The SMPSO obtained the best value for the ϵ-indicator but the median values in Figure 3.3 show that the other EAs obtained similar values. This suggests that, even when the EAs obtained solution sets on top or under the reference solution set, they were similarly close to it. This is supported by the hypervolume values obtained by all the EAs between 15 and 20. The lowest running time was achieved by GDE3 with 3.26×10^2s. The other EAs achieved acceptable running times between 464 s and 544 s due to the use of multiple processors simultaneously for the experiments in addition to the good hardware specifications of the compute node. The rest of the median values for each performance

metric and EA are shown in Figure 3.3.

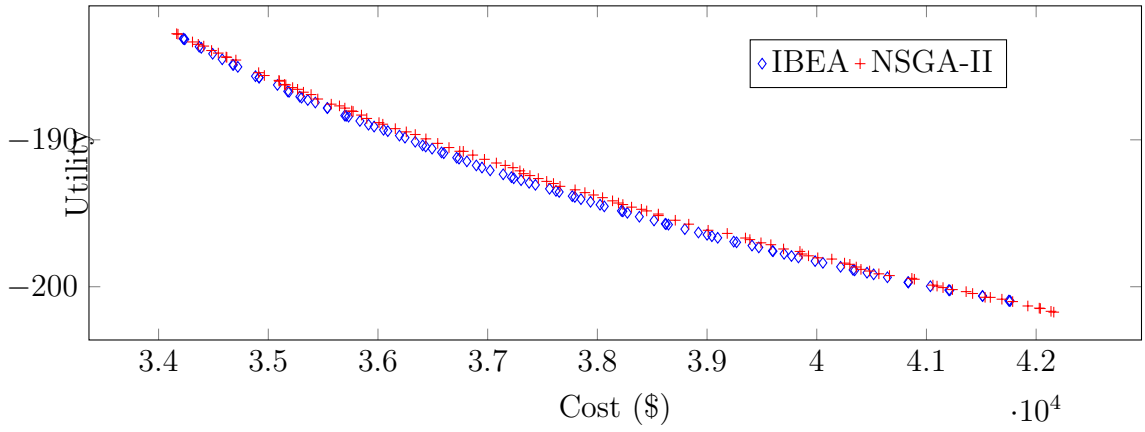

Fig 3.4: IBEA and NSGA-II solution sets obtained in their first run.

Figure 3.4 shows two of the most diverse solution sets obtained in the second run for IBEA and NSGA-II. One of the disadvantages of solution sets like these is that decision makers could get overwhelmed by the amount of different solutions that they present. By inspection, from Figure 3.4, the cost is between $\$3.41 \times 10^4$ and $\$4.22 \times 10^4$. Solving the cost function for E_t yields 412.17 kW and 458.60 kW, respectively. These values are less than E_{max}.

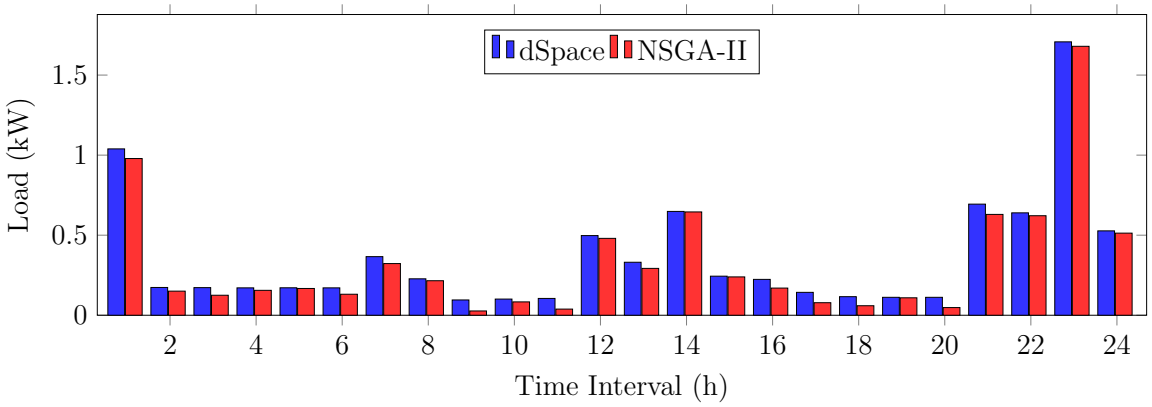

Fig 3.5: Consumer load obtained from the dSpace and the corresponding load obtained running NSGA-II on the CU problem.

Figure 3.5 shows a comparison between the loads obtained from the dSpace and the loads forecasted using NSGA-II. This is due to the constraint where τ_l and τ_u are present. Since the cost is minimized in the CU problem, the forecasted loads are less than the ones obtained from the dSpace in most of the time intervals. Similar results were obtained

by the other EAs. No results were obtained for CMA-ES as it did not converge to any solutions after running for 3.46×10^5 s. This was a consequence of the processing and management of large co-variance matrices associated with the total number of variables ($200 \times 24 = 4800$).

3.5 Conclusion

To summarize, this chapter presents a set of performance metrics to evaluate a set of EAs that optimized a MOO problem on a SG scenario of 200 houses and 24 time intervals of one hour including objectives for cost and utility in addition to constraints related to the maximum generation capacity of the producer and the loads of consumers. Day ahead load forecasting was achieved using these policies. Mean performance metric values obtained in the experiments give insight about the solution sets of the EAs. In most cases, the loads forecasted by the EAs are less or equal the ones obtained for the dSpace. Still, they were within $ds_c^t - \tau_l$ and $ds_c^t + \tau_u$, which enforce the DR component of the CU problem through load curtailment.

The EAs also obtained values for E_t within 0 and E_{max} which ensure the stability of the energy generation systems of the producer while supplying the necessary amount of energy to the consumers. Although, the values of E_t by the EAs are way smaller than E_{max}, the loads of consumers could be used to project the necessary power needed at certain time during a SG deployment so that the wastage of energy is minimized. This can be achieved by establishing a communication line between the consumers and the producer. In that case the consumers would communicate how much energy they would consume at certain times and the producer would supply the demand without over-generating energy. Thus, the deployment of the energy services in SGs could be done with no violations. Schemes involving this strategy are described the next chapter.

Chapter 4

Transactive Energy Optimization

In the previous chapter, a model considering the cost and utility of consumers was optimized for day-ahead load forecasting. Although the benefits of the optimization platform are evident, the results also show that the supply is way over the demand. In this chapter, a TE control mechanism through MASs is proposed to solve this. The theoretical and technical aspects of this solution are described next.

4.1 Transactive Energy Multi-Agent Systems

The implementation of SG technologies has led to significant improvements in efficiency related to the generation, distribution, and consumption of energy [21]. Furthermore, requirements related to the scalability of energy services that comply with producer and consumer policies, have increased the focus from the economic aspects of the SG to the control applications that guarantee its reliability. This is called TE.

In TE approaches, economic and control techniques are combined to improve the SG reliability and efficiency [21] by coordinating the deployment and monitoring of distributed energy service resources. For example, smart meters monitoring the consumption of energy in a house could send this information to its local renewable energy resource or the producers. At the energy production side, optimizations could be done to supply the demand with a minimum amount of energy without compromising the energy systems. This requires a mechanism to, not only establish communication between the energy consumption and production sides, but also to enforce DR strategies between them (See Figure 4.1).

The proposed TE approach leverages functionalities from the optimization platform presented in Chapter 3 to perform optimizations at different SG infrastructure levels to

control the behavior of its entities. The entities can be Photovoltaics (PVs) or houses. Agents could be implemented to do this allowing energy producers and consumers to set the policies they want the optimization algorithms to consider while exchanging information with other entities. The information can flow between entities of the same side or between entities of different sides to choose the appropriate energy service deployment. In this work, this is investigated.

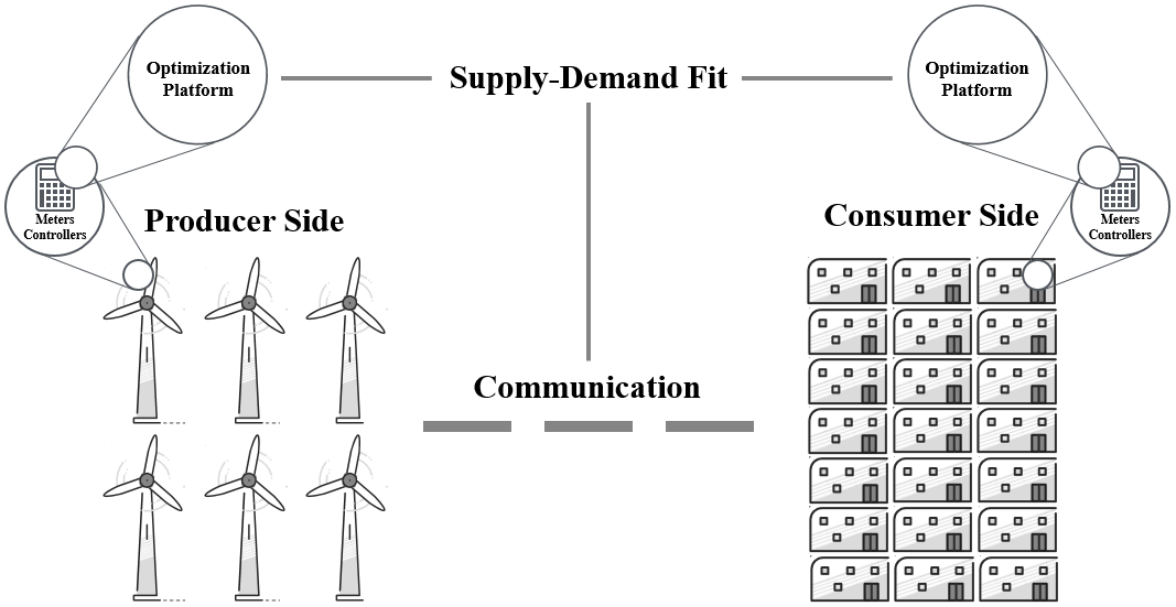

Fig 4.1: Representation of the proposed TE approach.

The DCG is proposed as the framework, to investigate the behavior of entities connected to the SG under different TE distribution and communication strategies. From it four combinations of distribution and communication scenarios derived. They include the distributed generation and one-way communication; centralized generation and one-way communication; distributed generation and two-way communication; and centralized generation and one-way communication (See Figure 4.2). In terms of distribution, the producer entities can be dispersed or centralized in the SG topology while the communication can be one-way or two way. One-way means the communication of information among the entities of the same side and two-way refers to communication of information between entities of different sides although they may still communicate with entities of their side. The models for the scenarios are described in the next sections.

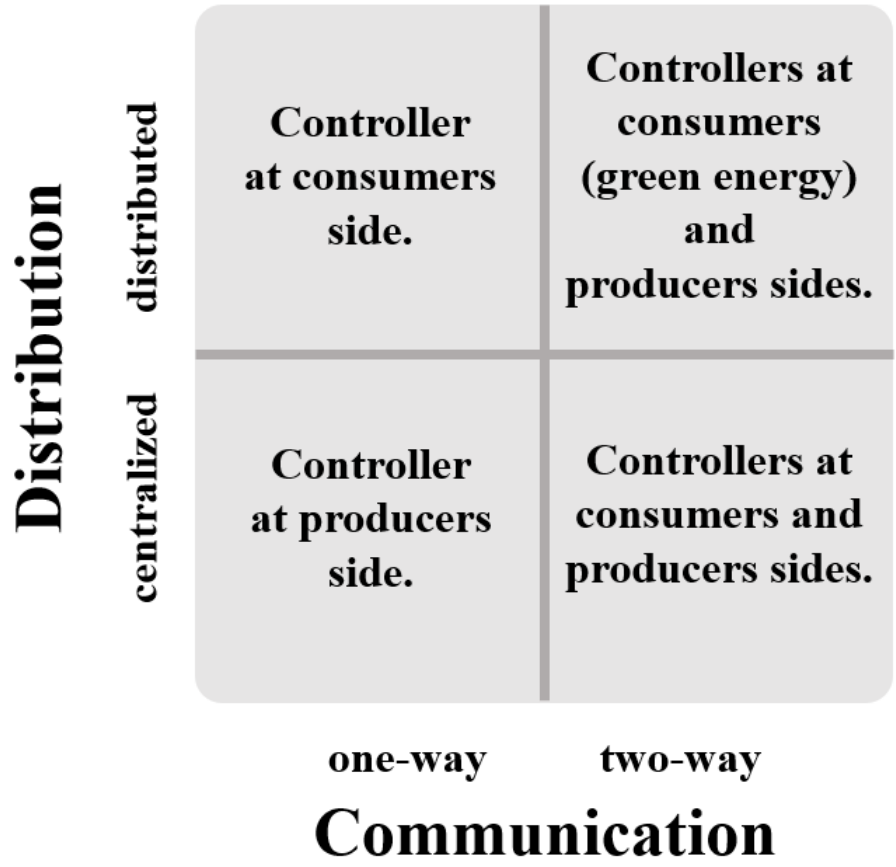

Fig 4.2: The Distribution-Communication Grid.

4.2 The Distribution-Communication Grid Models

Each scenario implements a model or combination of models that enforce DR on some or all the entities present in a SG.

4.2.1 Distributed Generation and One-way Communication

The model of this scenario is similar to the one presented in Section 3.1, but the value of E_{max} is set to the number of consumers multiplied by 47.43 kW, the average house energy consumed per day obtained from the data sets used for the experiments of these scenarios. In this model the controllers reside at the consumers side and exchange information within themselves. The producer entities are dispersed along the topology of the scenario based on this model, one per house (See Figure 4.3).

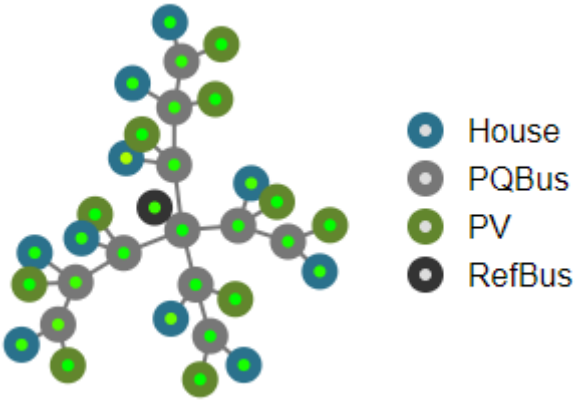

Fig 4.3: An example of a distributed SG topology.

In this model the number of consumption profiles grows as the simulation time increases. This means that the optimization in the scenario based on this model is dynamic because the matrix increases in size by one column when the time interval increases by one unit, in contrast to the optimization performed in Chapter 3 where all the values in the matrix were determined from the beginning.

4.2.2 Centralized Generation and One-way Communication

In [22], a model describes the cost of generating energy with renewable energy resources including PVs and batteries while considering their availability. The cost of generating or supplying energy with PVs or batteries, respectively, can be formulated as:

$$C_{PVBatt} = \frac{(I_{PV} + OM_{PV}) + (I_{Batt} + OM_{Batt})}{N}, \tag{4.1}$$

where

$$\begin{aligned} I_{PV} &= \lambda_{PV} A_{PV}, \\ OM_{PV} &= OM_{PV} A_{PV} \sum_{i=1}^{N} \left(\frac{1+\nu}{1+\gamma}\right)^{i}, \\ I_{Batt} &= \lambda_{Batt} P_{CapBatt}, \\ OM_{Batt} &= OM_{Batt} P_{CapBatt} \sum_{i=1}^{N} \left(\frac{1+\nu}{1+\beta}\right)^{i}, \end{aligned} \tag{4.2}$$

being I_{PV}, OM_{PV}, I_{Batt} and OM_{Batt}, the initial and Operation & Maintenance (O&M) cost of the PVs and the batteries, respectively; N, the number of time intervals; λ_{PV} and

λ_{Batt}, the PV panels and batteries cost, A_{PV} and $P_{CapBatt}$ the PV surface area and battery capacity; OM_{PV} and OM_{Batt}, the PVs and batteries O&M cost; ν, the escalation rate (changes in the cost of generating or supplying energy with the PVs or batteries); γ, the interest rate of generating energy with the PVs; and β, the inflation rate (an increase in the cost of supplying energy with the batteries). The values of these parameters are described in Section 4.3.

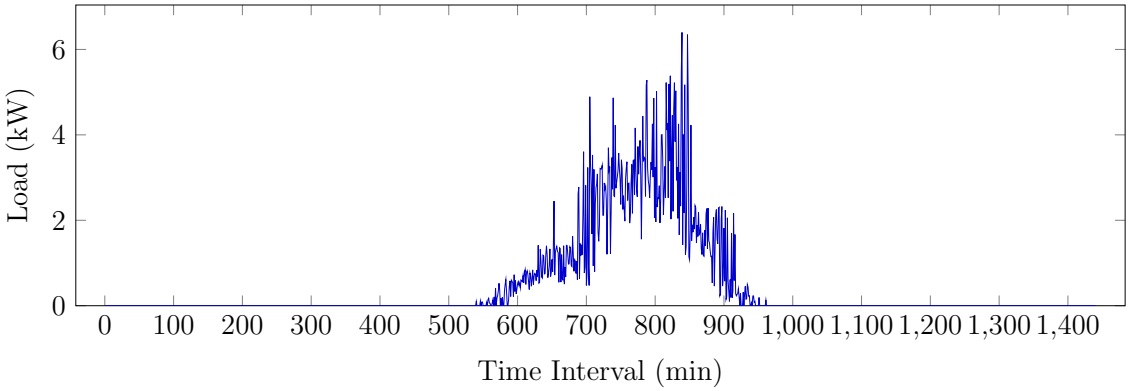

Fig 4.4: Example of a PV energy generation profile.

An example of the energy generation profile of a PV present in scenario based on this model is shown in Figure 4.4. In the producers side there is a battery bank that functions as a backup in case the PVs cannot supply the demand. The total number of batteries is equal to the total number of PVs with a capacity of 15 kWh each. In all the models where the batteries are present, they begin fully charged. In time intervals where the PVs cannot supply the demand, the energy to supply it is taken from the batteries. Even after this is done, if the supply is not met, curtailment is done on the demand. When the PV energy generation exceeds the demand in a specific time interval, the excess is stored in the batteries that are not fully charged in an evenly way.

The availability can be defined as:

$$A = \frac{A_{PV}}{A_{PV_{max}}}, \tag{4.3}$$

where $A_{PV_{max}}$ is the maximum surface area of the PVs. The constraints are:

$$\begin{aligned} A_{PV_{min}} \leq A_{PV} \leq A_{PV_{max}}, \\ P_{CapBatt_{min}} \leq P_{CapBatt} \leq P_{CapBatt_{max}}, \end{aligned} \tag{4.4}$$

and enclose the PV panel area and the maximum capacity of the batteries between a minimum and a maximum value. In the model of this scenario, the controllers are at the

producers side and exchange information within themselves. The producer entities are clustered at a side of the topology of the SG, equal to the number of houses (See Figure 4.5).

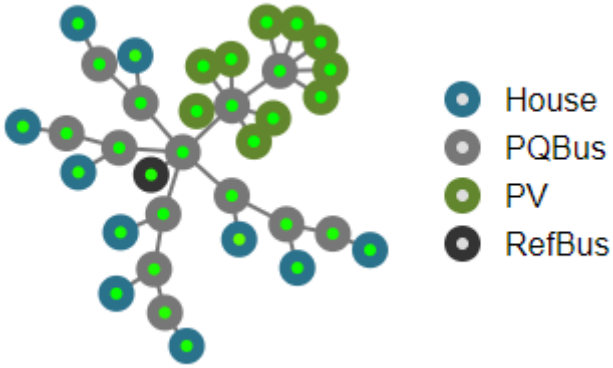

Fig 4.5: An example of a centralized SG topology.

4.2.3 Distributed Generation and Two-way Communication

In this scenario the controllers reside at both the producers and consumers sides. The model of the consumers side is similar to the one presented in Section 4.2.1, but a fourth term is added to the cost function to incentive consumers that prefer parts of their demand to come from renewable energy resources.

$$\begin{aligned} C_{ren} &= d_{ren} R_t, \\ R_t &= \sum_{c=1}^{C} r_c^t, \\ Cost_{Ren} &= a_c NR_t^2 + b_c NR_t + c_c - C_{ren}, \\ NR_t &= \sum_{c=1}^{C} nr_c^t, \end{aligned} \tag{4.5}$$

where d_{ren} is the renewable energy cost per energy consumed, R_t and NR_t are the total renewable and non-renewable energy consumed, respectively, up to a time interval $t \in [1, T]$ for $r_c^t \in R$ and $nr_c^t \in NR$. R and NR are matrices of dimensions $(C \times T)$. The constraints are:

$$\begin{aligned} (r_c^t + nr_c^t) - \tau_l \leq dv_c^t \leq (r_c^t + nr_c^t) + \tau_u, \\ 0 \leq (R_t + NR_t) \leq E_{max}, \\ NR_t \leq NR_{max}, \end{aligned} \tag{4.6}$$

These constraints are similar to the ones presented in Section 3.1.1. But, they consider the renewable and non-renewable parts of the demand. The last constraint ensures the stability of the non-renewable energy resources. The model of the producers side is similar to the one presented in 4.2.2. Since there is communication between the consumers and producers sides and this time there is a part of the total energy consumed that comes from non-renewable resources, an additional constraint is defined for this model:

$$P_{PV_t} + P_{Batt_t} \leq P_{D_t}, \tag{4.7}$$

where P_{PV_t}, P_{Batt_t}, and P_{D_t} are the energy generated or supplied by the PVs or the batteries, the total demand communicated from the consumers to the producers, respectively. This constraint prevents the system from oversizing, thus adding cost to the deployment of the SG.

4.2.4 Centralized Generation and Two-way Communication

In the model of this this scenario the controllers reside at both the producers and consumers sides. The controller of the consumers side is the same as the one presented in Section 4.2.1. The controller of the producers side is the same as the one presented in Section 4.2.2, but with the constraint in Equation 4.7.

4.3 Problem Formulations

With the producers and consumers models defined for each scenario, the corresponding MOO problems can be formulated as follows.

4.3.1 Distributed Generation and One-way Communication

As explained in Section 4.2.1, the model of this scenario is the same as in Section 3.1, since the optimization is only performed on the consumers side. Because the experiments are done with 500 houses, E_{max} is set to 15780 kW. The values of the other relevant variables

are listed in Table 4.1.

$$\begin{aligned} \min \quad & \left\{ \sum_{t=1}^{T} \Big(Cost\,(E_t) \Big), - \sum_{t=1}^{T} \Big(Utility\,(E_t) \Big) \right\} \\ \text{s.t.} \quad & ds_c^t - \tau_l \leq dv_c^t \leq ds_c^t + \tau_u \forall c \in [1, C] \wedge \forall t \in [1, T], \\ & 0 \leq E_t \leq E_{max} \forall t \in [1, T], \end{aligned} \tag{4.8}$$

Table 4.1: Values of the relevant variables in the CU problem.

$\mathbf{a_c}$	$\mathbf{b_c}$	$\mathbf{c_c}$	**C**	**T**	$\mathbf{E_{max}}$	$\tau_\mathbf{l}$, $\tau_\mathbf{u}$
0.2	0.3	0.05	500	1-1400 mins	15780 kW	0.08 kW

4.3.2 Centralized Generation and One-way Communication

The optimization is done on the producers side cost of producing or supplying energy with the PVs or the batteries, respectively. The objectives are minimized and maximized in terms of A_{PV} and $P_{CapBatt}$:

$$\begin{aligned} \min \quad & \{C_{PVBatt}, -A\} \\ \text{s.t.} \quad & A_{PV_{min}} \leq A_{PV} \leq A_{PV_{max}}, \\ & P_{CapBatt_{min}} \leq P_{CapBatt} \leq P_{CapBatt_{max}}, \end{aligned} \tag{4.9}$$

The values of the relevant variables and constraints for this problem are listed in Tables 4.2 and 4.3. These values are derived, in part, from [22].

Table 4.2: Values of the relevant variables in the CA problem.

N	$\lambda_\mathbf{PV}$	$\mathbf{OM_{PV}}$	λ_{Batt}	$\mathbf{OM_{Batt}}$	ν	γ	β
1-1400 mins	$\$450/m^2$	$\$7.6 \times 10^{-5}$	\$100/kWh	$\$1.9 \times 10^{-5}$	1.9×10^{-5}	1.1×10^{-8}	7.6×10^{-9}

Table 4.3: Values of the constraints in the CA problem.

$\mathbf{A_{PV_{min}}}$	$\mathbf{A_{PV_{max}}}$	$\mathbf{P_{CapBatt_{min}}}$	$\mathbf{P_{CapBatt_{max}}}$
0 m^2	619.4 m^2	0 kWh	15 kWh

4.3.3 Distributed Generation and Two-way Communication

The optimization is performed the producers and consumers sides. In the consumers side the optimization problem is defined as:

$$\begin{aligned} \min \quad & \left\{ \sum_{t=1}^{T} \left(Cost_{Ren} \left(NR_t, R_t \right) \right), - \sum_{t=1}^{T} \left(Utility \left(NR_t + R_t \right) \right) \right\} \\ \text{s.t.} \quad & (r_c^t + nr_c^t) - \tau_l \leq dv_c^t \leq (r_c^t + nr_c^t) + \tau_u \wedge \forall t \in [1, T], \\ & 0 \leq (R_t + NR_t) \leq E_{max} \forall t \in [1, T], \\ & NR_t \leq NR_{max}, \end{aligned} \tag{4.10}$$

The values of the relevant variables and constraints are listed in Table 4.1. At the producers side the optimization problem is the same as the one in Equation 4.9 plus the other two constraints in Section 4.2.3:

$$\begin{aligned} \min \quad & \{-C_{PVBatt}, -A\} \\ \text{s.t.} \quad & A_{PV_{min}} \leq A_{PV} \leq A_{PV_{max}}, \\ & P_{CapBatt_{min}} \leq P_{CapBatt} \leq P_{CapBatt_{max}}, \\ & P_{PV_t} + P_{Batt_t} \leq P_{D_t}, \end{aligned} \tag{4.11}$$

The values of the relevant variables and constraints are listed in Tables 4.2 and 4.3. The maximum amount of non-renewable energy that can be consumed is set to 3 kW.

4.3.4 Centralized Generation and Two-way Communication

The optimization at the consumers side is the same as the one described in Section 4.3.1 and the optimization at the producers side is the same as the one described in the previous section.

4.4 Scenario Implementation

The scenarios are simulated through a co-simulation framework. It allows the modeling of entities at their system level by integrating them through operational coupling methods [23] that enable them to exchange data while running on multiple domains with different time steps [24]. Consequently, the assessment of large-scale systems, such as SGs, is possible before they are deployed. Some of the current co-simulation frameworks for this purpose

include the Transactive Energy Simulation Platform (TESP) of the Pacific Northwest National Laboratory (PNNL) and *Mosaik*.

4.4.1 Mosaik Simulators and Control Mechanisms

Mosaik is a SG co-simulation framework that allows developers to create, modify, reuse, and combine new and existing simulators to create large-scale SG scenarios [25]. This can be achieved through the *Simulator* class. Classes that inherit from this class must implement the *create*, *step*, and *get_data* methods. In the *create* method the entities of a specific simulator are created with their unique identifiers and type. These identifiers differentiate the entities when, for example, communicating their information to other entities. In general, its format is {*sim-model-name*}-{*sim-model-number*}.{*model-type*}_{*model-number*}. For example, a house could have a unique identifier as *HouseholdSim-0.House_4*. Once the entities are created, the *step* method can be called to carry out their behavior.

The *step* method performs a simulation step based on some input data for a time interval. This input data can be information posted by other entities during running time. For example, the information of all the houses in a simulator can be loaded in memory from a Comma Separated Values (CSV) file, at a specific time, allowing for operations to be done on it. Then, this information is stored in a data structure that is later read in the *get_data* method and posted to other entities. The *step* method returns the time at which the data is to be posted. It is important to remark that the time step of the simulators do not have to be the same, as *Mosaik* handles this by setting the global time step to the time step of the most frequent simulator. For instance, if time step of a simulator is t and that for another is $t + n$, the second would see the information posted by the first n times.

In the *get_data* method the data of a simulator is posted to other simulators. The data prepared in the *step* method is put in another data structure that contains the source entity unique identifier and the information it wants to post. The information must contain the attribute name and its associated value. For example, a house can add information to the data structure as a dictionary:

```
{
    'HouseholdSim-0.House4': {
        'P_out': 134.23
    }
}
```

Listing 4.1: Partial example of a dictionary with information of a house entity.

Any information posted by a specific entity can only be read by the entities connected to it. Connections can be defined in a JavaScript Object Notation (JSON) file (See Listing 4.2) for *PYPOWER* and at running time implementing the *connect*, *connect_one_to_many*, and *connect_randomly* methods. The JSON file contains nodes, transformers, and branches with their respective relevant values. The branches are pairs of nodes or nodes with transformers, in this case. The connection methods are called on a world object that represents the scope of a simulation. The *connect* method connects two entities through their specified attributes. For example, a house can be connected to a node through their respective *P_out* and *P* attributes. From the node's point of view, the power flows out while from the house, it flows in. The other two methods can be used to connect an entity to a set of entities (*connect_one_to_many*) or a set of entities to a second set of entities (*connect_randomly*). Other options can be passed as arguments to these methods to define the distribution of the connections (evenly or not). *Mosaik* handles the exchange of information between entities through plain network sockets and/or JSON encoded messages.

```
{
    "base_mva": 10,
    "bus": [
        ["tr_pri",    "REF", 20.0],
        ...
    ],
    "trafo": [
        ["transformer", "tr_pri", "tr_sec", 0.25, 4.2, 0.00275, 6.9, 360.8]
    ],
    "branch": [
        ["branch_1",  "tr_sec",    "node_a1",  0.100, 0.2542, 0.080425, 0.0,
     240.0],
        ...
    ]
}
```

Listing 4.2: A partial example of a JSON file to specify entity connections for *PYPOWER*.

For simplicity, the scenarios are implemented using three simulators including the *Household*, *PVCSV*, and *PYPOWER*. The first two are implementations based on *Mosaik's* CSV simulator that allows simulator instances to load data from files. The file that holds the PV energy generation profiles contains two columns corresponding to the time-stamp in one minute time step and the energy generated in watts. The other file that holds the energy consumption profiles of the houses contains a list with the unique identifiers of

the nodes and a column correspond to the time-stamp in 15 minutes time step as well as additional columns corresponding to each house energy demand. The *PYPOWER* simulator allows the definition of energy entities including nodes, branches, transformers, and a reference bus with their relevant values including the impedance, active and reactive power, voltage angle, etc. They are set to *Mosaik's* defaults. Other simulators including *WebVis* and *HDF5* are used to visualize the topology of the simulations at running time through a web application hosted at *http://localhost:8000* and to save the *Household* and *PVCSV* simulators time series data in a hierarchical data format file for later analysis, respectively.

The topology for the distributed and centralized scenarios has five branches with one transformer and 100 houses each.

Algorithm 2 General algorithm of the *step* method.

```
1:  procedure StepSAMController(time, inputs)
2:      sim_current_data ← readRowCSV(time)
3:      all_profile_data ∪ sim_current_data
4:      variables ← get_problem_variables()
5:      variables ∪ all_profile_data
6:      opt_problem ← CUorCAProblem(variables)
7:      algorithm ← NSGA-II(opt_problem)
8:      results ← run algorithm
9:      result ← minimum_cost(results)
10:     optimized_loads ← get_variables(result)
11:     get_data_structure ∪ optimized_loads
```

The MASs for controlling the behavior of the *Household* and *PVCSV* simulators are implemented in their respective *step* methods. In general, these methods follow the procedure in Algorithm 2. In line 2, the row of a CSV file corresponding to the current time step is loaded. Then, the data of that row is added to a matrix that contains all the data up to the time step for all the entities of a specific simulator. In lines 4 to 5, the variables for the optimization problem are set and the matrix is added to it. The MOO problem, either the CU or Cost-Availability (CA), is instantiated with the variables in line 6. After this, the NSGA-II is initialized is with the problem. This EA is used for these simulations over the others, because it achieved diverse solutions in the previous experiments and better running times than IBEA. The optimization problem is solved and the solution with the minimum cost is set to *result* in lines 8 and 9, respectively. Finally,

the variables representing the generated or consumed energy are assigned to the data structure for the *get_data* method.

As the experiments for the DR case study, the experiments for the TE scenarios were carried out on a Linux 14.04 LTS compute node running on Chameleon Cloud. An image of this node is also available (See Appendix C).

4.5 Experimental Results

Figure 4.6 shows the energy demand and supply for a house and its associated PV, respectively, obtained for the distributed generation one-way communication scenario. Given that the optimization is carried out only on the consumers side and no information is ever exchanged between the producers and the consumers, the PV is unable to supply the demand between time intervals 0 to 40 and 64 to 96. From time intervals 41 to 63, the supply exceeds the demand between a minimum of 0.009 kW and a maximum of 3.74 kW. This period represents a total of 1.43 kWh of energy wasted.

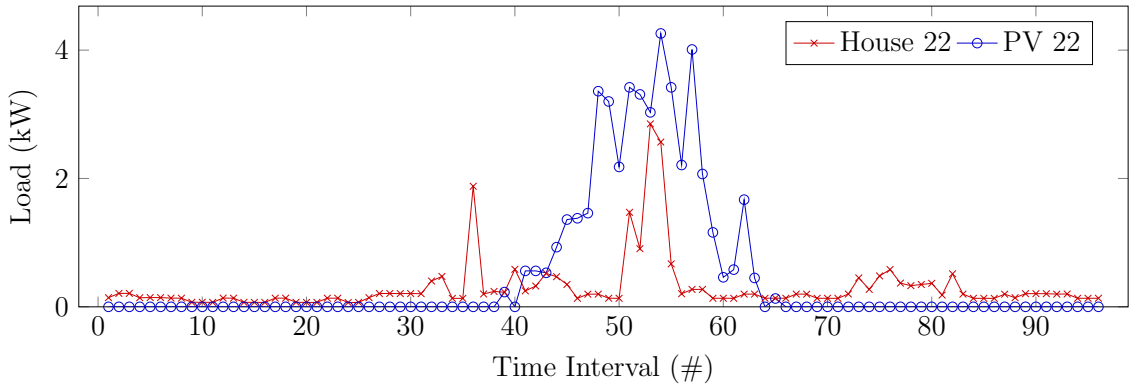

Fig 4.6: Comparison between a house load profile and its associated PV energy generation profile in the distributed one-way scenario.

The results for the centralized one-way scenario are shown in Figure 4.7. Similar to the results obtained in the previous scenario, in this scenario the optimization is carried out only on the producers side and no information is ever exchanged between the producers and the consumers. The PVs are unable to supply the demand between time intervals 0 to 43, 60, 61, and 63 to 96. Between time intervals 44 to 59 and 62 the supply exceeds the demand between a minimum of 92.44 kW and a maximum of 1623.31 kW. This period represents a total of 765.79 kWh of energy wasted.

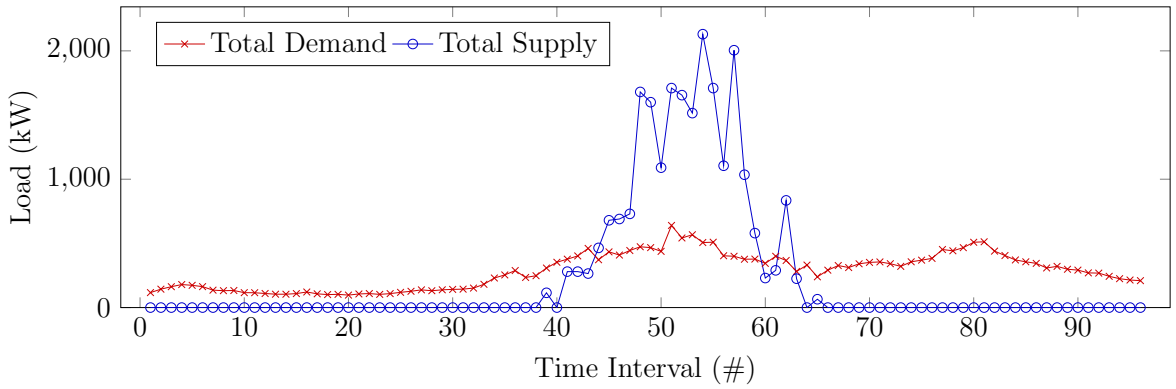

Fig 4.7: Comparison between the total house load profile and total PV energy generation profile in the centralized one-way scenario.

In the centralized generation two-way communication scenario, the demand is supplied in all the time intervals (See Figure 4.8). This is due to the communication of information between the entities and to the batteries that respond to the demand in case the PVs cannot supply it even after the optimization is done in the producers side. In this case the wastage of energy is 29.32 kWh, being this less than the wasted energy in the previous scenario.

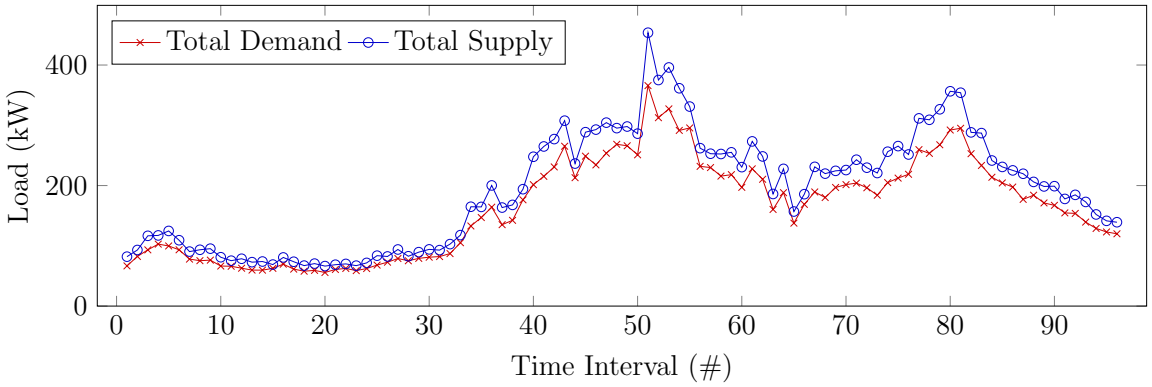

Fig 4.8: Comparison between the total house load profile and total PV energy generation profile in the centralized two-way scenario.

Figure 4.9 shows the renewable and non-renewable energy consumed by a house in the distributed two-way scenario. Using the cost function of the DR scenario, the cost paid by this house can be obtained. Figure 4.10 shows two series of the cost function as if the energy consumed had no incentives (left) and the one obtained applying the incentives (right). When applying the incentives, the cost is reduced. The decrease in cost can be controlled modifying the cost function coefficients to obtain more realistic results. In terms

of supply-demand fitting, similar results were obtained for the individual loads as in the previous scenario.

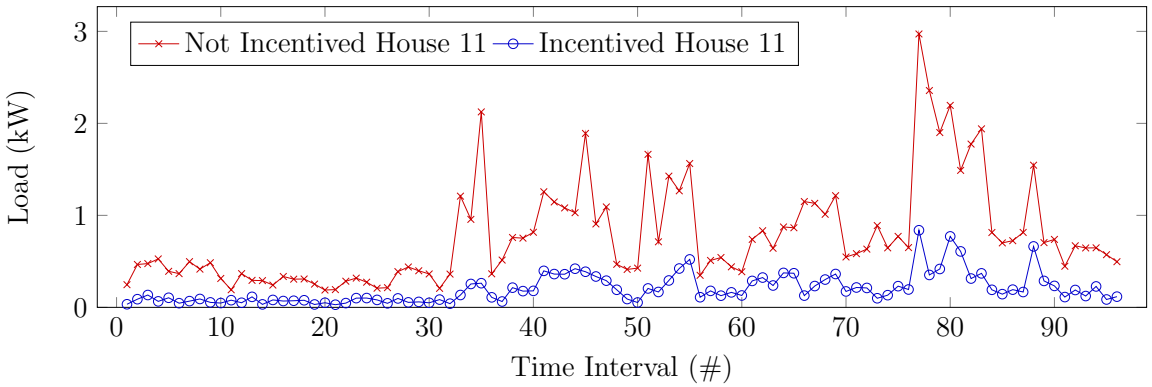

Fig 4.9: Comparison between the renewable and non-renewable load profiles in the distributed two-way scenario for House 11.

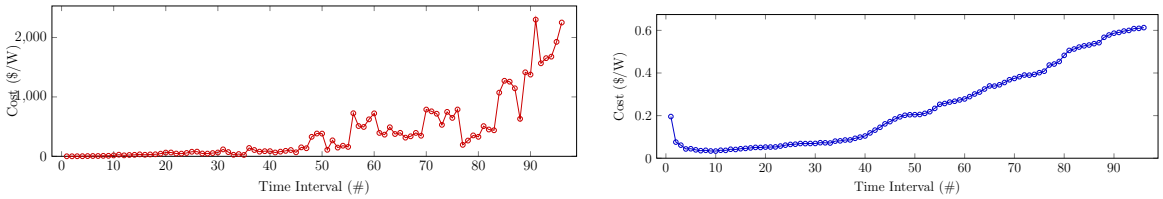

Fig 4.10: Comparison of the cost per kilo-watt of house 11 as if it consumed the energy without incentives (left) and with incentives (right).

4.6 Conclusion

In this chapter, several energy grid scenarios based on the DCG are modeled through a co-simulation framework that integrates them at different SG architecture levels. The models consider the cost and availability of producing energy and the cost and utility of consuming it. Throughout the optimization of these objectives, intelligence (MAS) is added to some of the entities in the scenarios to controlling their behavior. These mechanisms also allow the exchange of data between the entities for improved decision-making on the parameters specified by the SG producers and consumers.

The results show that energy services can be coordinated on the entities that exchange information about their energy profiles to achieve efficient supply-demand fitting. Thus, minimizing the amount of energy wasted and the costs associated with producing or consuming it.

Chapter 5

Conclusion and Future Work

Recapitulating, traditional energy grids are being modernized with the introduction of two new technologies: SGs and TE. The objectives of this work include the investigation of optimization techniques suitable for SG modeling and the research of TE simulation scenarios.

A framework for evaluating the performance of EAs in the context of solving SG related problems is proposed where SG custom optimization problems, EAs, and performance metrics can be created and instantiated. The CU and CA problems minimize the cost of producing or consuming energy, maximize the utility of consuming energy, and maximize the availability of energy resources such as PVs and batteries. The EAs investigated in this work were selected due to their various schemes for searching for solutions in decision spaces. A set of performance metrics were implemented to evaluate their performance in terms of accuracy, cardinality, and diversity. Additionally, a DR case study is presented to demonstrated that when considering the energy policies of consumers and producers in MOO problems, day-ahead load forecasting is possible on SGs.

The DCG is proposed as a framework to evaluate how SG entities behave under different TE distribution and communication strategies. Four scenarios derive from it including centralized and distributed generation of energy and one-way and two-way communication of information between consumers and producers. In each scenario, MASs function as mechanisms to controlling the behavior of energy entities such as houses and PVs. The scenarios were implemented through *Mosaik*, a co-simulator framework for SG modeling. The results show that better supply-demand fitting can be obtained when energy entities exchange information about their energy supply or demand profiles while optimizations take part on their behavior. Thus, decreasing the cost of energy consumption.

Future work will be focused on extending the capabilities of the optimization framework

to support more EAs and performance metrics as well as MOO problems. Additional TE scenarios are to be developed for new models that consider the energy or market behavior of entities including producers and prosumers such as windmills and electric cars, respectively. New MASs could be developed to control the communication of information between entities considering schemes of frequency and priority. Furthermore, other green energy strategies could be added to the models that follow specific patterns of energy generation or consumption.

References

[1] Konstantinos Kampouropoulos et al. “Optimal control of energy hub systems by use of SQP algorithm and energy prediction”. In: *IECON Proceedings (Industrial Electronics Conference)* (2014), pp. 221–227.

[2] C. Fatih Kucuktezcan, V. M. Istemihan Genc, and Osman Kaan Erol. “An optimization method for preventive control using differential evolution with consecutive search space reduction”. In: *2016 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)* (2016), pp. 1–6.

[3] Ramesh Rajagopalan. “A multi-objective optimization approach for efficient energy management in smart grids”. In: *2015 IEEE Green Energy and Systems Conference (IGESC)* (2015), pp. 7–10.

[4] Siwei Jiang et al. “Consistencies and contradictions of performance metrics in multiobjective optimization”. In: *IEEE Transactions on Cybernetics* (2014). ISSN: 21682267. DOI: 10.1109/TCYB.2014.2307319.

[5] Wilson Rivera and Manuel Rodriguez. “Towards Cloud Services in Smart Power Grid”. In: *2016 IEEE Innovative Smart Grid Technologies - Asia (ISGT-Asia)* (2016), pp. 570–573.

[6] Q. Zhang and Hui Li. “MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition”. In: *IEEE Transactions on Evolutionary Computation* 11.6 (2007), pp. 712–731.

[7] Sergio Salinas, Ming Li, and Pan Li. “Multi-objective Optimal Energy Consumption Scheduling In Smart Grids”. In: *Tsg* 4.1 (2013), pp. 341–348. DOI: 10.1109/ICCAIRO.2017.28.

[8] E. Zitzler and K. Simon. “Indicator-Based Selection in Multiobjective Search”. In: *8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)* 3242.i (2004), pp. 832–842.

[9] Kalyanmoy Deb, Manikanth Mohan, and Shikhar Mishra. "Evaluating the ϵ-Domination Based Multi-Objective Evolutionary Algorithm for a Quick Computation of Pareto-Optimal Solutions". In: *Evolutionary Computation* 13.4 (2005), pp. 501–525.

[10] N. Hansen and A. Ostermeier. "Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation". In: *Proceedings of IEEE International Conference on Evolutionary Computation*. May 1996, pp. 312–317.

[11] S. Kukkonen and J. Lampinen. "GDE3: the third evolution step of generalized differential evolution". In: *2005 IEEE Congress on Evolutionary Computation* 1 (2005), pp. 443–450.

[12] A. J. Nebro et al. "SMPSO: A new PSO-based metaheuristic for multi-objective optimization". In: *2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making(MCDM)*. Mar. 2009, pp. 66–73.

[13] K Deb et al. "A Fast and Elitist Multiobjective Genetic Algorithm: NGSA-II". In: *IEEE Transactions on Evolutionary Computing* 6.2 (2002), pp. 182–197.

[14] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. "SPEA2: Improving the Strength Pareto Evolutionary Algorithm". In: *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems* (2001), pp. 95–100.

[15] Nery Riquelme, Christian Von Lucken, and Benjamin Baran. "Performance metrics in multi-objective optimization". In: ().

[16] G. G. Yen and Z. He. "Performance Metric Ensemble for Multiobjective Evolutionary Algorithms". In: *IEEE Transactions on Evolutionary Computation* 18.1 (Feb. 2014), pp. 131–144.

[17] Fonseca C M Lopez-Ibanez M Paquete L Beume M. et al. "On the Complexity of Computing the Hypervolme Indicator". In: *IEEE, Transactions on Evolutionary Computation* 15.5 (2009), pp. 1075–1082.

[18] E Zitzler et al. "Performance assesment of multiobjective optimizers: an analysis and review". In: *Evolutionary Computation* 7.2 (2003), pp. 117–132.

[19] David Hadka. "MOEA Framework User Guide: A free and open source Java framework for multiobjective optimization". In: (2014). URL: `http://www.moeaframework.org/`.

[20] J. Mambretti, J. Chen, and F. Yeh. "Next Generation Clouds, the Chameleon Cloud Testbed, and Software Defined Networking (SDN)". In: *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*. Oct. 2015, pp. 73–79.

[21] GWAC. "GridWise Transactive Energy Framework Draft Version". In: *GridWise Architecture Council on Transactive Energy* 1 (2013), pp. 1–23. DOI: PNNL-22946.

[22] Mohammad B Shadmand et al. "Multi-Objective Optimization and Design of Photovoltaic-Wind Hybrid System for Community Smart DC Microgrid". In: 5.5 (2014), pp. 1–9.

[23] Van Hoa Nguyen et al. "On Conceptual Structuration and Coupling Methods of Co-Simulation Frameworks in Cyber-Physical Energy System Validation". In: *Energies* 10.12 (1977). ISSN: 1996-1073.

[24] S. Sicklinger et al. "Interface Jacobian-based Co-Simulation". In: *International Journal for Numerical Methods in Engineering* 98.6 (), pp. 418–444.

[25] Steffen Schutte, Stefan Scherfke, and Michael Sonnenschein. "mosaik - Smart Grid Simulation API". In: *Proceedings of SMARTGREENS 2012 - International Conference on Smart Grids and Green IT Systems* 2 (2012), pp. 14–24.

Appendices

Appendix A

Optimization Platform Documentation

A.1 Installing *Platypus*

This section assumes that the user has *root* privileges on a Linux 14.04 LTS node.

A.1.1 Miniconda

The first step involves downloading *Miniconda*, an *Anaconda* package and environment manager that allows developers to create environments with the desired *Python* version and modules in a structured way. In the terminal, run the following commands to download and install it. When done, the current terminal should be closed and a new one should be open to be able to use the command *conda*.

```
1 $ cd /tmp/
2 $ wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.
    sh # Change to Linux-x86.sh for 32 bits
3 $ chmod +x Miniconda3-latest-Linux-x86_64.sh # or Miniconda3-latest-Linux-
    x86.sh
4 $ ./Miniconda3-latest-Linux-x86_64.sh # or Miniconda3-latest-Linux-x86.sh
```

Listing A.1: Downloading and installing Miniconda latest version.

A.1.2 Virtual Environment

In this step, a *Python* virtual environment is created through *Miniconda*. A virtual environment is a self-contained directory tree that contains a version of *Python* in addition

to other packages. This is done through *Miniconda's* command setting the virtual environment's name and *Python* version to 3.4.5. To begin working on the virtual environment, it must be activated. Run the following commands in the terminal to do this:

```
$ conda create -n optimization python=3.4.5
$ source activate optimization
(optimization) $
```

Listing A.2: Creating and activating a *Python* 3.4.5 virtual environment.

A.1.3 Platypus

Git is an open source distributed version control application to manage git repositories. Run the command *git* in the terminal. If *git* is not installed, run the following commands:

```
(optimization) $ sudo apt-get update
(optimization) $ sudo apt-get install git
```

Listing A.3: Installing git.

Once installed, use *git* to download *Playtpus* and install it using *Python* through the following commands:

```
(optimization) $ git clone https://github.com/Project-Platypus/Platypus.
  git
(optimization) $ cd Platypus
(optimization) $ python setup.py develop
```

Listing A.4: Downloading and installing *Platypus* in the virtual environment.

Remember to activate the virtual environment (*source activate optimization*) before running a program that uses the *Platypus* module.

Testing *Platypus*

To test that *Platypus* is installed correctly, save the following example to the node where it was just installed and run it:

```
from platypus import NSGAII, DTLZ2

problem = DTLZ2()

algorithm = NSGAII(problem)
algorithm.run(10000)
```

```
7
8 for solution in algorithm.result:
9     print(solution.objectives)
```

Listing A.5: *Platypus* post installation example.

In this case a MOO benchmark, the Deb Thiele Laumanns and Zitzler 2 (DTLZ2), is run using NSGA-II with 10000 function evaluations. At the end, the values of the objectives obtained by each solution in the solution set are printed to the terminal.

A.2 Getting the Optimization Platform

To download the optimization platform and its dependencies, run the following commands:

```
1 (optimization) $ cd /tmp/
2 (optimization) $ git clone https://github.com/chicodelarosa/
    SGOptimizationPlatform.git
3 (optimization) $ cd SGOptimizationPlatform
4 (optimization) $ pip install -r requirements.txt
```

Listing A.6: Downloading the optimization platform and installing its dependencies.

A.3 Using the Optimization Platform

The platform comes with all the required variables and files to run the CU problem. In case, new optimization problems, variables, and files are needed they can be added to the corresponding directories. The directory tree of the optimization platform is shown in Figure A.1. The *CustomProblems* directory holds the MOO problems implemented with *Platypus' Problem* class. Any custom problem that wants to be red outside this directory needs to be imported to the *init* file.

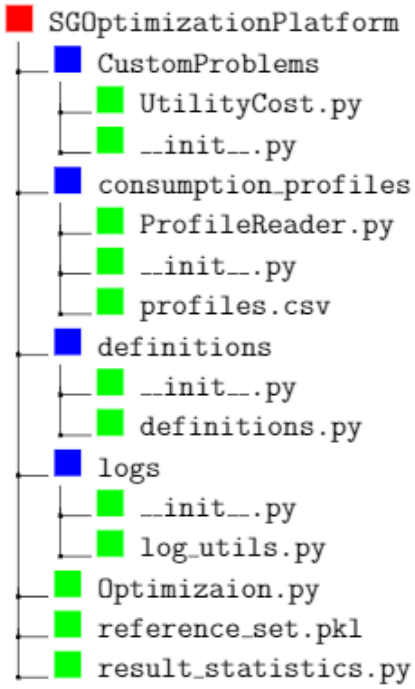

Fig A.1: Optimization framework directory tree.

A.3.1 Adding Optimization Problems

To add new optimization problems to the platform, the *Platypus Problem* class must be used. Any class that inherits from the *Problem* class must define its own constructor and *evaluate* method. In the constructor, the *super()* method must be called, this method receives three arguments including the number of variables, objectives, and constraints of the problem.

The variable types can be set as *Real*, *Integer*, *Binary*, *Permutation*, and *Subset*. Custom types can also be created as needed. Depending on the optimization problem, these types can be used. When defining types, the lower and upper bounds of each variable must be set. The length of the types array must be equal to the number of variables. The constraints are the operators for each constraint of the optimization problem. Valid constraints include "==", "<", ">", "<=", ">=", and "!=". The length of the constraints array must be equal to the number of constraints.

The directions tell *Platypus* how to optimize each objective. The *Problem* class provides two instance fields in case an objective is to be maximized or minimized. These are callable as *Problem.MAXIMIZE* or *Problem.MINIMIZE*, respectively. The length of the directions array must be equal to the number of objectives.

A.3.2 Adding Evolutionary Algorithms

To add new EAs to the optimization platform, the *Platypus Algorithm* class must be used. Any class that inherits from the *Algorithm* class must define its own *constructor*, *step*, *iterate*, and *run* methods. In the constructor, the *super()* method must be called, this method receives one argument, the problem.

In the *step* method the number of function evaluations is checked as the stop condition for any EA optimizing the problem. Also, in this method, the *iterate* method is called if the stop condition has not been reached. This method is the heart of any EA. There, the algorithm evaluates the objectives, assigns the fitness of its population, defines or updates its population archives, and performs selections, mutations and crossovers between the current population and the offspring. In the *run* method some setup is done prior calling the *set* method.

A.3.3 Adding Performance Metrics

To add new performance metrics to the optimization platform, the *Platypus Indicator* class must be used. Classes that inherits from this class, must implements their own constructor and calculate methods. The constructor may receive the parameters of the performance metric and must call the *super()* method.

Depending on the type of performance metric, one or more solutions sets are passed as arguments to the *calculate* method. This method returns the value or values associated with the implementation of a specific metric.

A.3.4 Entity Data

The CU problem works with consumer consumption profiles. These profiles are kept in CSV files. Before the optimization problem is optimized, the profiles are read from the CSV file. The values are converted to a list of lists that is kept in memory while the optimization takes place.

Other entity profiles such as information of PVs can be included in the optimization platform by adding a directory and an *init* file in addition to a reader in case the information kept in its CSV is stored in a way that does not correspond to profile, time interval. In this way, when running custom optimization problems, the data can be loaded into memory accordingly.

A.3.5 Definitions

The *definitions* directory holds files with variables needed by the objects that optimize a problem. Definitions include values such as E_{max} and the cost function coefficients. Again, the *init* file allows programs outside this directory to read these variables.

A.3.6 Logs

The *logs* directory holds the *Logger* class that contains methods for keeping information about the progress of the optimizations. It also contains a method that saves that information in text files. This is useful for carrying out statistical analyses on the results obtained by the EAs. The *init* file allows programs outside this directory to read these methods.

A.3.7 Putting Everything Together

In the *Optimization* file, all the information of profile files, variables, and objects are imported. In line 118, the *evaluator* is defined for the experimenter. The number of threads is passed as an argument to the *ProcessPoolEvaluator* method, using the *cpu_count* method, to tell *Platypus* how many of them it can use. The experiments are run on the CU problem with different EAs while the performance metrics are calculated and all the information in each round is logged and saved locally for later analysis. In the case another problem is to be optimized, the *problem* variable must be set to the installation of the specific custom problem.

Lines 84 through 100 are commented and includes creation of the reference solution set of non-dominated solutions that is used by the performance metrics. At the end, a pickle object is used to dump the reference solution set object to a file. Once the reference solution set is saved in a file, lines 104 to 105 load the information stored in it and set it to the *reference_set* variable.

Appendix B

GitHub Repositories

The GitHub repositories of the optimization platform and the TE MAS simulations are available upon request at dan.rosa@upr.edu. The following sections contain the links.

B.1 Optimization Platform

https://github.com/chicodelarosa/SGOptimizationPlatform

B.2 Transactive Energy Multi-Agent System Simulations

https://github.com/chicodelarosa/TESAMSimulation

Appendix C

Virtual Machine Disk Images

The virtual machine disk images of the optimization platform and the TE Multi-Agent System simulations are available upon request at dan.rosa@upr.edu. They are in qcow2 format.