Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Smart Prices for the Smart Grid

Master Thesis

Ivo de Concini

September 2015

Advisor: Prof. Dr. Lothar Thiele

Department of Electrical Engineering and Information Technology, ETH Zürich

Abstract

The ongoing transition from the traditional electrical grid to the digitally interconnected smart grid and the increasing deregulation of electricity markets, allow to envision new types of distributed local energy markets. In order to develop such markets, it is important that energy producers and consumers can access a local energy price.

In this project, we define an agent-based real-time local energy pricing-model for the electrical grid, based on supply and demand. We describe how such a model can be computed, by solving a minimum-cost flow problem. We derive theoretical bounds and perform simulations, that show the low-sensitivity of our solution approach. We develop realistic trading-strategies for the network-agents and use them, to perform experiments that validate its dynamic stability.

Finally, we present the implementation of a working prototype, analyze its performance, to locate possible bottlenecks, and propose solutions, to remove them.

Contents

Chapter 1

Introduction

An electrical grid (short: *grid*) is an interconnected network with the goal to deliver electric power from generating stations, suppliers, to demand centers via transmission lines, that operate at different voltage levels via substations, that transform the power between the different voltage levels [1].

The movement towards renewable and sustainable energy production of the last decades however, has brought significant changes to the structure of the grid: While the energy production used to be mostly carried out by large scale power-plants, today it is increasingly common also for smaller actors, like house-holds with small photo-voltaic generation facilities, to provide electricity to other members of the grid.

This shift was enabled mainly by two developments: The first one was a significant modernization of the infrastructure of electrical grids *(smart grids)*, adding among others a digital-communication layer between customers and suppliers. The second one was the liberalization of the energy market in many countries, in the form of legislative efforts[1], specifically targeted to increase the use of renewable energy sources.

Traditionally, electrical energy is traded on two different types of markets:

- *Wholesale* markets, where generators and retailers trade over short-term electricity delivery and for future delivery periods, and
- *Retail* markets, where end-customers can choose between different retailers to buy electricity.

However, the electricity output from sustainable energy sources, like solar-power or wind, is highly dependent on natural circumstances, and therefore in its nature very volatile and hard to predict with high accuracy. Additionally, at the current technological state, it is generally difficult and expensive

[1]e.g. the German Renewable Energy Act (German: Erneuerebare-Energien-Gesetz, EEG)

to store electrical-energy. This fact makes it hard for small electricity generators to trade over future deliveries on wholesale markets, while their limited energy output might not be worth the effort of short-term trading with large retailers. As a consequence, as of today, small generators usually sell their energy to a retailer at predefined rates (*feed-in tariffs*), which then take care of re-distributing it to the end-customer.

The modern smart-grid infrastructure and the general liberalization of the energy markets, allow us to envision new types of markets, where small generators can directly trade with consumers, without the need of a retailer between them. For the development of such markets, it is of major importance that users of energy as well as suppliers have direct access to an *instant* meaningful local energy price, which is determined by the energy-demand, the availability and price of the energy-supply, as well as the costs to use the transmission-infrastructure.

1.1 Prior Work

Since the beginning of the deregulation of the traditionally government-controlled electricity markets in the early 1990s, in favor of more competitive markets, there has been an increased interest by the research community to better study the electricity pricing-mechanisms. These efforts can be roughly subdivided into two main areas, that have been interesting for this project: The first one is focused on the modeling of existing markets, mainly with the goal of accurate *electricity price forecasting (EPF)* [2], while the second one is focused towards developing real-time pricing schemes, as drivers for a more efficient resource-usage (*Smart Management System*) [3] and better load-balancing (*Demand Side Management (DSM))* [4]).

A comprehensive overview of the different typologies of electricity markets, as well as the basics of price formation in post-regulation electricity markets is given in [5, p. 1-20]. A classification of the different modeling approaches for analyzing and predicting electricity prices is provided in [6], where among stochastic, statistical, and artificial-intelligence models, the author also introduces multi-agent models, which simulate the operation of generation- and demand units to obtain prices based on supply and demand. Such models are the basis for many real-time pricing based DSM-systems, as well as for the work presented in this thesis.

A real-time pricing algorithm for the smart-grid, between several demand-centers and *one* energy supplier, has been presented in [7]: Based on the consumption patterns of demand-centers, the authors formulate utility-functions for each demand and use them to formulate a real-time pricing algorithm, that computes the prices by distributedly solving a convex optimization problem, which maximizes the utility of the demands and minimizes the costs for the supplier.

A different dynamic pricing model, based on a game-theoretical approach, has been proposed in [8], which this time assumes multiple suppliers and two types of demands: Traditional electricity users, who pay a fixed price and opportunistic users, who can change their demand, as well as the supplier, based on the electricity price. In this theoretical model, which does not take into consideration transmission costs, electricity suppliers can offer different prices in different regions of the grid, and try to maximize their profits.

The pricing model proposed in this thesis relies on solving a minimum cost flow problem for the electrical grid. An agent-based approach to solve this problem by applying a distributed cost-scaling push-relabel algorithm has been proposed in [9]. Another distributed, market based approach, based on mixed integer programming, has been proposed in [10]. The goal of both of this solutions, is to minimize the production costs of electricity (*fuel cost*): Therefore, they do not include the transmission costs, which one needs to consider to compute a real-time energy price.

1.2 Contributions

In this project we will define a meaningful energy real-time price model for the electrical grid, which we will then formalize in an algorithm, which, based on the principles of supply and demand, determines the current market price of energy at every location of the electrical grid.

Using a mixture of experimental investigations, based on simulations and some theoretical insights, we will evaluate and refine our approach making sure, that it yields consistent and stable results.

We will then use our algorithm to implement a working prototype-system that can be applied to existing, large-scale networks and provides real-time price information. Finally, we will analyze the performance of our prototype and suggest further improvements.

Chapter 2

Problem Formulation

In this chapter, we will define the aim of this project and introduce the assumptions and terms we will use throughout: First we will propose an abstract model of the electrical grid. Then we will define the requirements for a meaningful energy price and use them to evaluate the different options, to implement such a price calculation in our model of the electrical grid. In the end we will settle on the most promising pricing-scheme and define it on a functional level.

2.1 Model of the Electrical Grid

Before we can address the question, how a real-time energy-price for the electrical grid could look like, we need to clearly define the different actors and components of the grid, that can influence the price and the way they can interact with each other.

As already mentioned in the introduction, an electrical grid is an interconnected network with the goal to deliver electric power from electricity generators to demand-centers via transmission lines and sub-stations. In this section we will introduce abstractions for the generators, the demand-centers and the grid infrastructure, which will provide us with the basis for our further considerations.

2.1.1 Sub-Grids and Transportation Costs

The transmission infrastructure of the electrical grid is composed of transmission lines, that transport electrical power at different voltage-levels, and sub-stations, that transform the power between those levels. This infrastructure is operated by an entity, which we call the grid-operator. There can be several grid-operators in an electrical grid.

Generators and demand-centers are connected to a sub-station and are both considered as customers from the perspective of a grid-operator: Generators, pay a *grid-usage cost c_u* to feed power to the grid, while consumers pay a grid-usage cost to draw power from the grid. The grid-usage cost is proportional to the amount of power which flows from or towards a customer and can vary from customer to customer.

The sub-stations are interconnected via transmission lines that can operate at a different voltage: In order to be exchanged at a different voltage level, the power needs to be transformed by the substation, which causes a certain transformation-cost (c_t), which is proportional to the amount of power that is transformed.

In our model, we will substitute these sub-stations with more abstract *sub-grids*, which are not necessarily physical sub-stations, but could for example also represent an electricity retailer. At each time t a sub-grid is characterized by:

1. A *unique id i*, which is used to identify the sub-grid.
2. A well defined *set of neighbors* $j \in N(i)$, that can feed into or draw power from sub-grid i. Possible neighbors are other sub-grids, suppliers and exchanges (Section 2.1.2), as well as demands (Section 2.1.3).
3. A well defined *transportation cost* $c_{i,j}$ for each neighbor $j \in N$. that represent the grid-usage costs as well as the transformation costs for transporting electrical energy from i to a neighbor j.
4. A price $p_i(t)$, i.e. the price we will compute for sub-grid i.

In order to better illustrate how the transportation costs come about in an electrical grid, we will now provide two examples. Figure 2.1 shows a schematic depiction of an electrical grid, containing two sub-grids, operating at a lower voltage (g_6,g_7), that are connected through a third sub-grid (g_8), which operates at a higher voltage. The grid contains two supplying customers (s_1, s_2) and three demanding customers (d_3,d_4,d_5), that are respectively connected to one of the two lower-voltage grids.

Example 1. We now want to compute the transportation costs of 1kWh from s_1 to d_3, which are both members of the same sub-grid: The usage cost for feeding 1kWh into the sub-grid g_6 amounts to $1 * c_{u1,6}$, while the usage-cost for consuming 1kWh from g_6 amounts to $1 * c_{u3,6}$. The total resulting transportation cost per kWh from s_1 to d_3 is therefore:

$$c_{s_1 \rightsquigarrow d_3} = c_{u1,6} + c_{u3,6}.$$

More interestingly, if we want to compute the transportation costs between two grid-customers that are members of two different sub-grids, we also

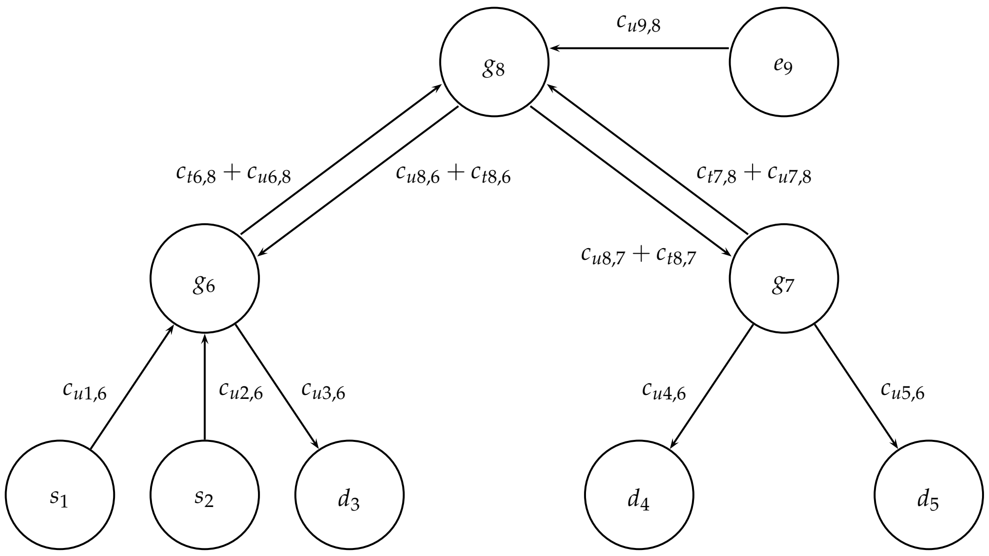


Figure 2.1: Schematic depiction of a grid composed of three sub-grids (with ids: g_x) operating at different voltage-levels, two suppliers (s_x), three demands (d_x) and one exchange node (e_9). The transportation costs are composed by transformation costs ($c_{ti,j}$) and grid-usage costs ($c_{ui,j}$).

need to take into consideration the transformation costs between those grids, as well as the usage costs for additional grids, that are involved in the transportation process.

Example 2. I if we want to transport 1kWh from s_1 to d_4, we need to ship it through g_6, we then to transform the power to a higher voltage, ship it through g_8, re-transform it to a lower voltage and finally ship it through g_7. The total transportation cost therefore amounts to:

$$c_{s_1 \rightsquigarrow d_4} = c_{u1,6} + c_{t6,8} + c_{u6,8} + c_{t8,7} + c_{u7,4}.$$

2.1.2 Energy Suppliers

An energy supplier (short: *supplier*) is a member of the electrical grid, which at a certain time, can feed a well-defined amount of power to into the electrical grid and for which at the same time, it makes a price-offer.

The term supplier is a deliberately general term, which can include generating stations of different sizes, from nuclear-plants to small private generation facilities, as well as abstract entities like e.g. energy brokers, who sell power generated by others.

At each time, a supplier has the following properties:

1. A *unique id i*, which is used to identify the supplier.
2. The amount of *power* $s_i(t)$, it is currently feeding into the grid.
3. A price $p_i(t)$, which a supplier expects for each unit of power.
4. Exactly one *sub-grid j*, associated with a transportation cost $c_{i,j}$ from supplier i to sub-grid j.
5. Access to the price $p_j(t)$ of sub-grid j.

Energy Exchange

The energy exchange is a special supplier, which can supply an unlimited amount of power at each time. We will later use it to balance our model and it can be seen as the equivalent of a *retail* market, which guarantees to meet all energy requests.

2.1.3 Demands

A demand center (short: *demand*) is a member of the electrical grid which draws power from a sub-grid for a price consisting of the sum of the electricity price at that particular sub-grid and the grid-usage cost that applies when transporting .

In our model, a demand is an abstract term for all energy consumers, which can be anything, from small private households to large factories.

At each time t consumer i is characterized by the following properties:

1. A *unique id i*, which is used to identify the demand.
2. The amount of *power* $d_i(t)$ it is currently drawing from the grid.
3. A *price* $p_i(t)$, which it pays for each unit of received power.
4. Exactly one *sub-grid j*, associated with a transportation cost $c_{j,i}$ from sub-grid j to demand i.

2.2 Meaningful Energy Price

As we have already seen in the introduction, there are countless ways in which electrical energy may be traded: Over short-term or long-term deliveries, between suppliers and retailers, between retailers and demands, between suppliers and demands, etc.

Each of this options significantly influences the way the energy price comes about. Therefore, we will first define some requirements, which we pose to a *meaningful* energy price, which we will then use to weigh out the different models we considered, when defining a pricing-scheme.

2.2.1 Requirements

In this section, we will describe and motivate the main requirements we pose to a meaningful energy price.

Immediacy. The nature of the electricity market is special, since electric energy is difficult to store but still needs to be available on demand. This is especially challenging for suppliers that rely on renewable energy sources, like the sun or wind, which are often very volatile and hard to predict. However, in order to be meaningful, the energy price should not rely onto unsure assumptions. Therefore we require, that it is immediate or in other words, that at each time t, it shall represent the state of the system as closely as possible.

Fairness. In order to be meaningful, the energy price should be based on a fair principle, which takes into account the interests of suppliers and demands. In other words, while each supplier must be free to propose its own offer to the market, each demand must also be free to choose the best possible offer.

The requirement of fairness also implies, that suppliers and demands should have access to the same price conditions.

Feasibility. A meaningful price must always be computed on the grounds of a feasible state of the system. This means that the power demands of all consumers must be fulfilled, while the available power supply of suppliers must not be exceeded. Furthermore, we also require that the system is:

- *energy neutral*, meaning that it does contain any other power sources, than suppliers and any other power sinks, than demands, and
- *price neutral*, meaning that the price must only be dependent of supplier prices and transmission costs.

Transparency. The price computation should always be transparent for suppliers and demands: This means on one hand, that the price should be *deterministic*, i.e. strictly dependent of suppliers and demands, and on the other hand, that it should show a *low sensitivity* towards single components of the system. In other words, small changes in the input state of the system, should only yield small changes in the outcome.

2.2.2 Solution Space

There are many possibilities, on which grounds an energy price can be computed: We could for example envision auction based models, where suppliers compete in order to supply the lowest selling price, or where demands compete to offer the highest buying price, models, where suppliers and demands interact and close contracts individually, or even models, where the price is computed using inputs which are external from the grid.

In order to be able to come up with a pricing-scheme that fulfills the requirements posed in the previous section, we will first define a useful solution space, which will allow later allow us to formulate different solutions to our problem. Our solution space has the following dimensions:

Interaction. This dimension comprises the different possibilities of how components of the system interact with each other. We distinguish two different possibilities:

1. *Direct.* Suppliers and demands directly interact with each other. If e.g. a demand wants to buy energy from a determined supplier, it needs to contact that supplier and they can agree on a price.
2. *Indirect.* Suppliers and demands only interact with some abstract system, which can be central or distributed, which, based on the information gathered from them, takes care of computing the energy price at each location of the grid.

Offers. This dimension contains the possible solutions to the question, who can make a price-offer, which can directly influence the price. We consider the following possibilities:

1. *Only Suppliers.* Suppliers can make a price offer, demands can accept it or refuse it.
2. *Only Demands.* Demands make price offer, suppliers can decide whether to supply for that price or not.
3. *Both.* Suppliers and demands make a price offer, the price is computed considering both.

Trade Object. The price of electrical energy is influenced by the form in which it is traded:

1. *Past Deliveries.* The price is influenced by how much energy has been delivered in the past.
2. *Present Deliveries.* The price at time t is only influenced by currently ongoing power deliveries.
3. *Future Deliveries.* Suppliers and demands can trade over long-term energy deliveries, which are taken into account when computing the price.

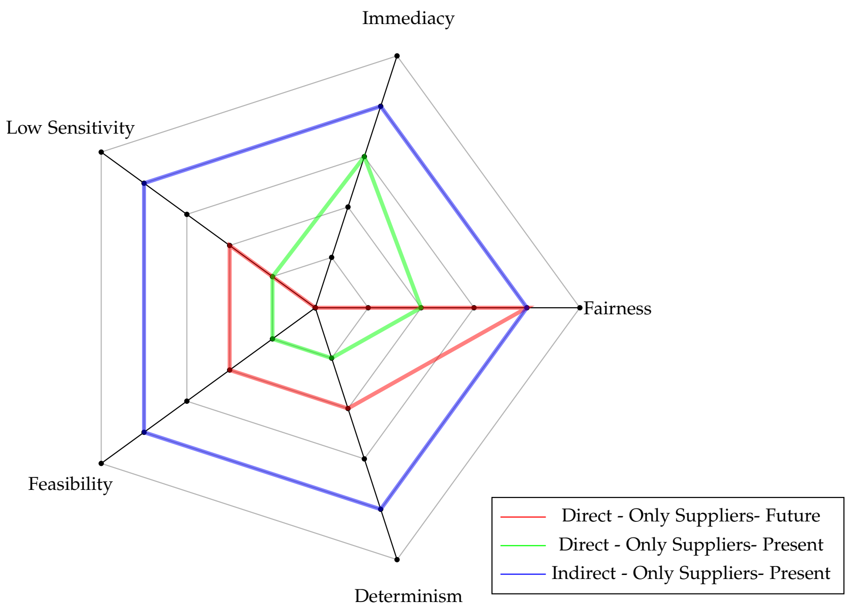


Figure 2.2: Evaluation of the proposed solution based on the defined requirements.

2.3 Experimental Exploration of the Solution Space

Before defining a detailed solution model, we will now briefly consider a few possible solution-approaches and evaluate them using our requirements.

2.3.1 Direct Future Deliveries Market Based on Supplier Offers

For this solution approach, we propose the following system: Suppliers can make a price offer for a certain amount of energy, which they guarantee they can supply over a certain amount of time. Consumers can see each price offer, along with the transmission costs that they would have to pay, to transport the energy from the supplier making that offer to their location.

If a consumer decides to accept an offer from a certain supplier, they close a contract, in which the they agree on the amount of energy the supplier commits to provide in a certain time-period. The price at the location of the demand can be computed as the sum of the price offered by the supplier and the transportation costs from the supplier to the demand.

Analysis. This solution approach is loosely modeled after the future-delivery markets, that we can typically find in wholesale-markets. It guarantees fair conditions to consumers and suppliers, who both have to voluntarily close a contract.

However, since the trade object of such a contract is based on future energy deliveries, which, especially for suppliers that rely on renewable energy sources, are often very hard to predict, it shows a very low immediacy. This implies, that under certain conditions suppliers might not be able to maintain their part of the contract, which means that the feasibility of the system is not guaranteed.

2.3.2 Direct Present Deliveries Market Based on Supplier Offers

We can try to improve the immediacy of the previous solution approach by making the price dependent on present, instead of future energy deliveries. In other words, suppliers can now only make price offers that are valid for infinitely small time intervals $\Delta_t \rightarrow 0$. Consumers are free to individually accept each offer, to cover their demand.

Analysis. This approach yields a higher immediacy than the previous one, since the probability that a supplier can not deliver the energy promised in the contract due to external reasons decreases with $\Delta_t \rightarrow 0$.

However, since suppliers and customers now need to close an increased number of contracts in short time-intervals. Even if no supplier and no customer in the system changes its input state, there is no guarantee that the same contracts will be closed in subsequent intervals. As a consequence, the price for a certain customer is prone to frequent and unpredictable changes. This violates our requirement of transparency, were we postulated that the price must be deterministic and have a low sensitivity.

2.3.3 Indirect Present Deliveries Market Based on Supplier Offers

We try to increase the determinism and lower sensitivity of the previous approach by substituting direct interaction between suppliers and customers in the previous approach, with an indirect interaction through a central system: Suppliers and demands continuously update the system with their own state, which in return computes the energy prices at each location of the grid.

Analysis. As opposed to the previous solution approach, where the prices at single locations of the grid where prone to unpredictable fluctuations for constant input states, computing all prices centrally allows us to ensure the determinism of the price. Since the system collects the whole state of the network, it is also easier to make sure that it produces feasible solutions and guarantees a fair price.

2.4 Supply and Demand Price

In the previous section, we evaluated different possibilities of defining a meaningful energy price for the electrical grid.

In this section we will use the insights we gained, to define a real-time supply and demand price, assuming a competitive market. First we will give an informal description of how such a price can be defined in an ideal electricity market, where a demand can choose exactly from which supplier to buy energy. After having described the behavior of our price under this assumption, we will show how such a price can be computed for an electrical grid, where electrical energy can not be labeled and demands can not choose whose energy to consume.

Price in an Ideal Market. A free market is a market in which the prices are based on the competition between private vendors and controlled supply and demand. If we model our pricing scheme according to this principle, we can assume that, given the choice between different suppliers, a demand will always choose to buy power from the most competitive supplier, which is the supplier who, including transportation costs, can cover the power-demand at the lowest price. As a consequence, the price for that demand is given by the sum of the price offered by the most competitive supplier and the transportation costs from said supplier to the demand.

If the most competitive supplier can supply more power than its needed by the demand, it can always sell the remaining power to other demands, given that its offer is more competitive for them than others.

If however the most competitive supplier does not have enough resources to cover the whole power-need of a demand, the latter will buy as much power as possible from that supplier, and cover the rest of its power-demand using power from the second most competitive supplier, and so forth, until its whole power-demand is covered. In this case, the unit-price payed by that demand is composed by the average price of all those suppliers, weighted by the amount of power each supplier is selling to the demand.

One can see how a demand will has to resort to less and less competitive suppliers, to cover an increasing power-demand, which results in a higher price. On the other side, with decreasing power demand, suppliers have to make more and more competitive offers to sell their energy, resulting in a lower price. This is essentially the behavior which in microeconomics is described by the principle of *supply and demand* and which we will now apply to our model of the electrical grid, to obtain our energy-price.

Definition (Price at Demand). Given that a demand i buys a total power amount of d_i from a set of suppliers $j \in S_i$, the price at demand i can be computed as:

$$p_i = \frac{\sum_{\forall j \in S_i} s_{j,i} * (p_j + c_{j,i})}{d_i}, \tag{2.1}$$

where $s_{j,i}$ is the amount of power that supplier j is selling to demand i, p_j is the price offered by supplier j and $c_{j,i}$ is the sum of all transmission costs between supplier j and demand i.

Application to the Real Grid. As we have already mentioned in the beginning of this section, in a real electrical grid demands can typically not decide from which suppliers they physically buy power. This is due to the fact, that various suppliers and demands share the same transmission lines, and once fed into a transmission line, power is distributed according to the laws of physics and can not be tracked.

Therefore, we need to compute a virtual distribution of the power flows in our grid, which reflects the behavior assumed for an ideal market and is independent from the physical power flows in the grid. This can be achieved using the data-layer of the smart-grid infrastructure, which provides us with information about the topology of the grid and the current state of its members.

Power Distribution. Since in our solution space exploration in Section 2.3 we concluded, that rather than using direct system, where suppliers and demands directly interact with each other, we want to compute our prices using an indirect system, we need to define an objective for which to compute our distribution of power flows.

Based on the assumptions that we made about an ideal free market, where we claimed that each demand will always choose the most competitive offers, we compute our power-flow distribution using a global objective, which minimizes the sum of the prices of every demand in the system. In other words, using the definition for the price at a demand with id i, which we introduced in Equation 2.1, we need to find a power flow that satisfies:

$$\min \sum_{\forall i \in D} p_i, \tag{2.2}$$

where D is a set containing all demands in the electrical grid.

How we can find such a distribution, is not clear yet. In the next chapter however, we will formalize our model of the electrical grid in a way, that will allows us to find such a power flow. We will then use it, to compute our energy prices, at each grid location.

Chapter 3

Solution

In this chapter, we will propose a scheme to compute the momentary supply and demand price, we introduced in Section 2.4. We will do this by first introducing an mathematical graph-model of the electrical grid and then by defining an algorithm which works in three stages:

1. Solving a minimal cost flow problem,
2. Performing a topological sort,
3. Computing the price at each node.

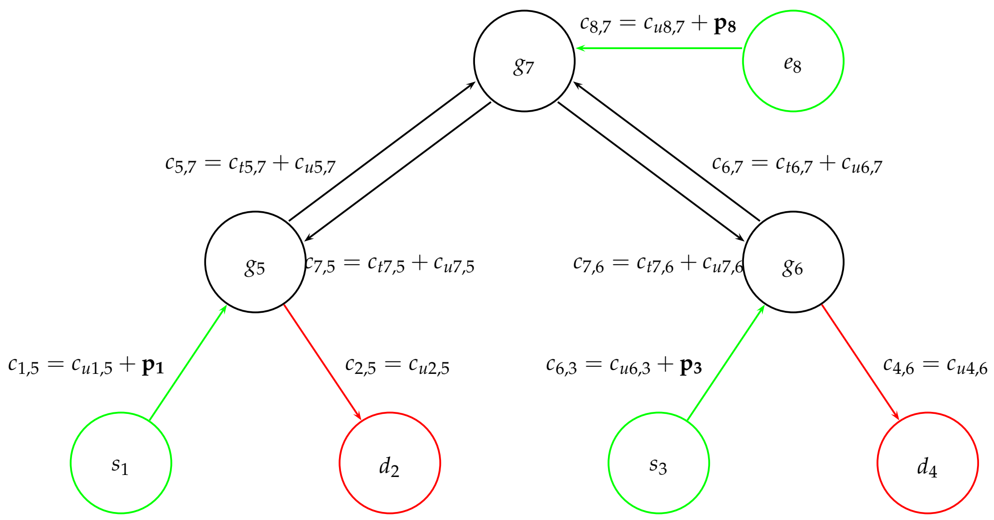
3.1 Graph Model of the Electrical Grid

After having introduced an informal model of the electrical grid in Section 2.1, consisting of suppliers, demands, sub-grids and transmission lines associated with a transmission cost, we will now formalize this model in a way, that is suitable for computing the momentary supply and demand price.

In systematic studies of the electrical grid, it is common to represent the grid as mathematical graph [11] [12] [13]. In particular, we will represent the electrical grid as a weighted directed graph, which we characterize using the following general definitions:

Definition (Weighted Directed Graph) A weighted directed graph G is a triplet of sets $G(N, E, C)$, where V is the set of nodes, E is the set of edges and C is the set of costs.

An *edge* $e_{i,j}$ is an ordered pair of nodes (i, j), where $i, j \in N$. If $(i, j) \in E$, then i and j are said to be neighboring or adjacent and i is called the source-endpoint of the edge, while j is called the target-endpoint.



$c_{8,7} = c_{u8,7} + \mathbf{p_8}$
g_7
e_8
$c_{5,7} = c_{t5,7} + c_{u5,7}$
$c_{6,7} = c_{t6,7} + c_{u6,7}$
g_5
$c_{7,5} = c_{t7,5} + c_{u7,5}$
$c_{7,6} = c_{t7,6} + c_{u7,6}$
g_6
$c_{1,5} = c_{u1,5} + \mathbf{p_1}$
$c_{2,5} = c_{u2,5}$
$c_{6,3} = c_{u6,3} + \mathbf{p_3}$
$c_{4,6} = c_{u4,6}$
s_1
d_2
s_3
d_4

Figure 3.1: Schematic depiction of a grid composed of three sub-grids (g_x) operating at different voltage-levels. The costs are composed by transformation costs ($c_{ti,j}$), grid-usage costs ($c_{ui,j}$) and supply prices p_i.

For a node $i \in N$ the number of source-endpoints attached to i is called the *in-degree* of i and is denoted by $deg_{in}(i)$. The number of target-endpoints attached to i is called the *out-degree* and is denoted by $deg_{out}(i)$.

The **cost** $c_{i,j}$ is a positive integer associated with the edge $e_{i,j} \in E$.

Definition (Path) A path of G is a subgraph P of the form :

$$V(P) = \{0, 1, \dots, l\},$$

$$E(P) = \{e_{0,1}, e_{1,2}, \dots, e_{l-1,l}\},$$

$$C(P) = \{c_{0,1}, c_{1,2}, \dots, c_{l-1,l}\},$$

such that $V(P) \subseteq V$, $E(P) \subseteq E$ and $C(P) \subseteq C$.

The *cost of path P* is defined as the sum of all costs found in the path:

$$c(P) = \sum_{\forall c_{i,j} \in C(P)} c_{i,j}. \tag{3.1}$$

Mapping of our Model to the Graph Model. In order to fully represent our model of the electrical grid, composed of sub-grids, suppliers, demands and transmission-costs, using the definitions above, we need to define which components of the model are contained in which set of the graph, along with the special properties they may have.

Nodes. The components of our electrical grid model are represented by nodes in the graph-model. More specifically, we distinguish three sub-sets of nodes:

1. *Supplier Nodes.* The set of supplier nodes $S \subset N$ contains all suppliers of the electrical grid. At each time t, each supplier node i is associated with two integers: The current *power supply* $s_i(t)$ [W] and its current *price offer* $p_i(t)$ [$].
2. *Demand Nodes.* The set of demand nodes $D \subset N$ contains all demands of the electrical grid. At each time t, each demand $i \in D$ is associated with one integer $d_i(t)$ [W], representing its current power demand.
3. *Sub-Grid Nodes.* The set of sub-grid nodes $G \subset N$, contains all sub-grid nodes.

Edges. The transmission line that interconnect the sub-grids and that connect suppliers and demands to a specific sub-grid are represented by directed edges. The direction of an edge represents the direction in which electrical power can be transmitted: Since in our model we assume, that suppliers can only supply energy, but not receive energy and are connected to exactly one sub-grid, they have *exactly one* out-going edge. Analogously, demands have *exactly one* incoming edge.

Costs. The costs associated with the edges, contain the transportation (grid-usage c_u and transformation c_t) costs introduced in Section 2.1, as well as the prices (p) offered by the suppliers. As displayed in Figure 3.1, the cost $[\frac{\$}{W}]$ between a supplier and a sub-grid node, is therefore defined as the sum of the grid-usage cost and the supply-price.

Graph Model. Finally, we can use this mapping from our model-components to mathematical graph components to introduce the following graph representation of the electrical grid:

Definition (Electrical Grid Graph) The electrical grid graph is a weighted directed graph $\mathcal{G}(N, E, C)$ such that each element $i \in N$ is either a supplier ($i \in S \subset N$), a demand ($i \in D \subset N$) or a sub-grid ($i \in G \subset N$). There is an edge $e_{i,j} = (i, j) \in E$, if there is a physical transmission line that connects the components represented by i and j and a cost $c_{i,j} \in C$ for each edge $e_{i,j}$.

The direction of an edge represents the direction in which electrical power can be transmitted. Therefore, supplier nodes $i \in S$ can only have exactly one outgoing edge $e_{i,j}$, while demand nodes $i \in D$ can only have exactly one incoming edge $e_{j,i}$

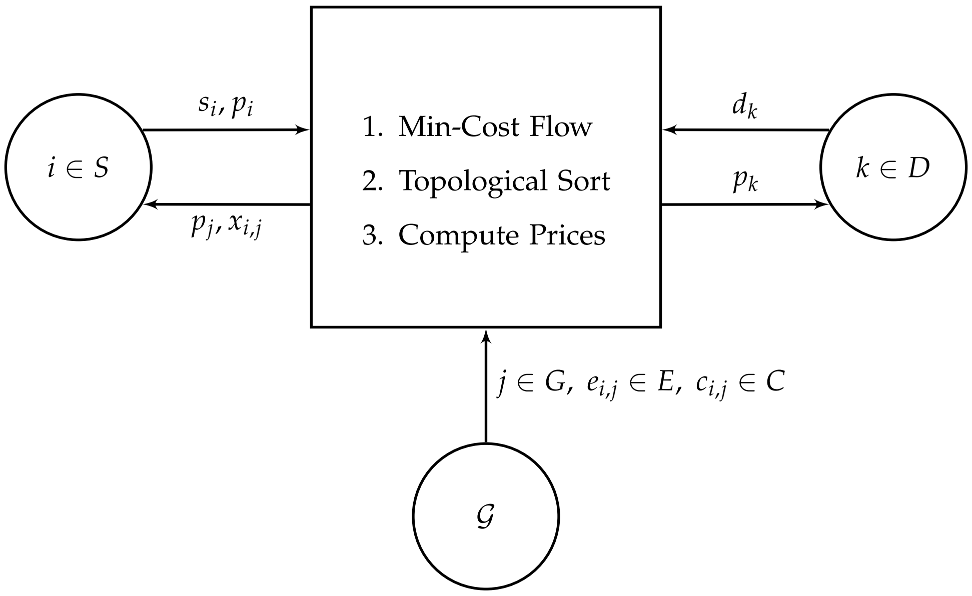


Figure 3.2: Inputs and outputs of the pricing algorithm.

The cost $c_{i,j}$ associated with edge $e_{i,j}$ is composed of the sum of the grid-usage cost $c_{ui,j}$ the transformation cost $c_{ti,j}$ and, if $i \in S$ of the price offer p_i of supplier i.

3.2 Input and Output

As can be seen in Figure 3.2, algorithm has the following input- and output parameters:

Input. The input for the algorithm is composed of:

- The electrical grid graph $\mathcal{G}(N, E, C)$ containing all nodes, all edges and all transmission costs.
- The state of each supplier-node $i \in S$ consisting of the current available power supply s_i and the current price offer p_i.
- The state of each demand-node, i.e. its current power demand d_k.

Output. The system has two different outputs for suppliers and demands:

- The output for a supplier $i \in S$ is composed by the price p_j at the grid-node $j \in G$, it is connected to, as well as the amount of power $x_{i,j}$ it is currently selling.
- The output for a demand $k \in D$ is the energy price p_k it is currently paying.

3.3 Minimal Cost Flow

In Section 2.4 we defined our momentary supply and demand price based on the assumption of an ideal and free market, where customers always choose to buy energy at the lowest possible price available at their location. Furthermore, we added a constraint to our model saying, that customers that are "closer" to a specific supplier than others, meaning that there is less transportation costs between them, get priority over customers that are *farer*.

This distribution of power flows along the edges corresponds to the minimal cost flow along the edges of our electrical grid graph. We can obtain this minimal cost flow, by solving a linear program (LP), with the following decision variables, objective function and set of constraints:

Decision Variables. The decision variable of this problem are the power flows $x_{i,j}$ [W] on all edges $(i,j) \in E$.

Objective Function. The objective is to minimize the total transportation cost of delivering power through the network. The cost of transporting a power flow of $x_{i,j}$ [W] along the edge (i,j) is given by the cost $c_{i,j}(x_{i,j})$, which is assumed to be linear, which means that $c_{i,j}(x_{i,j}) = c_{i,j}x$.

Therefore, the total cost in the network is given by $\Sigma_{(i,j)\in E} c_{i,j} x_{i,j}$, which leads to the objective function:

$$z = \min_{x_{i,j}} \sum_{(i,j)\in E} c_{i,j} x_{i,j} \tag{3.2}$$

Constraints. The system has the following constraints:

1. The flow out of a supply node must be equal or less than the supply. Naturally, in the physical network the flow out of a supply node is always equal to the supply, which we defined as the power, which is currently being fed into the system. However, we do not consider surplus power in our price model and therefore we can write the following constraint:

$$\sum_{(i,j)\in E} x_{i,j} \leq s_i, i \in S \tag{3.3}$$

2. Like in the physical network the power-flow must be conserved in each sub-grid, meaning, that the flow entering a sub-grid, must also exit it:

$$\sum_{(i,j)\in E} x_{i,j} = \sum_{(j,k)\in E} x_{j,k}, j \in G \tag{3.4}$$

3. The power flow into a demand node is equal to the power requested:

$$\sum_{(i,j)\in E} x_{i,j} = d_j, j \in D \tag{3.5}$$

4. The power flows can not be negative:

$$x_{i,j} \geq 0, \;\; \forall (i,j) \in E \tag{3.6}$$

Solving this linear program will yield a minimum cost flow f_{min}, which can now be used to compute the momentary demand and supply price for each demand node.

3.4 Price Propagation and Computation at each Node

Solving the LP introduced in the previous section for our model of the electrical grid, we obtain the power flows on each edge that we assume for an ideal and free market. This flows can now be used to compute the effective energy price at each node, starting from the supplier nodes and following the power flows in topological order until reaching the demand nodes.

3.4.1 Price at Supply Nodes

The price at a supply s_i is equal to the price p_i offered by the node and therefore does not need to be computed.

3.4.2 Price at Sub-Grid and Demand Nodes

Since all edges in our graph have positive costs and there are no negative flows, we can prove that an optimal minimum cost flow does not contain cycles and can therefore, by removing all edges with 0-flow, be transformed into a directed acyclic graph (DAG), where each edge has two weights, namely the cost and the power flow.

This DAG can now be sorted into a topological order, following the flows from the supply nodes to the demand nodes. Given the set of incoming edges at node i, $E_{in,i}$, we now define the price at node i as:

$$p_i = \frac{\sum_{\forall (j,i) \in E_{in,i}} x_{j,i} * (c_{j,i} + p_j)}{\sum_{\forall (j,i) \in E_{in,i}} x_{j,i}}, \tag{3.7}$$

which at each node i corresponds to the weighted average of the energy quantity supplied to i from different sources.

3.5 Uniqueness of the Solution

In order to meet the determinism-requirement posed in Section 2.2.1, the solution of our price-algorithm must be unique for a given set of inputs. The LP introduced in Section 3.3, which computes the global minimum of the transportation costs in our graph model of the electrical grid, does not guarantee a unique distribution of power flows to achieve said minimum. Since our price computation depends on the power flows computed by the LP, we need to analyze, how they impact the uniqueness of our solution.

Example 1. In order to perform this analysis, we will first consider the example network depicted in Figure 3.3.

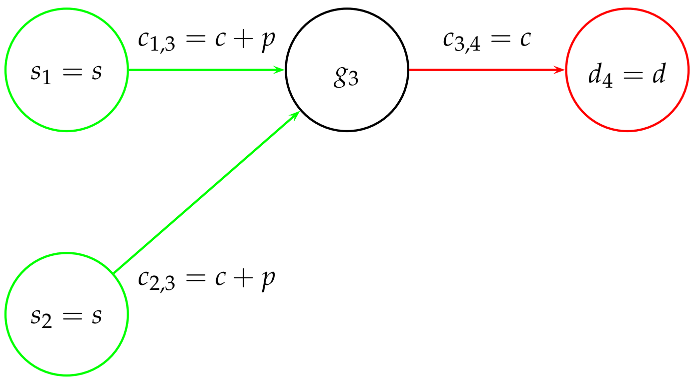


Figure 3.3: Example illustrating a scenario with no unique solution.

In this example, we have two suppliers, s_1 and s_2, which make an identical price offer p, and one demand node d_4. The power supply of each supplier (s) is larger than the demand $d_4 = d$. Each edge has is associated with the same cost c. This leads to the following LP:

$$\begin{array}{llllllll}
\text{min} & \text{(c+p) } x_{1,3} & + & \text{(c+p) } x_{2,3} & + & \text{c } x_{3,4} & & \\
\text{s.t.} & x_{1,3} & & & & & \leq & s \\
 & & & x_{2,3} & & & \leq & s \\
 & -x_{1,3} & + & -x_{2,3} & + & x_{3,4} & = & 0 \\
 & & & & & x_{3,4} & = & d,
\end{array}$$

which has the solution $z = d(c + p + c)$, which can be obtained by all flow-distributions of form:

$$\mathbf{x} = \begin{pmatrix} a \\ d - a \end{pmatrix}, \quad \text{for } a \in [0, d],$$

which however using Equation 3.7 all lead to the same price at sub-grid node g_3:

$$p_3 = \frac{a*(c+p)+(d-a)(c+p)}{a+d-a} = \frac{(c+p)(a+d-a)}{a+d-a} = c+p. \qquad (3.8)$$

Therefore, in this case, the computed price is unique regardless of the uniqueness of the flow distribution obtained by solving the minimal cost-flow problem.

Example 2. We will now consider the slightly more complex network represented in Figure 3.4:

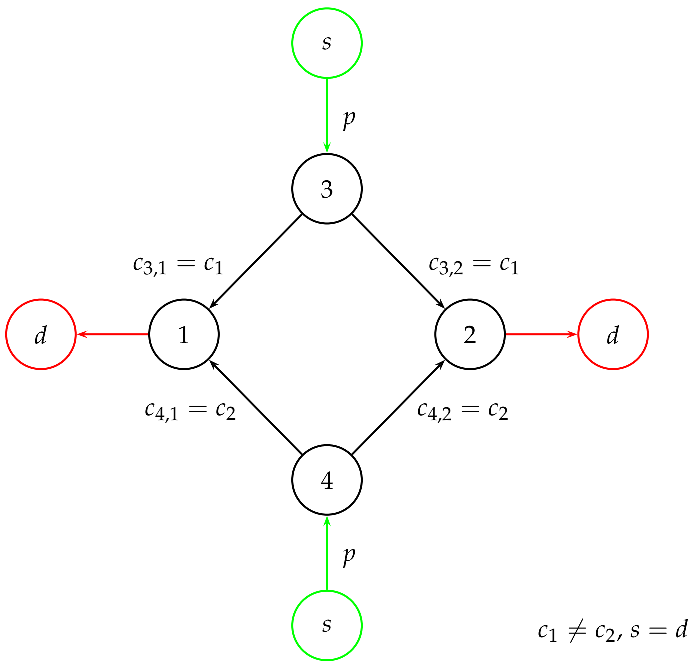


Figure 3.4: Example illustrating a scenario with two suppliers, two demands with demand, and four grid-nodes. The transportation costs between grid-nodes and supplier- or demand-nodes are 0, the supply s of the supplier nodes is equal to the demand d of the demand nodes and $c_1 \neq c_2$.

Since the total power-supply $2s$ in the system is equal to the total power-demand $2d$, we can easily determine that the LP has exactly one solution:

$$z = \min \mathbf{c}^\mathsf{T}\mathbf{x} = s(2p + c_1 + c_2), \qquad (3.9)$$

where $\mathbf{c}$ is the cost-vector $\mathbf{c} = \begin{pmatrix} c_{3,1} & c_{3,2} & c_{4,1} & c_{4,2} \end{pmatrix}^\mathsf{T}$ and $\mathbf{x}$ is the flow-vector $\mathbf{x} = \begin{pmatrix} x_{3,1} & x_{3,2} & x_{4,1} & x_{4,2} \end{pmatrix}^\mathsf{T}$.

For the sake of simplicity, we will now assume w.l.o.g. that $p = 0$ and that therefore $z = s(c_1 + c_2)$. Similarly to the previous example, there are many minimum-cost flows, that achieve this minimum and which have the form:

$$\mathbf{x_{a,b}} = \begin{pmatrix} a \\ s-a \\ b \\ s-b \end{pmatrix}, \quad \text{for } a, b \in [0, s],$$

which this time however do yield different prices at the nodes 1 and 2. Lets for example consider the two corner cases $\mathbf{x_{0,0}} = \begin{pmatrix} 0 & s & 0 & s \end{pmatrix}$ and $\mathbf{x_{s,s}} = \begin{pmatrix} s & 0 & s & 0 \end{pmatrix}$ which, as one can easily verify, are both minimum cost flows and lead to the following prices in nodes 1 and 2:

	p_1	p_2
$\mathbf{x_{0,0}}$	c_2	c_1
$\mathbf{x_{s,s}}$	c_1	c_2

These results are clearly infringe our determinism requirement and could lead to undesired run-time behavior. Therefore we propose two solutions to this problem: The first one, is to change the objective function in a way, that always yields unique solutions, the second one is to change the graph topology, in order to avoid ambiguous situations:

3.5.1 Add Quadratic Perturbation

As it has been shown in [14] it is possible to solve a linear program, by adding quadratic perturbation and solving the resulting quadratic problem in its dual variable space. Furthermore it has been shown in [15] that if the original linear program has an optimal solution, the solution of the corresponding quadratic problem, will be unique.

Concretely, given a linear optimization problem of form:

$$\min \mathbf{c}^\mathsf{T}\mathbf{x}, \quad s.t. \ \mathbf{A}\mathbf{x} \geq \mathbf{b}, \tag{3.10}$$

adding a quadratic perturbation of form $\varepsilon \mathbf{x}^\mathsf{T}\mathbf{x}$, will lead to the quadratic problem:

$$\min \mathbf{c}^\mathsf{T}\mathbf{x} + \varepsilon \mathbf{x}^\mathsf{T}\mathbf{x}, \quad s.t. \ \mathbf{A}\mathbf{x} \geq \mathbf{b}, \tag{3.11}$$

which for small enough ε solves the original problem, by picking the optimal solution with the smallest $\mathbf{l_2}$-norm. Although this approach would solve the problem of uniqueness, it requires solving a quadratical program, which is significantly less efficient than a linear program. Therefore, we will now look at another solution, that does not require us to change the original LP-formulation.

3.5.2 Randomize the Edge-Costs

Another approach to make the solution of the minimal-cost flow problem unique, is to avoid ambiguous situations by ensuring, that no two edges in the graph are associated with the same cost.

This can be achieved by *once* substituting each cost $c_{i,j} \in C$ with the randomized cost:

$$\tilde{c}_{i,j} = c_{i,j} + \varepsilon r_{i,j}, \quad \text{for } \varepsilon \ll c_{i,j}, \;\; r \sim U([0,1]), \tag{3.12}$$

where ε is very small and r is selected uniformly at random, when solving the minimal-cost flow problem. After that, the prices can still be computed using the original, not randomized, costs, which means that for small enough ε, the prices will still be exact.

This randomization of the linear coefficients of the objective function, minimizes the probability of it being parallel to one of the edges of the polytope, which constitutes the feasible region given by the constraints. Therefore, the solution to the problem must lie in a vertex of the solution, i.e. at a corner of the feasible region.

Although this solution slightly impacts the fairness of our algorithm, since some nodes might consistently get worse price offer than others, who would have access to the same price, we reckon that the impact will be relatively limited in real-life networks. On the other hand, it allows us to keep the results consistent and deterministic, without resolving to the use of time-intensive quadratic programs.

3.6 Model Adjustments

The mathematical model does not yet cover the solution of two corner cases: The first one is the case, where there is not enough available power-supply to cover the total demand, the second one is the case, when there is no power-demand at a certain grid-node.

Infinite Supplier. The first problem was already addressed in Section 2.1.2, where we introduced a special supplier, with an infinite power-supply and a determined price-offer, that we called the *energy exchange*. In our graph model, the energy exchange is simply a supplier $i \in S$, associated with a very high power supply s_i.

Infinitesimal Demand. If we recall Equation 3.7, given the minimum cost flow, we defined the price at each node i as follows:

$$p(i) = \frac{\sum\limits_{\forall(j,i)\ \in\ E_{in,i}} x_{j,i} * (c_{j,i} + p(j))}{\sum\limits_{\forall(j,i)\ \in\ E_{in,i}} x_{j,i}}. \tag{3.7}$$

This definition however, is not applicable to nodes that have no incoming power-flow. This is a problem, since we want to compute the energy price at each location of the grid, regardless of the current energy demand. We solve this problem, by assuming a node with an infinitely small demand $\delta_0 \to 0$ at each sub-grid node if our graph. The idea is that while δ_0 guarantees an incoming flow at each sub-grid node, it is small enough not to influence the price. Following the same idea, also demand nodes that only want to query the price at a certain location of the network, without actually consuming power, will be represented with a demand $d_i = \delta_0$.

Chapter 4

Functional Validation

In this chapter we will perform a series of experiments and derive theoretical bounds to validate the behavior of our algorithm: First we will analyze the sensitivity it has towards single suppliers and then we will look at different strategies suppliers can employ in order to maximize their profits. Finally, we will perform an experiment that shows how the price evolves in a dynamic system, where suppliers and demands follow a determined strategy.

4.1 Experimental Setup

4.1.1 Network Topology

Since the network topology has a significant impact on the price computation performed by our model, it is important to perform experiments on a realistic network structure.

Although some test systems, like the historical IEEE model systems, are available[1] and widely used in the literature, we prefer to use statistically similar random networks, since the model systems are limited in size and, being from the 1960s, relatively dated.

In 1998, Watts and Strogatz [13] first proposed an algorithm to statistically model the power grid as a *small network* graph, in which most nodes are not neighbors of one another, but can still be reached by every other node via a small number of hops. More specifically, the average path length $\langle l \rangle$ is proportional to the logarithm of the number N of nodes in the network:

$$\langle l \rangle \propto \ln N. \tag{4.1}$$

[1]Washington University's *Power Systems Test Archive Site*: http://www.ee.washington.edu/research/pstca/

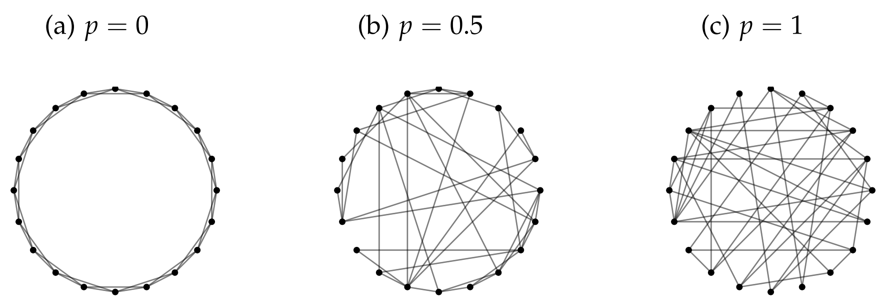


Figure 4.1: Small-World networks generated with the Watts-Strogatz method for N = 20, k = 4 and for different rewiring probabilities p. While $p = 0$ corresponds to a regular ring lattice, $p = 1$ yields a completely random network.

Given the desired number of nodes N and an average nodal degree of $\langle k \rangle$, satisfying

$$\langle k \rangle \ll N \ll e^{\langle k \rangle}, \tag{4.2}$$

the first step for the algorithm is to construct a regular ring lattice, in which every node is connected with $\frac{\langle k \rangle}{2}$ nodes on each side. Then it iterates through each node and, with probability p, re-connects edges on the right side of a node to another node selected uniformly at random. The parameter p allows to tune the graph between regularity ($p = 0$) and disorder ($p = 1$). In a further effort to generate random power-grid topologies, with statistical properties similar to real networks, Wang *et al.* [12] analyzed the statistical properties of 6 real-world networks (Table 4.1). They came to the conclusion that, while real topologies do manifest small-world properties, they are connected with a low average nodal degree $\langle k \rangle = 2 \sim 5$, which *does not scale* with the network size. This means, that the Watts Strogatz model, which requires condition (4.2), can only be applied to networks of limited size. They validated this claim by performing experiments that showed that for $\langle k \rangle = 2 \sim 3$ the network size should be limited to 30 and for $\langle k \rangle = 4 \sim 5$ to 300 nodes.

As a consequence, they proposed an alternative scheme to generate more realistic network topologies: First a number of distinct *small-world subnetworks*, whose size is limited by (4.2), is created, using a method similar to the one proposed by Watts and Strogatz. In a second step, each subnetwork is connected to other subnetworks, by randomly rewiring a number around $\langle k \rangle$ edges.

For our simulations, we will use the method proposed by Wattson and Strogatz, whenever the condition (4.2) holds and the updated scheme from [12], for larger networks.

	(N, m)	$\langle l \rangle$	$\langle k \rangle$
IEEE-30	(30,41)	3.31	2.73
IEEE-57	(57, 78)	4.95	2.74
IEEE-118	(118, 179)	6.31	3.03
IEEE-300	(300, 409)	9.94	2.73
NYSIO	(2935, 6567)	16.43	4.47
WSCC	(4941, 6594)	(18.70)	2.67

Table 4.1: Average path length and nodal degree for 6 real world networks.

4.1.2 Costs

Since in our model the final price is composed by the energy production cost as well as grid-costs, the ratio between those two components plays an important role for the overall network behavior.

We use data from a report of the Swiss Association of Electricity Companies (VSE)[2] to determine that in the years from 2010-2014 the electricity price paid by an average household was composed of 44.3% grid costs, 41.14% production costs and 14.56% taxation. Excluding taxation from our model we can conclude, that the cost-ratio ρ_c between production-costs c_p and grid-costs c_g is given by

$$\rho_c = \frac{c_p}{c_g} = \frac{41.14}{44.3} = 1.077 \approx 1. \tag{4.3}$$

As the total grid-costs for shipping electricity on a path from a supplier to a consumer are given by the sum of all grid usage and transformation costs along that path, given a network with an average path length of $\langle l \rangle$, the average cost $\langle c_{i,j} \rangle$ added by transporting 1kWh through edge (i, j) is given by:

$$\langle c_{i,j} \rangle = \frac{\langle c_p \rangle * \rho_c}{\langle l \rangle}, \tag{4.4}$$

where $\langle c_p \rangle$ is the average energy production cost.

4.2 Global Influence of a Single Supplier

In order to assess the quality of our solution approach, it is important to analyze the sensitivity our pricing algorithm has towards one single supplier: We want to understand the impact small changes in the price offer or the

[2]Verband Schweizerischer Elektrizitätsunternehmen, full report: http://www.strom.ch/fileadmin/user_upload/Dokumente_Bilder_neu/010_Downloads/Stromgrafiken/Strompreise/06_Komponenten_Strompreis_2015_d.pdf

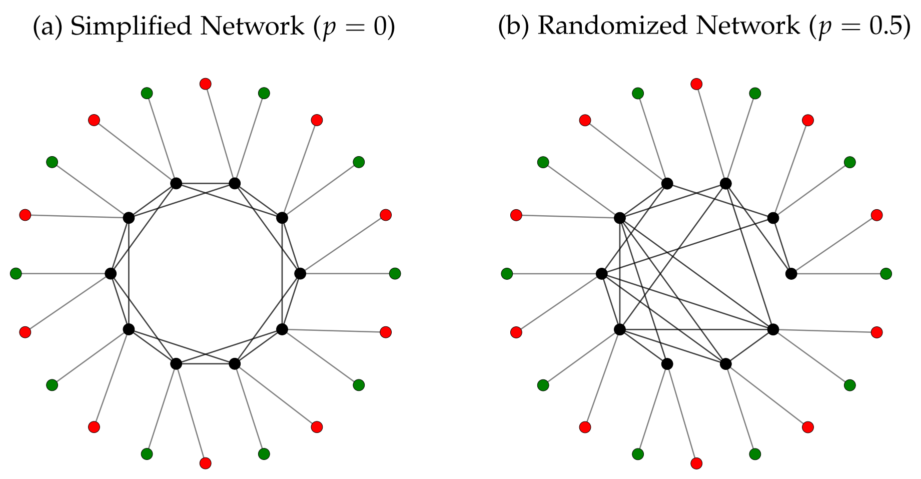

(a) Simplified Network ($p = 0$)
(b) Randomized Network ($p = 0.5$)

Figure 4.2: Networks used for deriving bounds with $N = 10$ and $k = 4$ and rewiring-probability p. A simplified network depicted in 4.2a will be used to derive the theoretical bounds, which will then be validated by simulations performed on randomized networks like in the one depicted in 4.2b.

amount of available power of a supplier have on the price in other nodes of the network. We will use this information to verify, that the system behaves in a stable and deterministic manner and yields a meaningful price at all times: First by verifying, that the size of the neighborhood influenced by a change in one single supplier s_i is bounded by the transportation costs in the network, which means that even a very large supplier is only able to influence a local portion of the network, and then by showing, that the average price in the network is directly dependent on the number of influenced nodes.

We will do this, by first deriving those bounds theoretically, making some simplifying assumptions about the electrical network and then by comparing those bounds with the results of simulations performed on randomized networks.

The simplified network, for which we compute the theoretical bounds, is composed of N grid nodes $g_i \in G$, each of which is connected to exactly one supplier $s_i \in S$ and one demand node $d_i \in D$. Each supplier can provide an infinite amount of power and initially has the same same price p_{init}, while each demand node has the same demand. The grid has a constant nodal degree of k, which means that each grid node $g_i \in G$ is connected to exactly k other grid nodes. Each edge in the network has the same grid cost c_g. An example of such a network with $N = 10$ and $k = 4$ is displayed in Figure 4.2a.

The networks on which we run the simulations is randomized using the method proposed by Watts and Strogatz in [13]. Again, each grid node is connected to exactly one supplier, whose infinite supply will be replaced by a very high number, and one demand node, with a very low demand. Figure 4.2b shows the randomized network derived from the theoretical network in 4.2a, by rewiring its edges with probability $p = 0.5$.

4.2.1 Number of Grid Nodes Influenced by Price Reduction δ_0

In this section we assume the previously introduced simplified network, with a very large number of grid nodes N. Initially, each supplier $s_i \in S$ offers the same price $p_i = p_{init}$: Since each grid node is connected to exactly one supplier and one demand and the transportation costs are equal on every edge, the energy price is the same on every grid node. We will now derive a theoretical bound for how many nodes change, if one supplier s_i reduces the price by an amount δ_0.

Theoretical Bound

Since N is very large, we assume that each node in the influence range of s_i has its own distinct neighborhood (i.e. if two grid nodes g_2 and g_3 are neighbors of g_1, they are not neighbors of each other). This means, that the neighborhood of s_i can be treated as a tree.

Since in the initial stable state, each grid node is connected to exactly one supplier which offers the price p_{init}, the price in each grid node initially is $p_{init} + c_g$. If however s_i was the only supplier in the network (Figure 4.3), the price in each grid node would increase proportionally to its distance to s_i. In other words, if the distance between a g_j and the s_i is given by $d_{i,j}$, the price in the node g_j is exactly:

$$p_j = p_{init} + d_{i,j} * c_g. \tag{4.5}$$

This trivially implies, that if s_i reduces its price by δ_0, the price in grid node j is given by:

$$p_j = (p_{init} - \delta_0) + (d_{i,j} * c_g). \tag{4.6}$$

Now we again assume that each grid node is connected to exactly one supplier: Since the algorithm minimizes the total costs in the network, given a price reduction of δ_0 in node si, the price in grid node g_j will only change if $p_j < p_{init} + c_g$, which is equivalent to $\delta_0 > -c_g * (1 - d_{i,j})$ and means that the price reduction of a single supplier s_i must at least be:

$$\delta_0 > d_{i,j} * c_g + \varepsilon, \varepsilon > 0, \tag{4.7}$$

to influence a grid node at distance $d_{i,j}$.

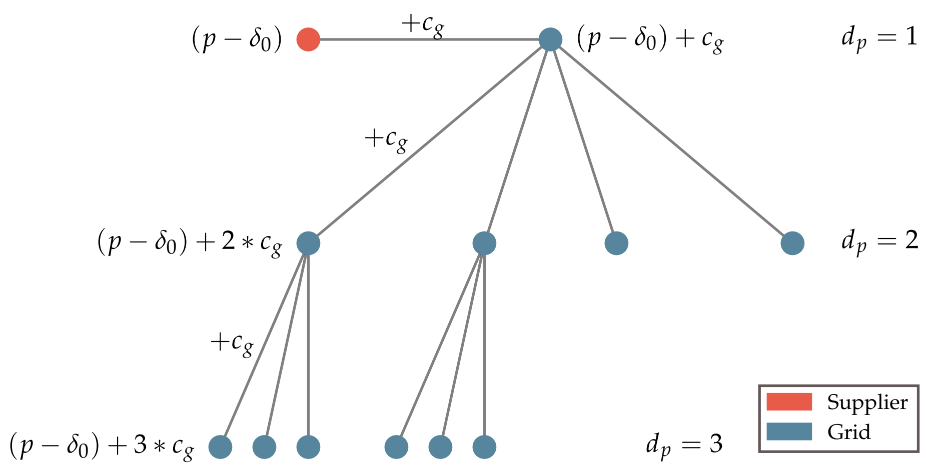


Figure 4.3: Example illustrating how prices propagate in the relatively close neighborhood of one supplier for very large N and $k=4$.

We use this to define the maximum propagation distance d_{max}, that a price reduction of δ_0 has, as:

$$d_{max}(\delta_0,c_g)=\begin{cases}\left\lceil\frac{\delta_0}{c_g}\right\rceil, & \text{if } \left\lceil\frac{\delta_0}{c_g}\right\rceil \leq \langle l\rangle,\\ \langle l\rangle, & \text{else.}\end{cases} \tag{4.8}$$

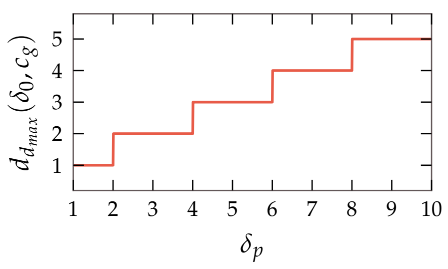
using the average shortest path length $\langle l\rangle$ as a bound, since at some point the change will be propagated to the entire network. As one can see in Figure 4.4a, this is equivalent to a step function, for the depth increases each time a price reduction is larger than a multiple of the transportation cost c_g.

Each time a price reduction is large enough, to reach a new propagation distance, all neighbors of the nodes on the previous depth will be influenced by this reduction. Therefore, we can define the number of nodes n_d, which are influenced by a price reduction with propagation distance $d = d_{max}$ recursively as:

$$n_d(d,k)=\begin{cases}1, & \text{if } d=1,\\ k+n_d(d-1,k), & \text{if } d=2,\\ (k-1)*n_d(d-1,k), & \text{if } d>2,\end{cases} \tag{4.9}$$

which can be roughly approximated as $(k-1)^d$ for $d >> 2$ and therefore, as

(a) Propagation depths for $c_g = 2$



(b) Number of influenced nodes

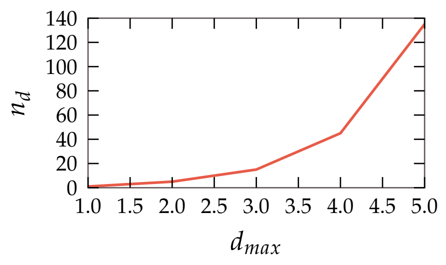


Figure 4.4: Propagation depths for growing δ_0 and $c_g = 2$ and the corresponding number of influenced nodes per depth. One can see, that the propagation distance increases each time δ_0 is larger than a multiple of c_g.

one can see in Figure 4.4b, increases exponentially with at each new propagation distance.

Experimental Validation

We validated this theoretical bound for n_d on a random network generated with the method proposed by Watts and Strogatz [13] with $N = 40$ and $k = 4$ for different generation-cost / transportation-cost r ratios $\rho_c = \frac{c_p}{c_g}$ (see Section 4.1.2). The resulting network has an average nodal degree $\langle k \rangle \approx 3.33$ and an average shortest path length $\langle l \rangle \approx 3.81$.

As one can see from Figure 4.5, where we display the number of influenced nodes normalized with the nodal degree k for different price reductions δ_0, the influence of a single supplier in the network is high for large values of ρ_c, i.e. when the transportation costs have a small impact on the total cost, but decreases rapidly for smaller values of ρ_c. For the value $\rho_c = 1$, which we assumed to be realistic in Section 4.1.2, a supplier can influence grid nodes up to a propagation distance of 3, after which the transportation costs will make its offer unfeasible for other nodes.

This proofs our initial claim, that the influence radius of a price in change in a single supplier is bounded by the transportation costs in the network and that even very large suppliers, like in our example with an infinite amount of power, can only influence a local portion of the network.

4.2.2 Influence of a Price Reduction δ_0 on the Global Average Price

In the previous section we discussed, how many grid nodes are influenced by a price reduction δ_0 in supplier s_i. Now we will use this notions to analyze, how such a price reduction impacts the average price in the network, since it influences nodes differently, depending on their distance to s_i.

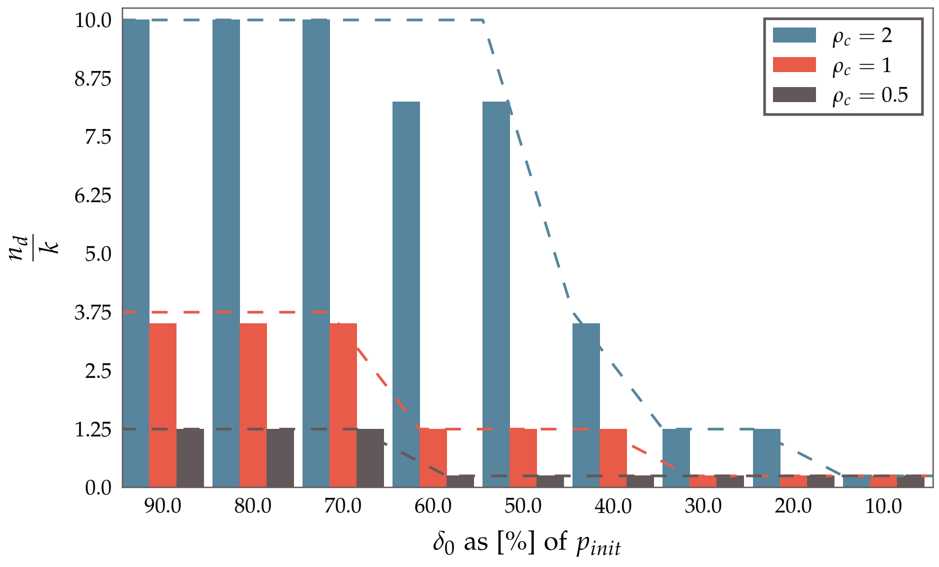


Figure 4.5: Comparison of the computed n_d and values measured from a randomized network with $N = 40$ and $k = 4$ for different generation-cost / transportation-cost ratios ρ_c.

Theoretical Bound

Assuming the same network we introduced at the beginning of Section 4.2, with N grid nodes, which all have the same demand and are connected to one supplier and where each supplier can provide an infinite amount of power, the initial global average price p_{avg} in each grid node is given by:

$$p_{avg} = \frac{1}{N} \sum_{i=1}^{N} \frac{s_i(p_{init} + c_g)}{s_i} = p_{init} + c_g. \tag{4.10}$$

Given a price reduction δ_0 in one supplier s_i, the price will be reduced in $n_d(d,k)$ nodes by an amount which decreases with growing distance to s_i, since each additional step adds new transportation costs. More formally, assuming that all transportation costs are given by c_g, the price reduction δ_i at propagation distance $0 \le i \le d_{max}$ amounts to:

$$\delta_i = \delta_0 - i * c_g. \tag{4.11}$$

Assuming that at each new propagation step $0 < i \le d_{max}$, the amount of newly influenced nodes $n_{inc}(i,k)$ is given by:

$$n_{inc}(i,k) = n_d(i,k) - n_d(i-1,k), \tag{4.12}$$

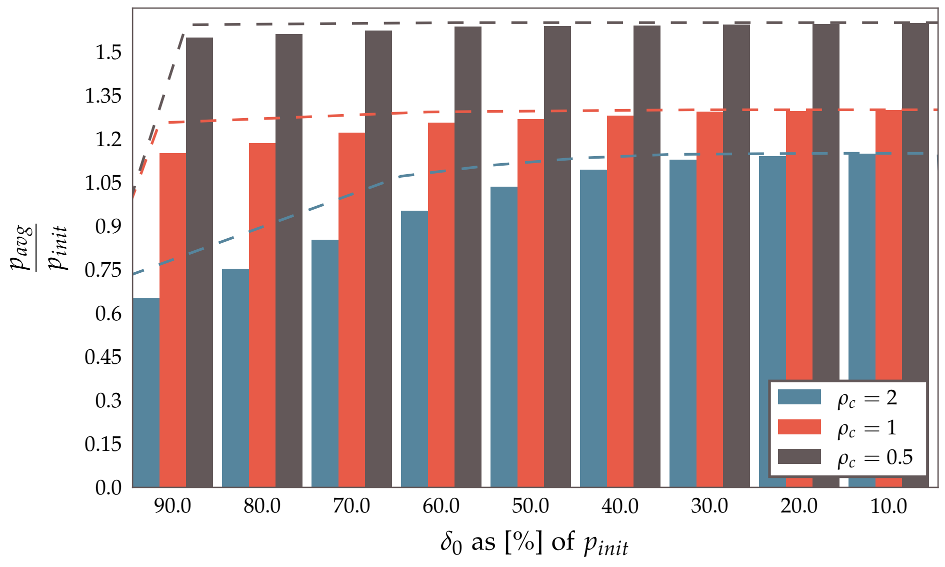


Figure 4.6: Comparison of the computed p_{avg} and values measured from a randomized network with $N = 40$ and $k = 4$ for different generation-cost / transportation-cost ratios ρ_c.

he total price reduction in our network δ_{total} can be determined as follows:

$$\delta_{total}(\delta_0) = \sum_{i=1}^{d_{max}(\delta_0, c_g)} n_{inc}(i,k) * \delta_i. \tag{4.13}$$

which leads us to the new average price:

$$p_{avg}(\delta_0) = p_{init} + c_g - \frac{1}{N} * \delta_{total}, \tag{4.14}$$

which as one can see from the dashed lines in Figure 4.6, is again bounded by the transportation costs in the network.

Experimental Validation

We validated our theoretical bound for the average price performing measurements on the same network used for the previous experiment. The results displayed in Figure 4.6 show, that the assumptions we made in this section give an acceptable upper bound for $p_{avg}(\delta_0)$. Again one can see, that the influence of a single supplier is bounded by the ratio between production costs and transportation costs ρ_c: In a network, where transportation costs make up a large portion of the total costs (high ρ_c) the influence of a single price reduction has far less impact than in a network, where production costs are more significant.

4.3 Pricing Strategies

After having determined the influence a change in a single supplier can have on the average price of the whole network, we will now direct our attention on the influence it has on the supplier itself. This will help us to get a feeling, about which strategies a supplier might apply in real life scenarios and will serve as basis for determining the global effects that profit maximizing strategies of the suppliers can have on the stability of the system.

To achieve this we will first analyze how a price change of δ_0 in one single supplier s_i influences the amount of the power sold by that determined supplier. We will then look at the profit it makes at different prices and try to determine a profit-maximizing strategy, which we will finally use to see what happens, when every supplier applies such a strategy.

We will do this by performing simulations on a network similar to the network used for our discussion in Section 4.2, but where the available power supply is limited: The network has N grid nodes $g_i \in G$, which are each connected to one supplier $s_i \in S$, one demand $d_i \in D$ and k other grid nodes. While the total available supplied power is set to 150% of the total demand, the supplier under test can supply an amount from $0\% < \iota_i < 50\%$ of the total demand, while the remaining supply is equally divided among the other suppliers. More formally, we define the *influence* ι_i of a supplier, as percentage of the total demand it can potentially cover and state that, given the total power demand d_{tot} and the influence of the supplier s_i under test $0\% < \iota_i < 50\%$ the influence of a supplier s_x in our network is given by:

$$\iota_x = \begin{cases} 0\% < \iota_i < 50\% & \text{if } x = i, \\ \frac{(150\% - \iota_i)}{N} & \text{else.} \end{cases} \tag{4.15}$$

Using this influence distribution we make sure, that although one supplier may be able to cover a large portion of the total demand, no supplier is essential for covering the demand. In other words, even if a supplier has a very large influence, there is no guarantee it will sell power if it makes an uncompetitive price offer.

Like in the examples in Section 4.2, the network topology is randomized using the method proposed by Watts and Strogatz, with a probability of $p = 0.5$ [13]. Each transportation has the same constant grid-cost c_g, which is chosen so that the ratio of generation and transportation costs $\rho_c \approx 1$. In the initial stable state, each supplier offers its energy for the same price p_{init}.

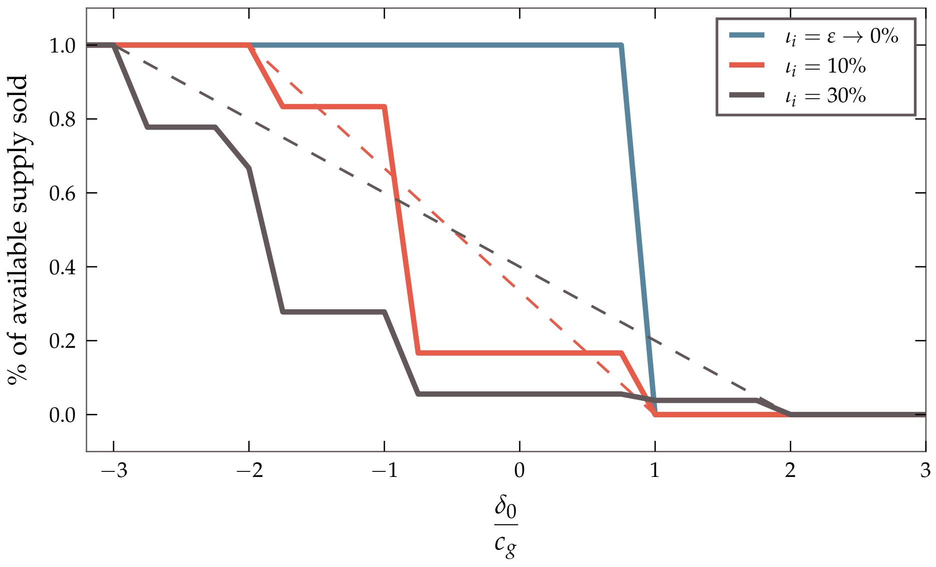


Figure 4.7: Shows how much of their available power suppliers with different influences sell for increasing price offers $p_{init} + \delta_0$. The dashed lines display the range between the price, where a supplier sells all of his available supply and the price, where it doesn't sell any power.

4.3.1 Influence of Price Change δ_0 on Power Sold by s_i

In order to determine how a price change of δ_0 in a single supplier s_i with influence ι_i affects the amount of power it sells, we performed three simulation runs on a randomized network with $N = 40$ and $\langle k \rangle \approx 3.05$ for three different influences: A very small $\iota_i = \varepsilon \to 0\%$, an intermediate $\iota_i = 10\%$ and a high influence $\iota_i = 30\%$.

The results of our experiment are displayed in Figure 4.7. We recall, that in our test network each grid-node is connected to exactly one supplier and one demand, all transportation costs in the network are given by c_g and that the pricing algorithm minimizes all costs in the network. Therefore, we can assume that if each supplier offers the same price p_{init}, each supplier will provide at least some amount of power to the grid node its is directly connected to: We assume, that a supplier s_i with a very low influence ι_i, is not able to cover the whole demand of that grid node g_i. Therefore, if it offers the same price p_{init}, offered by all other suppliers, it will sell all of its available supply to g_i. Since the power needs to traverse one edge to get from s_i to g_i, the price offered by s_i will be perceived as $p_{init} + c_g$ at g_i. The remaining demand at g_i, will be covered by other suppliers and will therefore have to traverse at least one more transportation edge, which amounts to a perceived price of at least $p_{init} + 2 * c_g$. For the supplier with very low influence s_i this means, that it can increase its price as long as it stays under the price perceived by g_i for other suppliers, and it will still sell all of its available supply. As soon as however the price offered by s_i is above this threshold ($\delta_0 > c_g$), the other suppliers, who a are already supplying power to g_i and have a significantly higher influence, will be able to cover the part of the demand previously covered by s_i for a lower price, which will make s_i's demand unfeasible, which means that it wont sell any more power. This behavior is also confirmed by our simulation results in Figure 4.7, where one can see the very steep fall of the amount of sold power, as soon as the supplier with low influence increases the price over the $\delta_0 > c_g$ threshold.

This behavior however changes as soon as the influence ι_i of one supplier s_i is large enough to cover the demand of more than the one grid node g_i it is directly connected to: In this case, in order to sell all of its available supply, the supplier needs to decrease its price to reach as many grid nodes as are needed to consume all of its supply. In contrast to suppliers with small influence, suppliers with a rather high influence ι might even be able to sell some power even when their price is higher than the threshold $\delta_0 > c_g$. This is due to the fact, that neighboring grid nodes might not be able to rely on their directly connected suppliers exclusively to cover their demand: In this case, a relatively high price offer by a supplier with high influence, which is in the close neighborhood might still be more competitive than others. This observations can also be verified by looking at the simulation results in Figure 4.7, where the supplier with the highest influence on one hand needs to decrease its price the most in order to sell all of its available supply, but on the other hand still manages to sell power with higher prices than other suppliers.

In this section we drew some conclusions, about how prices impact the amount of power a supplier can sell depending on its influence. However, this is still not sufficient to formulate a profit-maximizing strategy, since it is not clear if selling more power for a low unit price is more profitable than selling less power for a higher price. This question will be the focus of the next section.

4.3.2 Influence of a Price Change δ_0 on the Profit of s_i

In the last section we performed some experiments to show, how a price change of δ_0 in one single supplier impacts the amount of power it can sell, depending on its influence. We will now use those results to determine how such a price variation influences the profit of a supplier.

Figure 4.8 shows the profits, normalized by the maximum profit, each supplier from the experiment made in Section 4.3.1 achieved for different price

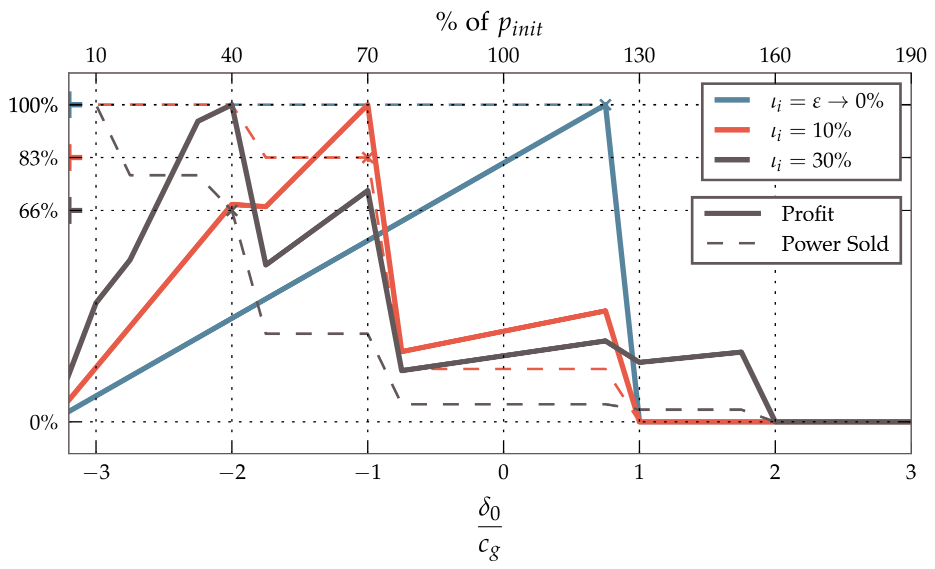


Figure 4.8: Shows the profit normalized by the maximum achieved profit for suppliers with increasing influence along with the amount of available supply they are selling for different price variations δ_0 from the initial price p_{init}.

variations δ_0: We notice, that the amount of its available power a supplier s_i needs to sell, in order to achieve the maximum profit, decreases with increasing influence ι_i. The reason for this is very simple for a supplier s_i with very low influence $\iota_i = \varepsilon \to 0$: Since, as we discussed in the previous Section 4.3.1, it has only enough available power to supply its directly neighboring grid node g_i and will always sell all of that supply as long as its price stays under the threshold $p_{init} + c_g$, it will achieve its maximum profit by offering a price lays exactly at that threshold.

The results however are more interesting for suppliers with a higher influence: Since they have the potential to reach an increasing number of nodes, and therefore to sell more power, by lowering their price, they need to find a trade-off between the amount of power they sell and the price they offer. More formally, if we express the price p_i offered by a supplier s_i as a factor x_i of p_{init}, so that:

$$p_i = x_i * p_{init} \tag{4.16}$$

and we the amount of power s_i with influence ι_i can sell at price x_i is given by some function $y(x_i, \iota_i)$, the supplier maximizes its profit by finding:

$$\max_{x_i}(x_i * y(x_i, \iota_i)), \tag{4.17}$$

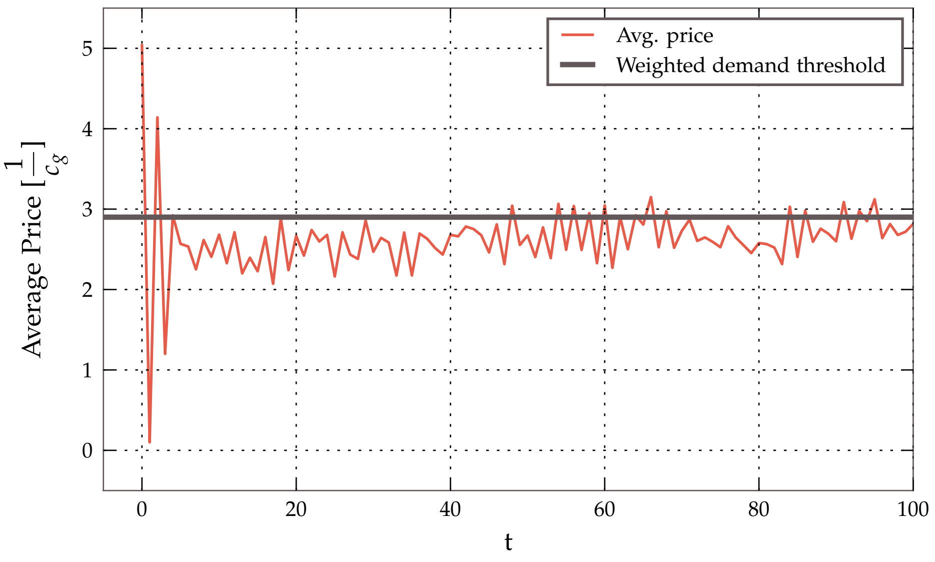


Figure 4.9: Shows the average price-threshold of the demand-nodes, weighted by their power-demand, as well as the average price in the network for increasing simulation steps

which as one can also see in Figure 4.8, is dependent on the influence of the supplier, as well as from the network topology: However, a supplier in our system only has knowledge about the current price in its neighboring grid node, as well as about the amount of power it is currently selling at a determined price. This means, that it does not have enough information to generally compute an optimal strategy and needs to rely on try-and-error as well as on experience.

4.4 Dynamic Simulation

While in the last sections we focused on analyzing how our system reacts to changes in one single supplier, now we would like to consider a case, where all suppliers and demands follow a certain dynamic strategy, which is dependent on the current state of the system. The goal of this analysis, is to determine, whether the average price in the network converges towards a reasonable interval, or whether it fluctuates.

In particular, we will simulate a scenario, where the total available supply in the network is slightly higher than the total available demand and where the power supply of each supplier is randomized. In contrast to a real scenario however, the single supplier know their influence ι and use it to formulate a strategy, loosely based on the empirical values derived from Figure 4.8.

More specifically, each supplier iteratively adjusts its price with the goal to sell around $(100 - \iota)\% \pm \frac{\iota}{2}\%$ of its power: If it sells below this threshold, it reduces its price by c_g, the grid cost of the network, if it sells above, it increases its price by this amount.

At the same time, each demand node has a random power-demand and a random price threshold: When the price at their location of the grid exceeds this threshold, they set their power-demand to zero and wait until it returns under the threshold, until they start buying again.

The result of this simulation, is displayed in Figure 4.9, which shows the average price-threshold of the demand-nodes, weighted by their power-demand, as well as the average price in the network for increasing simulation steps. As one can see, in the beginning the suppliers start with a price well-above the threshold. The price in the network immediately plummets, since no demand is willing to buy power for that price. As a consequence, the suppliers reduce their price-offer, which causes some of the demands to buy power. After only a few iterations, the price reaches a value around the average price threshold, where it stays with some fluctuations, due to suppliers trying to maximize their profit. This can be considered as a meaningful result, since the price stays around the maximum value that customers are willing to pay, which reflects the free-market behavior, which is governed by the principle of supply and demand.

Chapter 5

Implementation

After having assessed the theoretical and behavioral properties of our algorithm, we proceed with the implementation of a working prototype for a server based system, which can be used for reference for later implementations and which we will use later to gain some insights about the performance and possible bottle-necks. In this section we will describe the general architecture and the main components the prototype, that we implemented as a Java EE application on top of the Spring Framework[1].

5.1 Architecture

The prototype described in this section and depicted in Figure 5.1, was implemented mainly using two architectural styles:

REST. In our system we have a number of suppliers and consumers that communicate with a server to send updates about their own state (e.g. a supplier submits a new price offer) and to query the state of the system (e.g. the current energy price at their respective grid node). We decided to implement this *client-server* communication using the HTTP-protocol, since it is based on the IP protocol, which is widely available and it some countries even a requirement [16] for the communication infrastructure of smart grids.

In order to achieve a high scalability for our system the communication uses Representational State Transfer (REST) style. REST starts from an abstraction of the system-components into processing elements (client/server), data elements (state updates/prices) and connecting elements (HTTP) and enforces specific constraints to their interaction [17]. Among others, one of the main constraints is that the interaction between components must always be state-less, meaning that each request sent by a client needs to contain all

[1]https://spring.io/

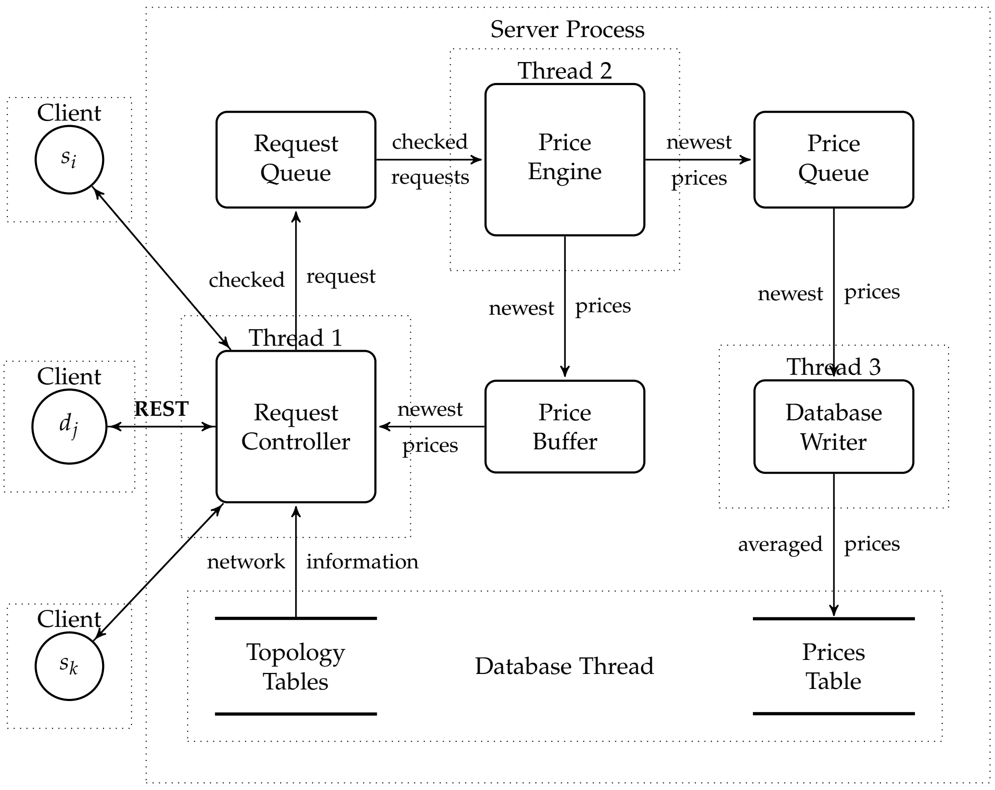


Figure 5.1: Architecture of the implemented prototype: Suppliers and demands communicate with a server using a client-server architecture, the server processes their requests using a filter-and-pipe data-flow.

the information needed by the server to fulfill that request: This greatly simplifies parallel processing of requests, since there is no need to share state information between server processes.

Data-Flow. Once requests are received on server-side, they are processed following the *filter-and-pipe* architectural style, which is a sub-style of the data-flow style [18]. In a data-flow architecture, the system is viewed as a series of transformations on successive pieces of input data. The filter-and-pipe style, incrementally reads in streams of input data and incrementally produces streams of output data. Each transformation step in between is computed by an independent component, the filter, and forwarded to the next filter using a FIFO-Queue, the pipe. The fact, that the processes do not share a state and only communicate through concurrent pipes, leaves room for scalable and parallel implementations: In the case of our prototype, all filters are implemented as separate threads, which are launched from a main server process.

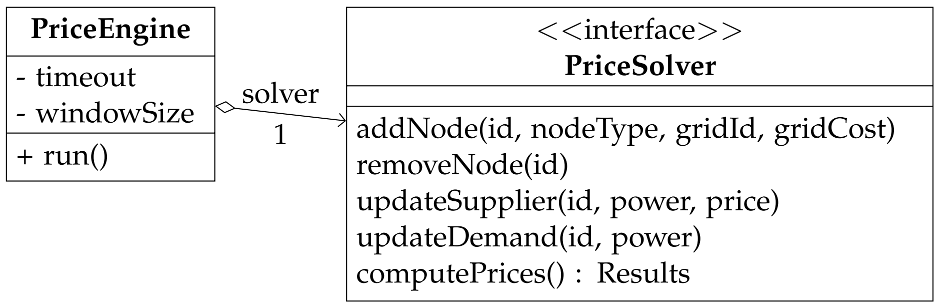


Figure 5.2: UML-Diagram illustrating the runnable Price-Engine and some selected API methods of the Price-Solver.

We will now give a coarse overview about the filters and how they process incoming requests using Figure 5.1 as a reference: The details of the workings of the most important filters and queues will be given in subsequent sections:

Incoming requests are first handled by the request controller, which validates their syntax as well as some semantic properties, using topological network information from a database: Invalid requests are rejected, while requests that can be answered immediately, like requests for the price at a certain node, are answered. Requests that need further processing, like a supplier updating its price offer, are put into a thread-safe FIFO-Queue, the Request-Queue.

The price engine polls the requests from the queue and uses them to update its own internal state, which consists of the network topology: After a certain amount of updates or after a certain time-out, it re-computes the prices for all nodes in the network, marks them with the current time-stamp and forwards them to both: The Price-Buffer, which always contains the newest prices and is used by the Request-Controller to reply to price-requests and the Price-Queue.

The Database Writer collects the timestamped prices from the Price Queue and periodically dumps the averaged prices, weighted by time-interval, to the database.

5.2 Price-Engine

The Price-Engine is a thread instantiated once by the main process and executed at start-up: It is responsible for periodically computing the prices of all enabled nodes in the network, using the algorithm introduced in Chapter 3. The most important parts of the Price-Engine are the `PriceSolver`, an object owned exclusively by this thread, which holds a graph representation of the network and performs all the necessary computations, and the

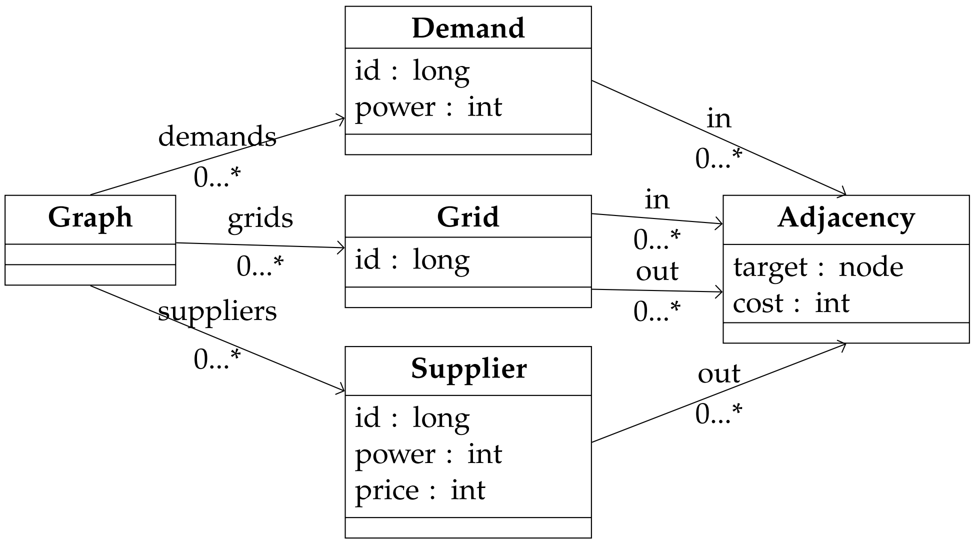


Figure 5.3: Diagram illustrating the graph representation of the Price-Solver. The nodes are stored for fast access in three different hash-tables, each node contains a list for its incoming or outgoing adjacencies.

`run`-Function, which keeps the state of the solver up to date and triggers the price computations.

5.2.1 Price-Solver

The Price-Solver essentially consists of two parts: The first part is composed by the graph representation of the network, for which it shall compute the prices and of public methods, to keep that representation up to date (see Figure 5.2). The second part is a set of methods that use the graph representation to formulate a linear program for an external LP-solver and finally compute the price at all nodes.

Graph Representation

The algorithm we introduced in Chapter 3 basically consists in three parts: Solving a minimum cost flow problem, performing a topological sort and finally computing the prices for the ordered set of nodes. Therefore we want to chose a graph representation, that allows to efficiently perform each of this steps and can quickly be updated to reflect the current status of the network.

In general there are two ways to store a directed weighted graph $G(V, E, W)$ with n nodes: As *adjacency matrix*, an $n \times n$ matrix where each entry $a_{i,j}$ represents the weight of the edge between v_i and v_j, or as an *adjacency list*, a collection of lists containing the neighboring nodes and the weight of the edge connecting them for each node v_i [19]. Since the electrical grid we want to represent has small-world properties (see Section 4.1.1), which implies that the graph is relatively sparsely connected, we chose to implement our graph using the more memory-efficient adjacency lists.

As one can see in the diagram in Figure 5.2, the Price-Engine offers several methods to keep the status of the network up to date: It is possible to add single nodes, to remove single nodes as well as updating the state of suppliers and demands. In order to perform this tasks efficiently, we must be able to quickly add, remove and access single nodes to the internal representation, regardless of the scale of the network. To achieve this, we designed the data-structure illustrated in Figure 5.3: Each node is represented as separate object, which contains the state of the node, as well as the adjacency lists for incoming and or outgoing edges. The nodes are stored into three different hash-tables, according to their type, which allows to add, remove and retrieve them in constant time, regardless of network size [19].

This data structure also allows to efficiently extract the objective function and the constraints of the linear program for finding the minimum cost flow: We recall our objective function as:

$$\min \sum_{(i,j)\in E} c_{i,j} x_{i,j}, \tag{3.2}$$

where E is the set of all edges, $c_{i,j}$ the cost and $x_{i,j}$ the power flow from node i to node j. We can formulate this by iterating each node and extracting the cost from all outgoing adjacencies. In the same iteration, we can also formulate our constraints in a similar manner: Therefore we conclude, that the formulation of the mathematical program is performed in linear time $O(m)$, where m is the number of edges in the network.

Once the minimum cost flow has been computed, this data-structure allows to efficiently perform the topological sort and the price propagation in one step: Since each flow in our network originates from a supplier, and the suppliers are stored in a separate hash-table, we can start by propagating the prices from them to all their outgoing adjacencies. As soon as a node has received the prices from all its incoming adjacencies, it can compute its own price propagate it to its own outgoing adjacencies until all demand nodes have been reached. Also this process has a linear maximum execution time of $O(m)$.

Solving the Minimum Cost Flow Problem

After having devised a data-structure, which is suitable for efficiently formulating the minimal cost flow problem as a linear program, we need to actually solve it. This step is performed using an external linear programming

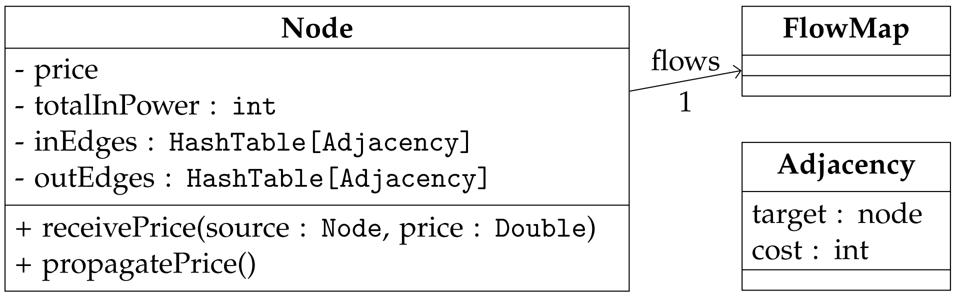

Node
- price
- totalInPower : int
- inEdges : HashTable[Adjacency]
- outEdges : HashTable[Adjacency]
+ receivePrice(source : Node, price : Double)
+ propagatePrice()
flows
1
FlowMap
Adjacency
target : node
cost : int

Figure 5.4: UML-Diagram illustrating the functions and objects employed by the nodes to propagate the prices in the network. See Appendix A.1 to see how the functions are implemented.

solver (LP-Solver). There is a wide variety LP-solvers, that differ in many ways like licensing, price, offered features, performance and not least the JAVA interfaces they offer (a detailed comparison can be found in [20]). This last point makes it very cumbersome to keep our implementation modular and open enough, to experiment with different solvers, since that would imply rewriting large parts of the code for each solver. Nevertheless, we were able to evade this problem by using the *Google Optimization Tools (or-tools)*[2], a set of open source optimization libraries, which among other offer a unified JAVA interface to several LP-solvers: These solvers include *GLOP*, an open source solver, which in Section 6.1.2 we will benchmark against *Gurobi*, the solver that outperformed all other solvers in [20].

Regardless of which solver is used, once the `computePrices()`-function (see Figure 5.2) is triggered, the objective function and the constraints are extracted from the graph-representation of the electrical grid, brought into the standard form required by *or-tools* and the minimum cost flow problem is solved. The output of the solve is a hash-table mapping each edge of the graph to the optimal power-flow running through that edge.

Price Propagation

Once we have the power-flow along each edge of the electrical grid, we can compute the energy price $p(i)$ at each node i using Equation 3.7, which in Chapter 3 we defined as:

$$p(i) = \frac{\sum\limits_{\forall(j,i)\ \in\ E_{in,i}} x_{j,i} * (c_{j,i} + p(j))}{\sum\limits_{\forall(j,i)\ \in\ E_{in,i}} x_{j,i}}, \tag{5.1}$$

[2] https://developers.google.com/optimization

where $(j,i) \in E_{in,i}$ is the set of incoming edges at node i, $x_{j,i}$ is the optimal power-flow and $c_{j,i}$ is the cost along edge (j,i).

Since the price at node i also depends on the price $p(j)$ of each neighbor with an incoming edge $(j,i) \in E_{in,i}$, we need to compute the prices in topological order, starting from the suppliers, which don't have any incoming power-flows.

This is achieved by two functions in the node-objects: `receivePrice(...)` and `propagatePrice()` (see Figure 5.4). The function `receivePrice(...)` allows a node to receive the current price from a previous node. It takes two arguments, namely the id of the previous node and its price, and stores them in its inner state along with the optimal power flow of the corresponding edge. Once it has received the prices from all the previous nodes, it computes its own price using Equation 3.7 and calls the function `propagatePrice()`. This function goes through all outgoing neighbors of the node, and sends them its own price by calling `receivePrice(...)` on them. This process will continue until every node has its own price. For a more detailed pseudo-code description of this process, please refer to Appendix A.1.

Output of the Price-Solver

The output of the price solver is a hash-table mapping the ids of all nodes, for which a price has been computed, to that price, the power flowing into that node, the power flowing out from that node as well as the time at which those values have been computed.

5.2.2 Run Function

The run function takes care of polling state updates from the Request-Queue, using them to update the internal state of the Price-Solver and to periodically trigger new price computations.

A pseudo-code implementation of the function is provided in Algorithm 1: Here we can see, that the `run()`-functions first polls the Request-Queue for new update requests. If there are no new requests in the queue, it waits for a certain time-out interval. If a new request was retrieved, the function takes care of accordingly updating the internal state of the price-solver. If a certain number of request has already been processed (window-size), it will trigger a new price computation. If no new request was retrieved, but there have been other requests which have been already processed, since the last price computation, it will also trigger a price computation.

The two parameters, `timeout` and `windowSize` (see Figure 5.2), can be used to influence the throughput of the Price-Engine: On slower systems, where

Algorithm 1 Function that runs the Price Engine.

```
function RUN( )
    pending ← 0  ▷ Number of state updates since last price computation
    while true do
        request ← QUEUE.poll(timeout)                    ▷ Get next request
        if request ≠ null then                      ▷ If poll did not time out
            processRequest(request)          ▷ Update state of Price-Solver
            pending++                   ▷ Increase number of pending updates
        end if
        if (request = null && pending > 0)
                || (pending > windowSize) then
            computePrices()           ▷ Compute new price for every node
            pending ← 0                                      ▷ Reset counter
        end if
    end while
end function
```

computing the prices for the whole network takes a longer time, it might be a good strategy to keep relatively high values for window-size and time-out. On faster systems, one might choose to reduce them, to achieve a higher time resolution for the prices.

Copies of computed prices at each node, as well as the amount of power each supplier can sell, are stored into two separate, thread-safe, locations: The Price-Buffer, a static field which always contains the newest computed results and whose access is regulated by semaphores and the Price-Queue, a thread-safe queue implementation, where the results are stored until the Database-Writer collects them.

5.3 Database

The Database is an independent component of our system, that is needed to provide network information to the request controller as well as storage for the prices computed by the price engine. The MySQL-Database use in this implementation, consists of 4 tables: three to capture the network topology and one containing the computed prices.

Topology Tables. In Section 3.1 we introduced a mathematical model of the electrical grid as weighted directed graph $G(V, E, C)$ where suppliers, demands and sub-grids are vertices $v_i \in V$ and the transmission lines between them are represented as directed edges $e_{i,j} \in E$ with cost $c_{i,j} \in C$. This model of the grid was represented in a relational-database using three tables:

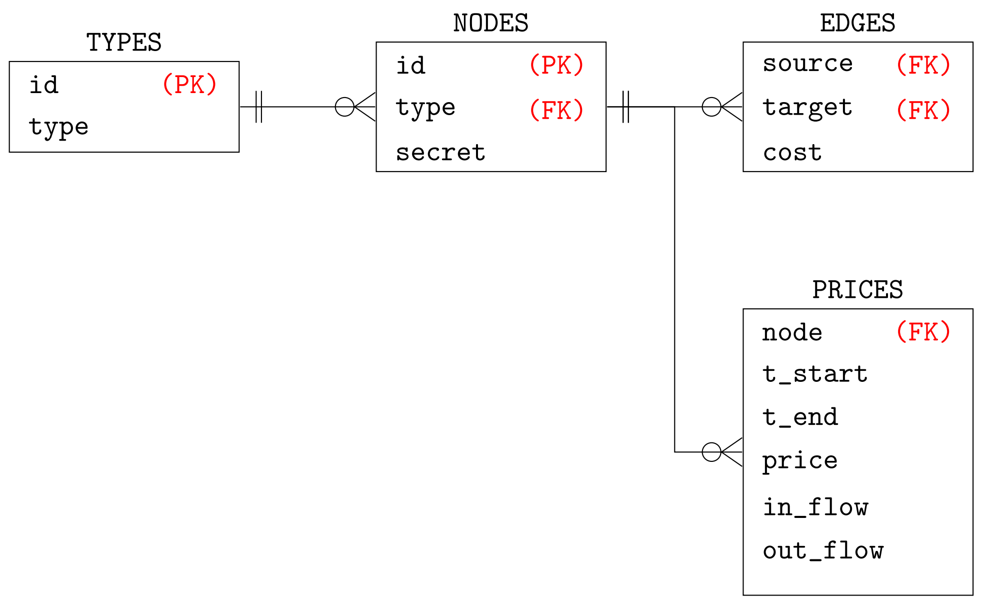


Figure 5.5: Entity-Relationship model of the database: All connection lines indicate one-to-many relationships, while the type of key (primary or foreign) is indicated in brackets.

Types The types-table contains the different types a node can have. In our case, a node can be of type supplier, demand, grid or exchange. Each type has a unique id, which is intended to be used as foreign key by other tables and a textual description.

Nodes The nodes-table contains an entry for each supplier, demand, grid or exchange known to the system. Each node has a unique id, a type and a secret, which can be used to authenticate the requests sent by nodes.

Edges The edges-table contains an entry for each directed edge in the network. An entry consists in the id of the source node, the id of the target node, as well as the costs associated with traversing that edge.

All entries in this table are only used by the Request-Controller to validate incoming requests and to augment them with information needed by the Price-Engine to compute the prices: It is not needed by the Price-Engine itself and therefore not performance critical to the computations.

Prices Table. The periodically computed prices for each node are stored into the prices-table along with the time-range they are valid for and the incoming and outgoing power-flow. The entries in this table are intended to be used to charge customers and compensate suppliers but are not relevant to the runtime-system.

Command	Parameters	Description
`enable`		*Enable node with the given id*
	`nodeId`	Id of node to enable
`disable`		*Disable node with the given id*
	`nodeId`	Id of node to disable
`updateSupplier`		*Set offered price and available supply for supplier with the given id*
	`nodeId`	Id of supplier to update
	`price`	New price offered by supplier
	`power`	New power supply available at supplier
`updateDemand`		*Set power demand for demand node with the given id*
	`nodeId`	Id of demand to update
	`power`	New power demand at node
`getStatus`		*Return price for demand and price at grid node and outgoing flow for supplier*
	`nodeId`	Id of supplier or demand

Table 5.1: Shows the commands made available by the REST-API of the Request-Controller. Each call can be authenticated by a secret, which was omitted from the parameters in this table.

5.4 Request-Controller

The Request-Controller is the interface between the server application and the clients: It receives the REST-requests from suppliers and demands, performs some validations, transforms them into requests understandable by the Price-Engine and replies to the clients.

The requests are sent by suppliers or demands as HTTP GET requests, where the Request-URI follows the following pattern:

```
http://HOST/COMMAND?PARAM1=val&PARAM2=val2[...],
```

where `HOST` must be replaced with the address under which the Request-Controller can be reached and where the command and the parameters are part of the REST-API displayed in Table 5.1.

Before a supplier or a demand is able to sell or buy power, it needs to announce it self to the system. They do this by calling the `enable` command along with their own unique id and their private secret. Upon receiving such a call, the Request-Controller uses information from the topology tables in the database to check if the caller is known the system and whether it can be authenticated using the shared secret. If those validations succeed, the controller retrieves the id of the grid node associated with the caller and the grid costs from caller to grid node and uses them, along with the caller id, to formulate a new `enable` request, which is put into the Request-Queue. After this, the status of the caller is set to `enabled`, which means that Request-Controller will from now on accept further requests from that node. A node can again disable itself, by calling the `disable` command, upon which it will be first removed from the Price-Engine and then set to `disabled`.

After being enabled, suppliers can update their price offer and their available power supply calling the `updateSupplier` command, while demands can update their power demand calling `updateDemand`. Upon receiving such a request, the Request-Controller if the caller is already enabled and the call parameters are in a valid value range, the request is put into the Request-Queue.

Lastly, suppliers and demands can query their own status using the `getStatus` call. Upon receiving such a call, if the caller is in `enabled` state, the system will immediately reply with the last available status: For suppliers, the status consists of price of the grid node they are associated with, the amount of power, that the system is drawing from them and the time-stamp, at which does values have been computed. For demands it consists of the price they are paying and the time-stamp.

As proposed by RFC-7231 [21], the Request-Controller uses the following status codes to indicate the success of a request:

Code	Text	Description
200	OK	Request successful
400	Bad Request	Syntax of request is invalid
401	Unauthorized	Authentication credentials missing or incorrect
403	Forbidden	Not enabled node tries to update
404	Not Found	Requested node or command not found

The body of the responses to the follows the JSON format as proposed by RFC-7159 [22].

5.5 Database-Writer

The Database-Writer is an autonomous component which runs in a separate thread and takes care of buffering the results computed by the Price-Engine over a determined time-interval, averaging them and asynchronously storing them to the `PRICES` table of the database.

We recall Section 5.2.1, where we stated that the output of the Price-Engine consists in a hash-table with an entry for each node i containing the time at which the price has been computed t, the price $p_i(t)$, the sum of the incoming power flows $x_{i,in}(t)$, as well as the outgoing power flow $x_{i,out}(t)$.

We define the average price over a time-interval $[t_1, t_n]$ as:

$$p_{avg}(t_1, t_n) = \frac{\int_{t_1}^{t_n} p(t)dt}{t_n - t_1}, \tag{5.2}$$

which given a series of times $T = [t_1, t_2, \ldots, t_n]$ and corresponding prices $P = [p_1, p_2, \ldots, p_n]$ can be approximated using right-Riemann sums as:

$$\tilde{p}_{avg}(t_1, t_n) = \frac{\sum_{i=2}^{n} (t_i - t_{i-1}) p_i}{t_n - t_1}. \tag{5.3}$$

The average incoming and outgoing flows for a node can be computed analogously.

The Database-Writer computes those averages at predefined time-intervals, after which it dumps them into the `PRICES`-table of the database.

Chapter 6

Performance

In this section we will stress-test the prototype and analyze the bottlenecks given by the single components and, whenever possible, suggest improvements.

The performance tests in this section will be performed on a PC equipped with a dual-core Intel® Core^{TM} i7-4600U CPU (2.1 GHz) and 12GB of RAM.

6.1 Price-Engine

As we described in the previous section, the Price Engine has to perform operations to keep its internal state up to date, has to solve a minimum cost flow problem and finally has to use those minimum flows to compute a price at each node.

In this section we will test the performance for each of those operations, identify the bottleneck of the solver and suggest how to solve it.

6.1.1 Updating the Network Topology

As we have seen in the previous section in Figure 5.2, the Price-Solve offers several API-methods to update the graph representation of the electrical network. They can be subdivided into two different kinds: Methods that update the *topology* of the electrical grid, by adding and removing nodes, and methods that update the *state* of single members of the network, e.g. the price offered by a supplier.

In order to measure the performance of these two types of operations, we took a random network of 20000 nodes, filled the Request-Queue with an increasing number of requests for each type and measured the time needed to perform all requests. As one can see from the results in Figure 6.1, in both cases the times grow linearly, with some noise caused by other threads being

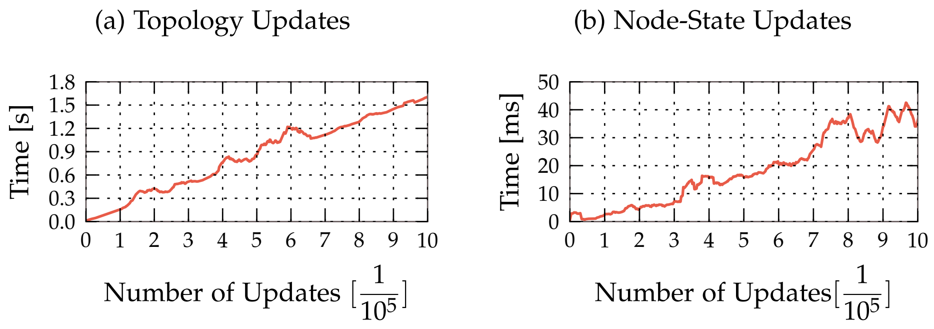


Figure 6.1: Shows the times needed to perform up to one million topology and state updates.

executed on the same CPU, for an increasing number of update operations. This is due to the fact, that each operation is performed in constant time.

One can also see, that topology updates take a significantly longer time compared to simple node-state updates. This is due to the fact, that while state-updates simply change the values of some object fields, adding or removing new nodes requires the instantiation of new objects.

The increased time needed to add and remove nodes to the internal graph representation however is less performance-critical than the performance of state-updates, since the expected behavior of an external agent (supply or demand), is to log on to the system once, perform a series of update actions and finally to log off: This means that in the normal use-case, state updates are expected to be significantly more frequent than topology updates.

6.1.2 Computing the Minimum Cost Flow

As already described in the previous section, the minimum cost flow needed to compute the energy prices for each node in the network is computed using an external LP-solver.

We tested the performance of two different solvers, the open source solver GLOP[1] and the commercial solver Gurobi[2], which according to [20] outperforms a number of commercial and open source solvers, excluding GLOP.

The tests were performed by solving the minimum cost flow problem for random networks of increasing size and by measuring the time needed to solve it. As one can see from the results in Figure 6.2, while the computation times for both solvers increase linearly for larger networks, Gurobi is consistently faster than GLOP. If for example using the latter, the minimum cost flow problem for a network with 100000 nodes can be solved in

[1] http://developers.google.com/optimization/lp/glop
[2] http://www.gurobi.com/

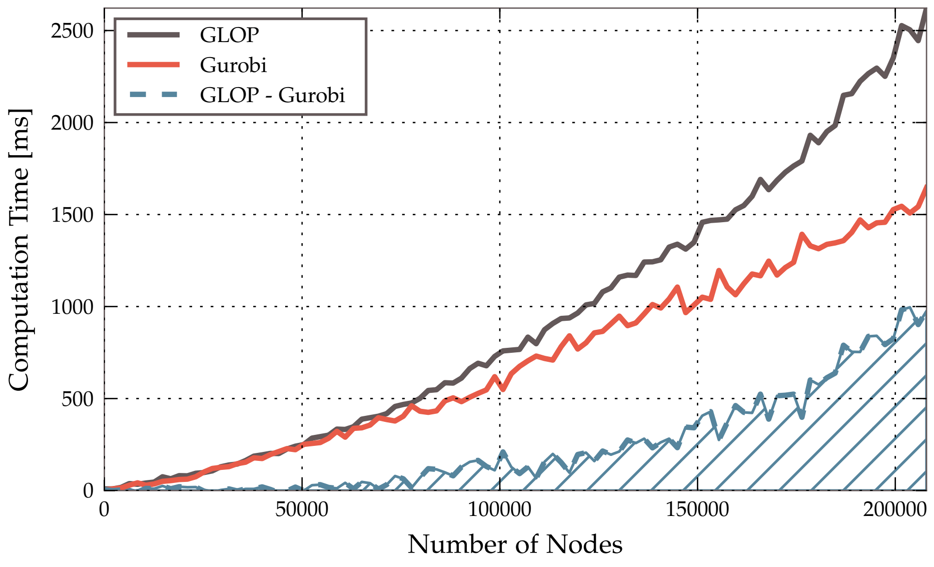


Figure 6.2: Shows the times needed to solve the minimum cost flow problem for networks of increasing sizes using the open source LP-Solver GLOP and the commercial solver Gurobi, as well as the time difference.

$\approx$ 1.2 seconds, it can be solved in $\approx$ 0.75 seconds using Gurobi. The average performance difference between the two solver lies at $\approx$ 37.5%.

6.1.3 Price Propagation

As described in the previous sections, after we have solved the minimum cost flow problem, the energy prices are computed in each node by propagating the prices from the suppliers to all other nodes, in topological order.

We tested the performance of the price propagation on networks with increasing size. As one can see from the results in Figure 6.3, the times increase linearly with increasing size of the network, with some noise caused by other threads running on the same CPU. Using GLOP, the price propagation on average amounts to $\approx$ 8.5% of the total price computation time, using Gurobi, which is consistently faster than GLOP, it amounts to $\approx$ 10.3%.

6.1.4 Overall Performance and Bottleneck

In this section we have analyzed the performance of the update operations of the Price-Engine, of the LP-solver and of the price propagation. While the update operations and the price propagation run fairly efficiently and scale very well, we can identify the solving of the minimum cost-flow as the main

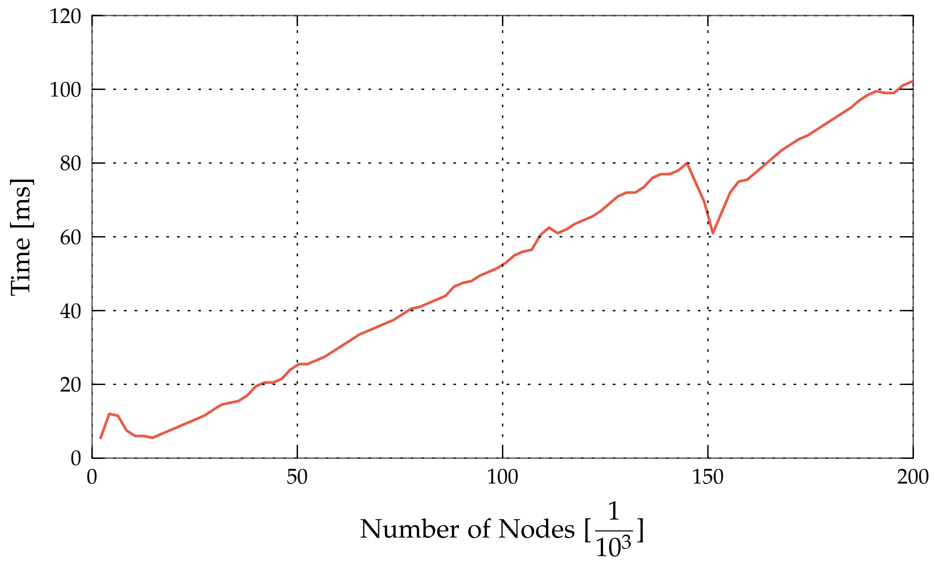


Figure 6.3: Shows the time needed to compute the prices, once the minimum cost flow has been computed, for increasing network sizes.

performance bottleneck. We can suggest different approaches to improve its performance:

Reduce Number of Variables. A first step in reducing the time complexity of our algorithm could be to reduce the number of decision variables and constraints in the model:

In Section 3.3, we added the following constraint to our model:

$$\sum_{(i,j)\in E} x_{i,j} = d_j, j \in D, \tag{3.5}$$

which states that the incoming power flow in a demand-node must always be equal to the power demand of that node. In other words, the power flow $x_{i,j}$ along an edge going from grid-node i to demand node j, will always be equal to the power-demand, no matter how big the cost $c_{i,j}$ on that edge is.

This fact can be used to reduce the number of decision variables and constraints by merging all demand nodes at a grid node into one demand node, whose demand is equal to the sum of the demands, and which is connected to the grid node with an edge with cost 0.

As one can see in Figure 6.4, where we compare the performance of Gurobi solving the standard model and the simplified model, on a network with n_g

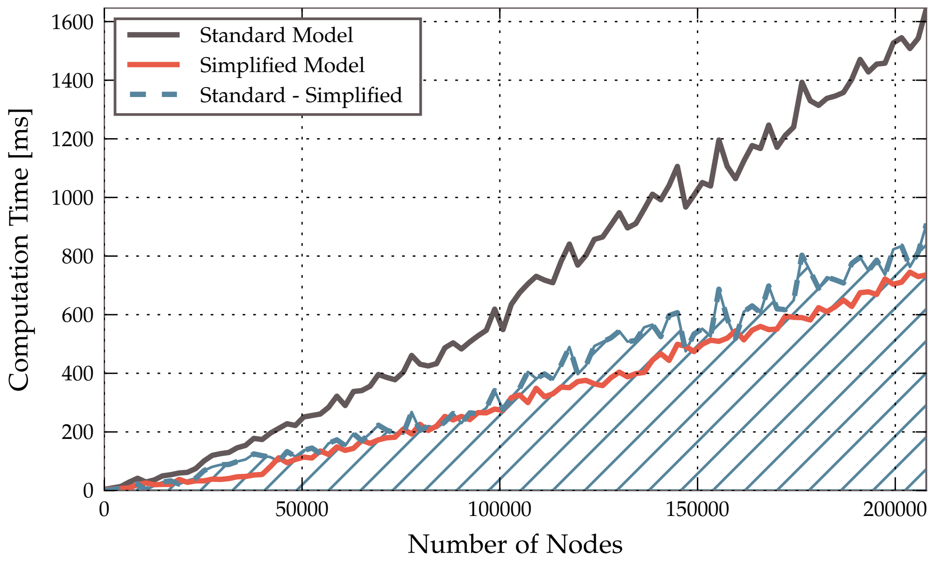


Figure 6.4: Shows the performance gained by simplifying the LP-model merging all demand-nodes at each grid-node.

grid nodes, $10 * n_g$ supplier nodes and $10 * n_g$ demand nodes, this leads to a significant performance improvement. In this example, solving the simplified model was $\approx 55\%$ faster on average.

Distributed Solver. Although we can increase the performance of our Price-Engine by simplifying the model, optimizing algorithms or by scaling up the hardware upon which our system runs, the centralized nature of our system will always set constraints to its scalability.

A better scalability could be achieved by using a distributed approach to solve the minimum cost flow problem, where the agents (suppliers and demands) solve local portions of the problem, and coordinate each other by passing messages. Such an approach, which however still relies on a global, centralized schedule, can be found in [9]. Another distributed approach, which uses market-based principles, to solve the minimum cost flow in an electrical grid can be found in [10]. Both of these approaches are however still preliminary studies, and have yet to be applied on larger scale systems.

Another approach to solve the optimal network flow problem distributedly is using *proximal message passing*, which has been applied in [23] and shown to be able to find optimal flows for networks with over 30 million agents in less than one second. It follows an iterative approach, based on the alternating direction multiplier method, where at each step, each device sends simple messages to its neighbors and minimizes its own objective function augmented by a term determined by the messages it has received. In contrast to other distributed implementations, this method does not rely on a centralized schedule and needs no global coordination, other than synchronizing the iterations.

Chapter 7

Conclusion

In this project, we proposed an algorithm to compute the local energy price for each location of an electrical grid based on supply and demand, which computes a minimal cost-flow of the network, based on which it determines the prices.

We asserted its low sensitivity, deriving theoretical bounds for the influence single components can have on the price and verifying them performing simulations on realistic randomized networks. We discussed possible profit-maximizing strategies for suppliers and established the stability of the algorithm simulating a network with agents that follow such a strategy.

We implemented this algorithm into a working server-application, which can be used to further evaluate our solution approach on real electrical grids and which can compute the prices for 200000 network nodes in around 1.5 seconds. During performance measurements, we came to the conclusion, that the main performance bottle-neck of this application is the solving of the minimum cost flow.

Further research to improve early-stage solution should be directed towards removing this bottle-neck: This could be done by implementing a distributed minimum-cost flow algorithm in the electrical grid using the alternating direction multiplier method.

Appendix A

Functions

A.1 Price Propagation

Algorithm 2 Price Propagation at node i.

```
function RECEIVEPRICE($j$, $p_j$)                          ▷ Receive price $p_j$ from node $j$
    $n_{in} \leftarrow n_{in} + 1$                          ▷ Increase number of received prices
    $x_{j,i} \leftarrow$ FlowMap.get(j,i)                   ▷ Get optimal power flow from $j$ to $i$
    $x_{in,tot} \leftarrow x_{in,tot} + x_{j,i}$            ▷ Increase total incoming power
    $c_{j,i} \leftarrow$ inEdges.get(j,i).cost              ▷ Get cost from $j$ to $i$
    $p_{tot} \leftarrow p_{tot} + x_{j,i} * (c_{j,i} + p_j)$   ▷ Increase total price
    if $n_{in} =$ size(inEdges) then                        ▷ All previous prices received
        $p_i \leftarrow p_{tot} / x_{tot}$                  ▷ Compute actual node price $p_i$
        propagatePrices()
    end if
end function

function PROPAGATEPRICES( )
    for Node target in outEdges do
        target.receivePrice(this, $p_i$)
    end for
end function
```

Bibliography

[1] S. Kaplan, "Electric power transmission: background and policy issues," *US Congressional Research Service, April*, 2009.

[2] R. Weron, "Electricity price forecasting: A review of the state-of-the-art with a look into the future," *International Journal of Forecasting*, vol. 30, pp. 1030–1081, Oct. 2014.

[3] X. Fang, S. Misra, G. Xue, and D. Yang, "Smart Grid — The New and Improved Power Grid: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 4, pp. 944–980, 2012.

[4] S. D. Ramchurn, P. Vytelingum, A. Rogers, and N. Jennings, "Agent-based control for decentralised demand side management in the smart grid," pp. 5–12, May 2011.

[5] M. Shahidehpour, H. Yamin, and Z. Li, *Market Operations in Electric Power Systems : Forecasting, Scheduling, and Risk Management*, vol. 9. 2002.

[6] R. Weron, *Modeling and forecasting electricity loads and prices: a statistical approach*. Wiley, 2007.

[7] P. Samadi, A.-H. Mohsenian-Rad, R. Schober, V. W. S. Wong, and J. Jatskevich, "Optimal Real-Time Pricing Algorithm Based on Utility Maximization for Smart Grid," in *2010 First IEEE International Conference on Smart Grid Communications*, pp. 415–420, IEEE, Oct. 2010.

[8] S. Bu, F. R. Yu, and P. X. Liu, "Dynamic pricing for demand-side management in the smart grid," in *2011 IEEE Online Conference on Green Communications*, pp. 47–51, IEEE, Sept. 2011.

[9] P. H. Nguyen, W. L. Kling, G. Georgiadis, M. Papatriantafilou, L. A. Tuan, and L. Bertling, "Distributed routing algorithms to manage power flow in agent-based active distribution network," in *2010 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT Europe)*, pp. 1–7, IEEE, Oct. 2010.

[10] B. HomChaudhuri, M. Kumar, and V. Devabhaktuni, "Market based approach for solving optimal power flow problem in smart grid," in *2012 American Control Conference (ACC)*, pp. 3095–3100, IEEE, June.

[11] G. A. Pagani and M. Aiello, "The Power Grid as a complex network: A survey," *Physica A: Statistical Mechanics and its Applications*, vol. 392, no. 11, pp. 2688–2700, 2013.

[12] Z. Wang, A. Scaglione, and R. J. Thomas, "Generating Statistically Correct Random Topologies for Testing Smart Grid Communication and Control Networks," *IEEE Transactions on Smart Grid*, vol. 1, pp. 28–39, June 2010.

[13] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks.," *Nature*, vol. 393, pp. 440–2, June 1998.

[14] O. Mangasarian, "Iterative solution of linear programs," *SIAM Journal on Numerical Analysis*, 1981.

[15] O. Mangasarian and R. Meyer, "Nonlinear perturbation of linear programs," *SIAM Journal on Control and Optimization*, 1979.

[16] Office of the National Coordinatorfor Smart Grid Interoperability, "NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 1.0," 2010.

[17] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," in *Proceedings of the 22nd international conference on Software engineering - ICSE '00*, (New York, New York, USA), pp. 407–416, ACM Press, June 2000.

[18] H. Zhu, *Software Design Methodology: From Principles to Architectural Styles*. Butterworth-Heinemann, 2005.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms , Third Edition*, vol. 7. 2001.

[20] B. Meindl and M. Templ, "Analysis of commercial and free and open source solvers for linear optimization problems," *Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS*, 2012.

[21] R. Fielding and J. Reschke, "RFC7231 Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content."

[22] T. Bray, "RFC7159: The JavaScript Object Notation (JSON) Data Interchange Format," 2014.

[23] M. Kraning, E. Chu, L. Javad, and S. Boyd, "Dynamic network energy management via proximal message passing," *Foundations and Trends in Optimization*, pp. 73–126, 2014.