TU Delft

Delft University of Technology

Thesis Proposal

Distributed Software Agents in Smart Grids

Author:
Roland Saur

Supervisors:
Dr. Amineh Ghorbani
Dr. Igor Nikolic
Dr. Martijn Groenleer
Prof.dr.ir. P.M. Herder

June 9, 2015

Contents

Summary

Global warming is one of the main issues of the twenty-first century. The introduction of renewable energy sources (RES) and electric vehicles(EV) is promising, but poses a problem to the low voltage grid.
Smart grid technology can be used to manage and thus mitigate the negative effects RES and EVs have on the grid. Smart grid technology to manage the low voltage grid not only deals with technical issues, but also has a social component. Smart gird technology should respect privacy, not ask for much user interaction and be widely accepted by users.
Most centralized approaches fail to achieve to comply with all the requirements. Distributed approaches seem promising but often need centralized management or infringe on the users privacy. If distributed solutions can be solve the management problem of low voltage grids and how they can coordinate their actions is the central question that arises.

1 Introduction

Carbon dioxide has become the main driver behind human made climate change, which poses a severe thread to society. The transport sector for now still relies mostly on oil. The switch to electrical vehicles although sensible, is only useful for reducing CO2 emissions if the electricity is generated by renewable energies. Unlike most other electricity sources, renewable energies are often integrated on the level of the low voltage grid.
Thus both electric vehicles and renewable energy sources(res) introduce massive changes in the, so far uncontrolled, low voltage grid. The grid not only has to be able to deal with the average load but more importantly the peak load that can occur within a day. Since PV production is dictated by the sun, PV production from distributed sources peak all at the same time and cause voltage peaks. The charging behavior of EVs is controlled by the owner of the car, however since most cars are used to commute to and from work, congested situation in which many people plug in their car at the same time, when they come home from work, can occur. For further increase of PV and EV in low voltage grids it might become necessary to reinforce the grid because of the increased peak loads. The goal of smart grids is to manage the low voltage grid to redistribute load more broadly in time to have a lower peak load. The lower peak load means that the existing grid can handle a higher penetration of renewable energies and electric vehicles.
In many smart grid solutions distributed intelligent agents and machine learning are being used. Distributed AI agents need some kind of reward function or goal for which to optimize. When it comes to reward mechanisms most research looks into using price structures to give an incentive to the agents to distribute their load to times when the price is lower.The overwhelming focus on monetary aspects has been criticized. Alternatively the grid could be seen as an common pool resource. This would mean that the reward function includes a direct feedback from the grid. This way the intelligent agent will have to strike a balance between charging the electric vehicle and not putting to much strain on the grid, instead of striking a balance between charging and cost. The rules about how and in which situation the grid constrains the behavior of the agents are set by the agents collectively through institutions.

In the following chapter a short introduction into the term smart grids is given, followed by a collection of related research. After this my research approach will be explained with a short introduction into background information about Markov decision processes, Q-learning and the concept of artificial Institutions in common pool resource problems. Based on this the model design will be explained. In the end there will be a time line and a list of the exam committee.

1.1 Smart Grids

The concept of smart grids is actually not very well defined. The term is more a problem description than a well defined concept. Most of the time it refers to the management of the production and consumption in the low voltage grid.
So far there was no need for managing the low voltage grid, since there was no production and the only demand was the average household electricity consumption. With the introduction of electric vehicles, the demand increases

significantly and becomes less predictable. With the uptake of PV production the consumer becomes also a producer of energy and the low voltage grid is no longer just the place where electricity is consumed but also produced. The additional demand and production is in size comparable to the already existing demand. The easiest but also most expensive solution is to reinforce the low voltage grid to handle this new situation. It is much cheaper to manage this upcoming demand and production, than to reinforce the grid.
The problem research has to deal with is how to manage a smart grid. One of the simplest ways is to use demand response and price structures. Users get feedback about their energy consumption and the current price. This way they will get a feeling for how much energy they use and hopefully adapt their behavior. This is however not guarantied to work and has some privacy issues.

Managing the smart grid through software shows also some problems. Managing it centrally is tainted by a feeling of loss of control for the user and is computationally heavy. Distributed solutions work with AI agents, which make some decisions about when to switch on different household appliances or charge a backup battery or electric vehicle. However if agents work independently of each other they either fail to find an optimal solution for the overall grid or need some outside incentive.

1.2 Smart Grid Related Research

This part is split into two research topics, the social aspect of smart grids and their technical implementation.

1.2.1 Social aspects of Smart grids

The social research is comprised of studies about acceptance of smart grids, management of smart grid through people called demand response, and the ethical problems that arise. [4] points out that the main obstacle in implementing smart grids is that most users have a feeling of loss of control over their own appliances.[6] on the other hand shows that, while there is a concern over privacy and loss of control, it is outweighed once the user realizes the benefits of smart appliances.
Demand response is the idea that by giving feedback about pricing, households are more aware of energy and are thus steered towards a energy conserving lifestyle.[4] notes that by giving active feedback about the households energy consumption, the power consumption was reduced by 9%. [5] talks about electricity consumption being contributed to lifestyle factors and that consumption can be reduced by "behavioral changes". The idea is that:"Technology and behavior thus have to complement each other."[5]. However others have raised moral concern about giving utility companies more power over appliances or giving them access to user data[7][8]. The fear is that utility companies or smart home appliance manufacturer could misuse or sell data to third party companies, or use price structures "... to influence consumer energy usage behavior"[7].

[3] points out that research in smart grids is often still stuck in a top-down centralized management way of thinking which might hinder the acceptance of

smart grids in society, and proposes to view smart grids more like an common pool resource. People would be more willing to accept smart grids if they are actively involved in coming up with rules how to manage the grid and they can adapt smart charging schemes to their needs. This would mean that the focus of research would change away from monetary incentives towards how to come up with rules to make the most of an existing infrastructure.Michael J. Sandel describes in his book ,"What Money can't buy" how monetary values can crowd out moral and ethical values and undermine community engagement in general. [8] has urged that ethical issues should not be features that can be added to existing smart grid concepts but rather central questions that should be addressed at the very beginning.

1.2.2 Technical aspects of smart grids

Although research in the technical management of smart grids can have very different scopes, the overwhelming focus is on electricity and the aim is almost always the minimization of cost and the reduction of peak load. [2] has pointed out that the focus on electricity might be too narrow-minded, and that a true smart gird would incorporate all kinds of energy. Although this point has validity, demanding smart grids that include heating and cooling of buildings might be premature since [3] has pointed out that "smart grid" is still mostly a buzz word.
In the future one part of the smart grid will be a smart home. [10] explores the possibility of using particle swarm optimization to manage energy demands with distributed energy resources within one home.
More interesting than the management of one smart home is the management of the interaction between the smart homes, distributed energy resources and the electric vehicles(EV). The negative effect of electric vehicles(EV) can be mitigated by the use of smart charging or can even be beneficial to the grid through vehicle-to-grid operation(V2G). Smart charging sees the electric vehicles(EV) as a load that has to be managed to prevent higher peak loads. V2G views the battery in the EV as an active part of the grid. It can be charged when there is too much supply by PV and discharged when there is little to no PV supply present[11].

1.2.3 Distributed Approaches to smart grid

Most approaches to find smart charging or V2G schemes involve some kind of centralized management [12][13][14]. [12] points out that any real time management of EV or smart grid has to be computationally inexpensive. In [12] the smart charging is based on a price structure incentivising people to charge their EV at times when there is otherwise little load in the grid. This still requires the user to make an active choice about in which time zone he or she would like his/her EV to be charged.
[14] proposes a solution, while centrally managed acknowledges the EVs as autonomous agents within the grid. These agents report their utility autonomously. Which car gets to charge how much is then managed centrally based on the utility each individual car reported. The paper also mentions the possibility that

agents could state incorrect values in order to seem more urgent to the system. In a congested situation in which there is too little capacity to satisfy all of the demand each agent would state the highest value to maximize its share. This is analogous to common pool resource problems, the agents try to maximize their own utility with disregard for the other agents and thus exhaust the system.
[15] has introduced a truly distributed method to deal with smart charging, where every agent makes autonomous decisions, yet the combined peak load is reduced compared to a uncontrolled charging scenario. Such solutions are multi agent systems, composed of several agents. The agents decide about how much to charge the car on behalf of a person, without actual interaction with the person. In order for the agents to make an informed decision, they have to interact with the environment and find optimal strategies for charging. This is normally done by Q-learning. However, Q-learning does not always converge when used in a multi-agent system. Since each agent has a different reward function there is rarely a policy found which maximizes the benefits of all agents [16]. Furthermore [17] points out that if the agents in multi agents systems appropriate a real and limited resource for humans, that the agents will contribute to the depletion of this resource if not properly restricted in their behavior.

[15] solves this by making the reward,an agent gets, a combination of utility gained from electricity usage, in household or through charging, and the cost of electricity usage. Because the cost of electricity is in the reward function, agents only charge up to a certain point, and it becomes more likely to charge when the price is low. This way the Q-learning algorithm in [15] distributes the load more broadly over time and converges to a better solution than in the uncontrolled charging scenario. However the agent does not get rewarded for stabilizing the grid, it gets rewarded for being most cost efficient, the positive effect for the peak load seems to be a side effect rather than the main goal of the research. This raises the question whether or not cost is the best indicator in the reward function to maximize the users benefit and the grids stability. On a broader perspective it raises the question how distributed AI systems, which do things on behalf of people should be managed.

1.2.4 Research Problem

In summary, the centrally controlled approaches to manage the low voltage grid fail to be responsive enough to deal with the fluctuating and changing nature of the low voltage grid and or infringe on the privacy of the user. Further make the feeling of loss of control and the privacy issues the uptake of such solutions rather unlikely. Distributed solutions have to be found to solve this very distributed problem. However most attempts at distributed solutions, need centralized management and or need to constantly communicate private data.

1.3 Research Question

The question to tackle is whether and how a decentralized approach can work in managing the smart grid. In order to answer this several sub-questions have to be addressed.

1. Can a artificial agent representing a single household, without the input from a central management system be beneficial to grid stability?

2. How can multiple of such household representing agents be coordinated in a decentralized way?

3. Can viewing the grid as a common pool resource and giving each household-agent the ability to contribute to the making of the rules and norms that govern the grid be beneficial to the overall grid stability?

1.3.1 Research Approach

1.3.2 Research methodology

In order to answer the research question a multi agent system has to be designed to intelligently manage the low voltage grid in a distributed way. The model will be represented by a markov decision process(MDP).
The environment of the MDP will be a low voltage grid. To have a realistic representation of a low voltage grid to work with, a low voltage grid has to be designed in theory, since there is no real grid information openly available. This theoretical grid then has to be transferred to a per unit model that can be interpreted by PYPOWER/MATPOWER. The agents in the MDP are software agents representing the different households. These software agents will control the charging and discharging of a battery. The state the agent is in is the SOC of the battery and the voltage at the grid point of the household.The action that the agent can take is to charge the EV at different rates within the next time step. This has an effect on the SOC and adds load to the grid point increasing the voltage deviation.

1. State
   (a) SOC
   (b) Voltage
2. Action
   (a) Charge EV [0,5]kW
   (b) (Vote to change Reward function

Within the experiments the voltage at the different grid points and the state of charge of the batteries will be kept record of and evaluated using R.

1.3.3 Addressing Research Question

To address the question an agent has to be designed, which can interact with the PYPOWER environment.A graphical representation to better understand how the software agent is to interact with the grid, can be seen in figure 1. This agent also has to have an internal policy to guide its behavior. The behavioral policy can be developed by the agent itself or can be imposed on the agent as an institutional policy. The institutional policy is elected by all the agents collectively.

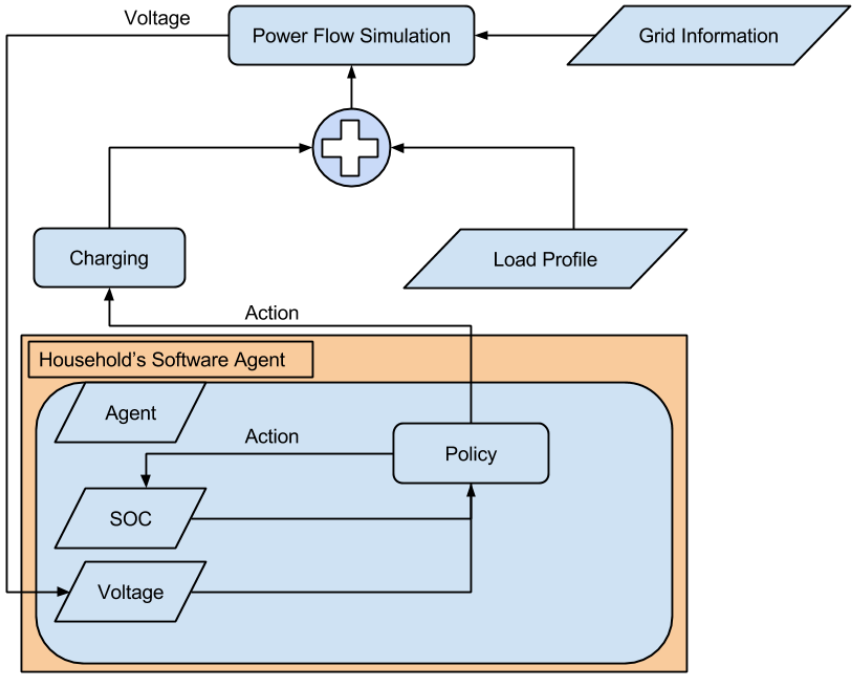

Figure 1: dsa

1.4 Research Tools

1.4.1 Powerflow simulation in MATPOWER/PYPOWER

MATPOWER/PYPOWER is a software package for power flow simulations. MATPOWER/PYPOWER is available for Matlab, octave and python. The software package models AC and DC electrical networks. The inputs to the model are the network topology, given in a n_bxn_b matrix, representing the impedance between the node points, and the n_bx1 complex load vector, representing the static loads at each node. The model is then solved for the voltage and angle for each node. The model works internally by defining so called cases. Within these cases each node, Transformer and household is represented as bus, with its own bus index. The household usage of electricity is represented by P and Q values at the respective bus index. The lines between the nodes are defined in per unit values of impedance, active and reactive resistance. In order to transfer real impedance values of lines to per unit values, each case has a defined base voltag(Vbase) and power base(SBase). From these a base impedance can be calculated, see equation 1.[22]

$$Z_{Base} = \frac{V_{Base}^2}{S_{Base}} \tag{1}$$

Given the base impedance each line impedance can be transformed to per unit by dividing it by the base impedance, see equation 2.

$$Z_{pu} = \frac{Z}{Z_{Base}} \tag{2}$$

Restrictions on the current that flows through the lines can be imposed. Further the model calculates how much energy has to generated by the generators, so that the only thing that has to be given is the load on the grid nodes.

1.4.2 Multi Agent System

Systems with more than one agent are called Multi Agent Systems(MAS) If there are more than one agent, there are different ways of dealing with coordinating the learning between the different agents. The easiest way is independent learning, in which the agents simply ignore the other agents. Coupled learning is when each agent tries to model and estimate the effect of the others on the system. In specific cases in which all the agents share the same reward function, sparse cooperation can be used to combine all the agents to one agent.[20]
For agents to further cooperate in multi agent systems they can vote on mutual laws. For this there has to be a set of options the agents can choose from and an internal function that ranks the different options. There are different voting mechanisms to combine the rankings the different agents made into one result. A simple majority vote takes the option that is most often chosen by the agents to be the best option. A Borda count takes not just the highest ranked option into account but also the whole ranking of all options. Each agent attaches a number to the different options. The highest ranked option gets the number k-1 attached, while k being the number of options. The option below that gets k-2 attached and so on until the least favorable option gets 0 attached. The options then sum up all the values attached to them by the different agents. The option with the highest number then becomes the mutually agreed upon option.[23]

1.4.3 Artificial Institutions

This section will explain what a common pool resource(cpr) is, how institutions can help solve cpr problems and how the idea of institutions can be applied to multi agent systems.
Common pool resources(CPR) experience properties of both public and private goods.
Similar to a public good, it is hard to exclude people from Common pool resources. An example for this would be the atmosphere. It is impossible to exclude people from the benefits of the atmosphere. Similar to private goods the subtractability of common pool resources is very high. The subtractability is not just how much value someone can extract from the resource but also if this extraction affects the resource and how much others can extract from it. This differs from a public good where the subtractability is low. For instance Wikipedia would be a public good and not a CPR because if you read an article on Wikipedia, it does not mean that others cannot read the same article at the same time. The access to the internet, which makes wikipedia possible, however is a common pool resource because if one person uses up all the bandwith to Wikipedia, it would deprive others of their access to Wikipedia.[1]

The problem of CPR results from these two properties. If an individual extracts value from the resource, the individual is the only one benefiting, yet the negative effect of resource depletion is shared by everyone. This incentivises individuals to extract as much as possible since their gain is greater than their damage and if they do not extract as much as possible they would only experience the negative effects. The behavior is the rational behavior of an individual but is irrational from a group perspective. This personally rational but group wise irrational behavior is known as the tragedy of the commons.
The tragedy of the commons is often solved by using institutions. Institutions are a set of rules and norms that all users in a system consent to. Rules need explicit, norms only implicit consent. By consenting to a set of rules, which restrict the user the commonly shared resource can be protected and it will not be depleted. This way the users can collectively arrive at a better situation than they could by individual reasoning[18].
[17] describes how software agents in real or virtual environments can experience the tragedy of the commons. Further software agents could autonomously appropriate a resource for a user. If these software agents do not cooperate or have rules to regulate their behavior they will contribute to the depletion of the resource. In human system a progressive tax on resource use is introduced to limit the users. The user only uses the resource up until the point when the tax burden outweighs the benefit[17]. An equivalent of such a tax mechanism for AI systems might be the reward function in Q-learning. It of course would not be in the form of money but in the form of an abstract negative reward for using the resource. The AI system would then use the resource until the reward it gains from the actual use outweighs the artificial negative reward introduced as an institution.

1.4.4 Borda count

A majority voting system only takes into account the top choice of voters. The Borda count also takes into account second and third choices of voters, this way it is more representative of the actual preferences of the voters.
Lets say there are k candidates $\Omega = \{\Omega_1, \Omega_2, \Omega_3, ...\Omega_k\}$. The voters then rank those k candidates from most to least favorable. Weights will be attached to the different candidates in the following manner. The best candidate gets the weight (k-1), the second best (k-2) and so on until the worst candidate receives 0. The choice with the highest weight count gets selected as the winner of the vote.[23]

1.4.5 k-means Clustering

The K-means clustering algorithm clusters n d-dimensional data points ,$x = \{x_1, x_1, ..., x_n\}$ into k clusters, S_i with mean values μ_i, so that the sum of square distances within the clusters is minimized.

$$arg\min \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2 \tag{3}$$

The algorithm consists of two steps which are repeated until the clusters no longer change. After initializing k centers of clusters randomly, all data points

get assigned to their nearest cluster center. Then the cluster centers get updated by calculating the mean of all the data points in the cluster.

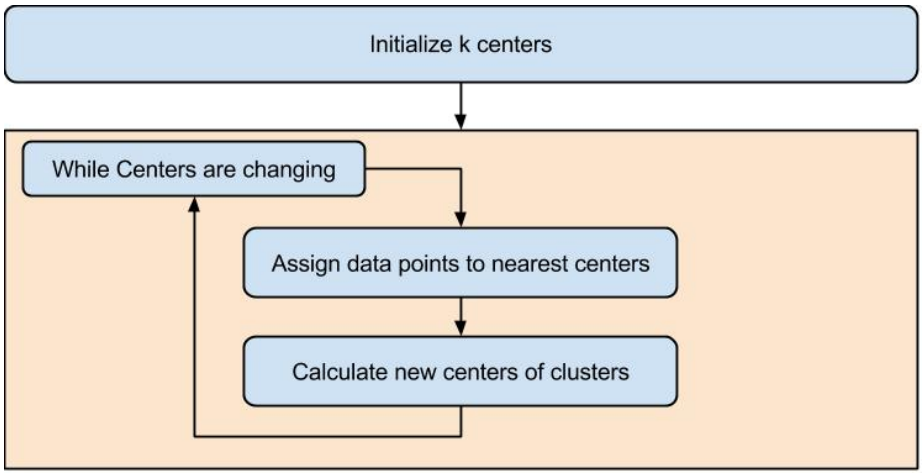

Figure 2: K-means Clustering Algorithm

2 Problem description

2.1 Stakeholders

2.2 Modeling approach

3 Model

3.1 Concept

First the components of the model will be explained. Then a overview of the processes that the agents go through will be given. The agents, how their rules and memory work and how the agents come up with rules collectively through voting will be explained afterwards.

3.1.1 Model components

The model consists of three parts , a physical layer represented by the Matpower model, software agents and an institutional layer, which sets the norms. The main active part in the model are the software agents. They interact with the physical layer by requesting a specific amount of power from the grid and receiving feedback about how much power they actually get. They also vote on rules and follow institutional rules as active rules. Agents have physical attributes, such as Battery , a node position and the corresponding voltage, which determine their place in the real world. The agents make decisions based on their rules. Their preference, memory and connections to others are used to come up with and vote on new rules.

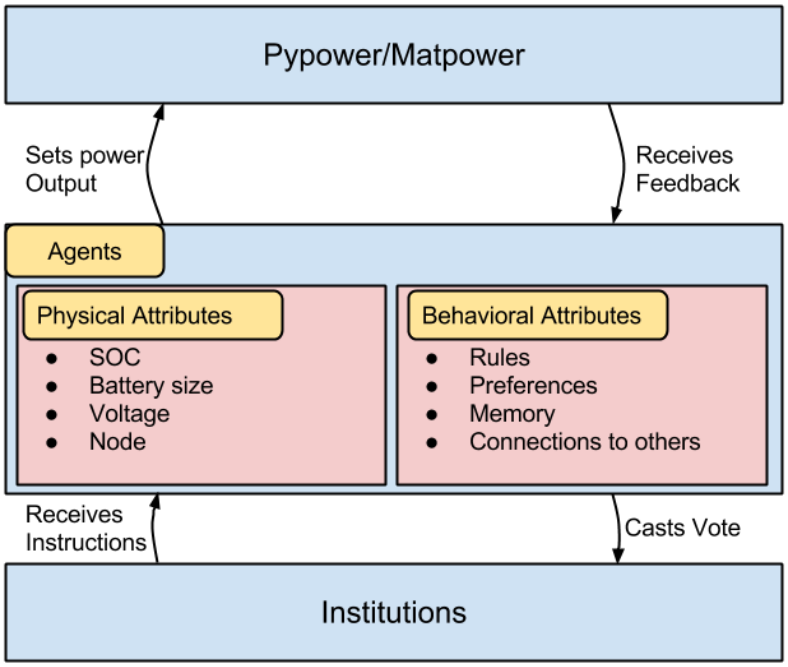

Figure 3: Model components

3.1.2 Agent gets up drinks a cup of coffee

Each agent goes through the following process, depicted below. For one week in intervals of 15 minutes, the agent chooses an action based on their internal rule. The matpower model then calculates a feedback which the agents internalize by updating their voltage and state of charge. Each week the agent evaluates the rule it has been living with based on how high their battery state of charge over this past week was, and commits the rule and an associated preference value to its memory. Afterwards the agent comes up with a new rule for the next week. When the memory of the agents is not yet filled the agents develop the rules themselves. Voting is also omitted when a rule that has been voted on, was not good enough to be in at least a certain percentage of the agents memories. This percentage is called the institutional success rate. For the voting the agents report their ranked memory. The vote itself is done by borda count and then clustered. The agent can come up with a new rule by either learning based on their memory, copying from the best performing other agent or randomly selecting a new rule. The new rule is then set as the new active rule in the agent, and the process is started over again.

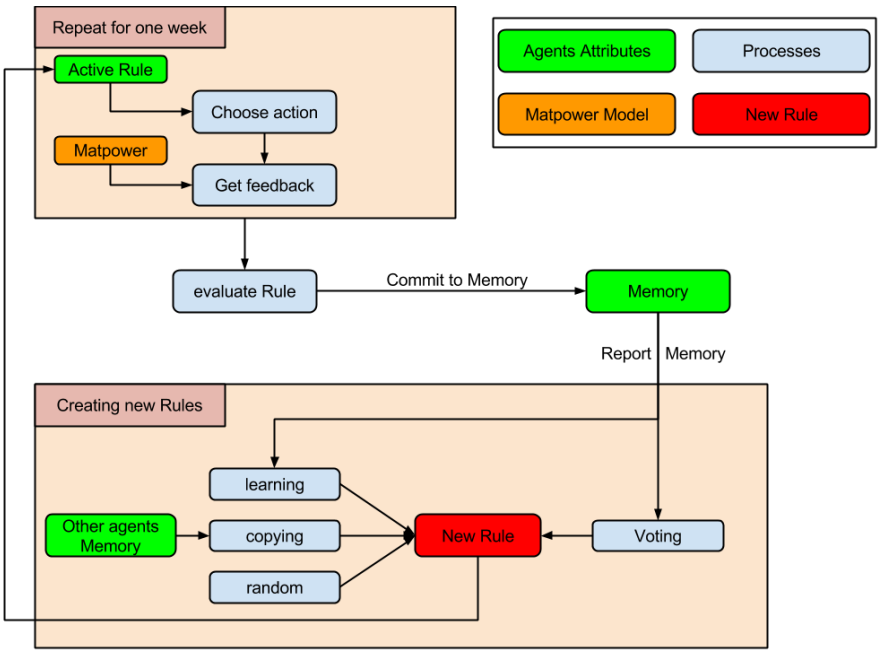

Figure 4: Model processes

3.1.3 Rules

Rules in the agents govern their behavior. They return a corresponding action given a specific state. The state of an agent is its state of charge(SOC), the voltage at its node and the current time. Rules basically map a specific combination of these variables to an action. The action is how much the agent will

charge in the next time step.

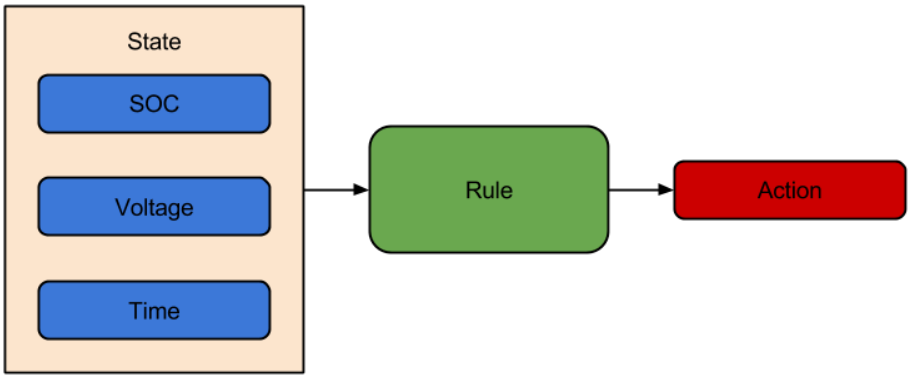

Figure 5: The function of rules

Rules can either be based on the current time and state of charge(soc) or on the voltage at the grid point and the state of charge(soc). Voltage bases rules are indicated with a 1 and time based rules are indicated by a 2.

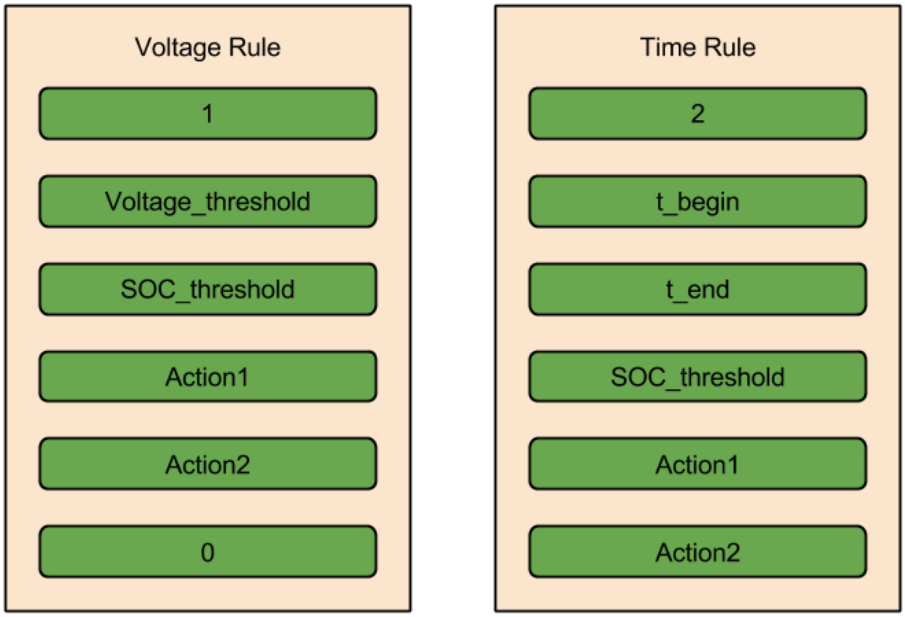

Figure 6: The structure of rules

A voltage rule has a voltage and a state of charge threshold and two actions - action1 and action2. If the voltage of the agent is below the voltage threshold defined in the rule and its state of charge is above the minimum state of charge defined in the rule, the agent decides to take action1. If this condition is not true the agent takes action2. Voltage rule: (State=Blue, Condition = Green, Action = Red)

```
If (Voltage < Voltage_threshold) & (SOC > SOC_threshold)
    then do Action1
else
    do Action2
```

A time rule has a similar structure. It contains two times, t_{begin} and t_{end}, to define a time interval and a minimum state of charge ($soc_{threshold}$). If the agents state of charge is above the minimum, defined in the rule, and the time is within the specified time interval the agent takes action1, otherwise the agent takes action2. Time rule:

```
If (time_end < Time < time_end) & (SOC > SOC_threshold)
    then do Action1
else
    do Action2
```

3.1.4 Memory

Each agent has a memory to keep track of past rules and how good the agent thinks this rule was. The memory is needed for the learning and for being able to vote. The memory consist of 5 past rules and associated preference values. The rules are kept in memory based on merit. Each time an agent commits a rule to its memory, it calculates the average state of charge of its battery under this rule. This value is the preference value for this rule. If the rule is already in the memory the preference value becomes the average of the two preference values. If the rule is new it replaces the worst rule in the memory if its preference value is higher than the preference value of the worst rule in memory. Afterwards the memory gets sorted from best to worst.

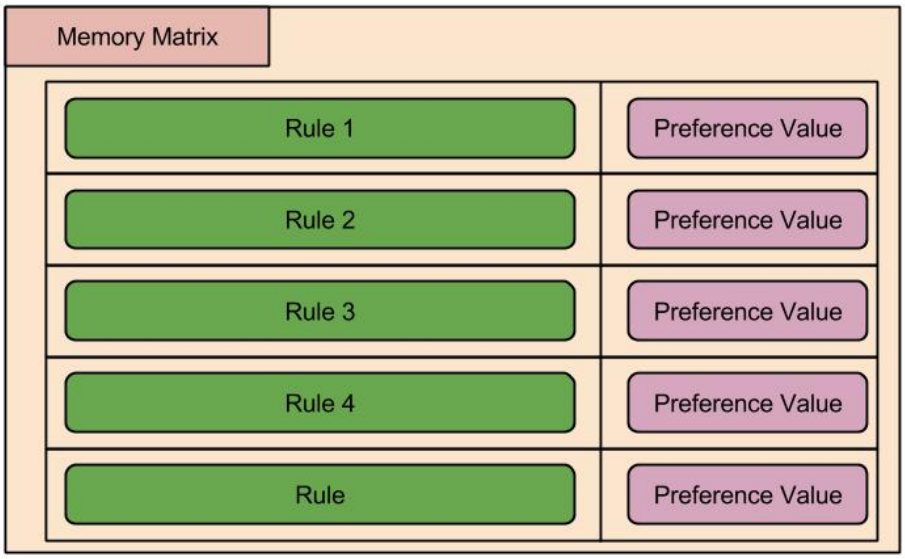

Figure 7: The structure of Memory in the agents

3.1.5 Voting

The agents report their memory as a vote. Borda count is used to duplicate the rules in the matrices based on their rank. The top ranked rule in one memory matrix will be duplicated 4 times, the second best rule in the matrix will be duplicated 3 times and so on. Duplicated this way the votes are then all combined into one election matrix. The decision whether a new rule should have a voltage or time condition can be made by simple majority vote. The specifics of the conditions will be determined by clustering the rules into 3 clusters. The center of the cluster containing the most votes will be used as the specific condition for the rule.

3.1.6 Model assumptions and choices

1. **General assumptions**

   (a) **The global time is kept track of by the ticks. One tick is 15 minutes**
   To make it easier to keep track of the time of day the model always starts at midnight. This way the modulo of the global tick by 96 indicates the time within the day.

   (b) **The agents can either be at home or not at home. The mean arrival/leaving times of all the agents are gaussian distributed. The mean battery drain of all the agents is gaussian distributed with mean equal to the average battery drain.**
   Given the central limit theorem the assumption that the mean of a variable over multiple agents is gaussian distributed means that the variable has the same distribution for each agent.

   (c) **Power consumption exceeding this voltage level will be capped.**

   The agents have to get feedback from the MATPOWER/PYPOWER model. In order for this to happen current and or voltage constraints have to be imposed on the model. Based on the maximum current in one line the maximum voltage that can appear can be derived.

2. **Assumptions/Choices about Rules**

   (a) **The number of rules that dictate the behavior of the agents will be fixed to one rule**
   Otherwise a meta rule is needed to decide which rule should be used. The one rule has to give an action in any possible situation. Not charging is still an action. This means the rule will be a if-else structure with 2 actions , an action which will be executed when the condition is fulfilled and one default action. The condition in the rule can only be dependent on variables the agent can observe - its SOC, Voltage and the time. Since the state of charge is the main interest of the agent it should always be in the rule. Two kinds of rules, soc-time-condition and soc-voltage-condition, will be possible.

(b) **Each agent keeps 5 previous rules and corresponding preference values in memory**
For simplicity sake the number of rules is kept fixed over time.

3. **Assumptions/Choices about voting**

(a) **Borda count will be used for the voting**
The voting mechanism should find the solution that is most prefered by all the agents. Simple majority rule often fails to do this. Borda count makes the voting more representative of the actual preferences of the agents.

(b) **K-means clustering algorithm will be used to cluster the rules into 3 clusters**
Since there are too many possible rules and too few agents to vote on them, simply counting will fail. Very similar rules would be counted as completely different rules. This issue will be fixed by clustering the rules into several clusters of rules. The k-means algorithm needs the number of clusters as input to be fixed.

(c) **Agents cannot vote and come up with rules at the same time**
In order for agents to vote they need to be able to rank their choices from worst to best. This ranking can only happen if the agents can evaluate the choices with some metric. This evaluation can either come from experiencing the rule and observing how well the agent performs under this rule or by a fixed evaluation function. An evaluation function can only work if the agent can beforehand estimate the effects of a rule on their performance. Since there are several agents interacting, estimating beforehand the effect of a rule is unlikely to work. This means that in order to give a meaningful preference value to a rule the rule has to be experienced. Agents can only evaluate and thus vote on the rules that they have followed in the past.

(d) **Voting will be omitted for one week if the institutional rule is not good enough**
Once the initial exploration phase is over and the agents have filled their memory and the voting has begun, the set of possible rules the agents can come up with by voting is limited. If a rule has been voted on and this rule then performs very poorly it might be because the set of rules it was selected from was too limited to contain a good solution. It would then only be reasonable to let the agents explore the complete set of rules again to expand the set of possible rules.

(e) **The first five weeks the agents only develop the rules themselves**
This is just to fill their memory. The voting with borda count makes very little sense if there are only one or two rules in the memory to be ranked.

3.2 Implementation

The model was written in Pyhton because the language is easy to use and has a full implementation of MATPOWER, called PYPOWER, and the k-means

algorithm. A detailed description of the model implementation and pseudo code can be found in Appendix C .

3.3 Verification

Two parts of the model have to be verified. That the multiagent system itself works and that the PYPOWER model represents reality in the intended way. For the multiagent system, the agents individual behavior and their collective behavior, interacting with each other was tested. For the PYPOWER model the constrain mechanism was tested.
An extensive list of all the tests that were done can be found in Appendix B.

4 Experimental

4.1 Basic setup

In this chapter the basic experimental setup will be explained. First the input, that is needed to run the model and the time frame for the model is stated. Then the data that will generally be output by the model will be explained and finally how this output will be interpreted will be explained.

4.1.1 Input

Each run of the model will consist of six months of simulation in 15 minute intervals. The global variables that are fixed over all the experiments and have to be provided are the following.

1. `time`
2. `average_arrival_time = 72 (6pm)`
3. `average_leaving_time = 32 (8am)`
4. `sd_average = 1 (quarter of an hour)`
5. `average_battery_drain = 10`
6. `sd_average_battery_drain = 2`
7. `average_soc = 20`
8. `sd_average_soc 2`
9. `soc_max_gloabl = 40`
10. `action_options = [0,1,2,3,4,5]`
11. `t_begin_options = [1,2,.....,96]`
12. `t_end_options = [1,2,.....,96]`
13. `soc_threshold_options = [ 0., 5., 10., 15., 20., 25., 30., 35., 40.]`
14. `voltage_options = [ 0.96, 0.97, 0.98, 0.99, 1. ]`

There are other global variables which will be initialized to an empty value but will emerge from the model.

1. `institutional_rule`

There are also global variables which will be defined within the specific experiments and are therefore mentioned later in the experimental part. Those variables mostly deal with how the agents come up with rules and how they vote on rules.

1. `random_change`
2. `copy_best_change`
3. `learn_change`

4. `institutional_success_rate`

5. `copy_all`

6. `majority_vote`

4.1.2 Output

Part of the output of each experiment has to be the the input variables that defined the exact parameters. The output is also defined by the question that needs to be answered. What is of interest in the model is how and if the model solves the common pool resource problem. The tragedy of the commons in this case is that some agents are not able to charge sufficiently. Therefore the state of charge of the battery over time is kept track of for each agent.The institutional rule, how quickly it changes over time and how frequently it fails gives insight into how the agents try to solve the tragedy of the commons. The exact structure of the output and how it will be done in detail will be explained in the appendix.

4.2 Performance indicators

The output will be evaluated using the statistical tool R. There are some basic statistics that will be used , such as the mean, median and spread of variables. More interesting is how the state of charge of all the agents will be used to interpret a model run or a specific rule as a success or failure. Simply looking at the mean state of charge over all the agents is not sufficient to indicate if some agent failed. In particular the state of charge of the agents which in the unconstrained test had the lowest average state of charge are of most interest. If a rule itself is a failure will therefore be judged by how the state of charge changed over a week. If the state of charge for the weakest agent never reaches 100% it is considered a "weak failure". If the state of charge for the weakest agent drops to 0 it is considered a "strong failure", because it indicates that there was not enough time to recharge the battery to a needed minimum. The graph below depicts a weak and strong failure. The brown line depicts the state of charge of an agent on level 5. The agent shown by the brown line has a higher average state of charge than the agent shown by the red line. However the agent on level 6 is arguably doing better, because its state of charge never drops to 0, meaning that there is always sufficient charge in the battery to commute to and from work. The model setup is considered a failure based on the frequency of failure of the institutional rules.

4.3 hypotheses

1. **Behavioral Hypotheses**

   (a) **By increase the copying behavior the model performs worse** Rules are evaluated by each agent individually. Rules which are best for the individual are bad for the system as a whole. Those rules are problematic. When more agents copy the rules from the best performing agent these problematic rules become more prevalent and thus the model performs worse.

(b) **By increasing the learning behavior the model performs worse**
By learning the agents only interpolate their memory but not extrapolate to new rules. This way the span of rules within the agents gets limited and thus also the overall pool of rules is limited. A limited set of rules makes it harder to find overall good rules and thus the performance of the model will suffer.

2. **Institutional Hypotheses**

(a) **The model performs better when using borda count compared to majority voting.**
Majority voting only takes the top ranked rule of each agent into account. Borda count takes a ranked list of rules into account and is therefore more representative of the actual preferences of the agents. This way rules which are not best for most but good for all are more likely to be elected and thus the model performs better.

(b) **A high institutional success rate makes the model unstable.**
The institutional success rate is the percentage of agents which have to be satisfied by an institutional rule to continue the voting. If the percentage becomes too high no rules will be able to be good enough. The model will jump between institutional and personal rule and not be stable.

(c) **A low institutional success rate makes the model perform worse.**
The institutional success rate is the percentage of agents which have to be satisfied by an institutional rule to continue the voting. If this rate is too low the model will continue voting even if the set of rules to vote on is too limited to come up with an adequate solution. The model will perform worse because what is seen as satisfactory is set too low.

(d) **Majority voting fails to overcome a greedy rule.**
If a greedy rule, charge as much as you can whenever you want, is already introduced to the system as an institutional rule, there is a certain proportion of the agents which will vote for this rule, because of their advantaged situation. In simple majority voting it is very unlikely that a bigger group of agents can agree on one rule to favor, than are already in favor of the greedy rule simply by their position.

(e) **Borda count can overcome a greedy rule.**
If a greedy rule, charge as much as you can whenever you want, is already introduced to the system as an institutional rule, there is a certain proportion of the agents which will vote for this rule, because of their advantaged situation. Unlike majority voting, borda count merely needs most agents being content with a rule to make it an institutional rule.

3. **Neighborhood Hypotheses**

(a) **Grouping agents in neighborhoods improves the performance of the model.**

Rules are evaluated by each agent individually. Rules which are best for the individual are bad for the system as a whole. Those rules are problematic. Problematic rules can spread through the set of agents by copying. Grouping the agents into neighborhoods contains problematic rules to the neighborhood, which makes them less likely to become institutional rules and thus improves performance.

(b) **Vertical grouping creates better results than horizontal grouping.**
Rules are evaluated by each agent individually. Rules which are best for the individual are bad for the system as a whole. Those rules are problematic. Problematic rules can spread through the set of agents by copying. Horizontal grouping is grouping the agents which are most alike in one neighborhood. Those who are closest to the source get grouped in one neighborhood and those who are furthest away from the source get grouped into another neighborhood. Horizontal grouping results in homogeneous groups. Problematic rules can spread more easily in homogeneous than in heterogeneous groups, therefore horizontal grouping will create worse outcomes because of more homogeneous neighborhoods.

4. **Contextual Hypotheses**

(a) **Emerging rules are only functioning in the particular infrastructure they emerged in.**
Even small changes in infrastructure have a significant influence on how a load is affecting the voltage on a grid point. This has an effect on whose demand is getting cut and therefore on how the different agents perform in the model. So rules which work in one setting, may creating bottlenecks in other setting.

(b) **Increasing the variance in base load results in more random rules.**
Randomness in the base load affects the effect a rule has on the performance of the agent and increases therefore the potential difference between how good the rule is and how good the agent thinks the rule is. If the variance in the base load is increased the evaluation of rules becomes less based on merit and more random. If the evaluation of the rules by agents becomes random the rules that get elected also become more and more random.

4.4 Experiments

In this chapter the experiments needed to test the hypothesis will be introduced. Each section begins by stating which hypotheses are addressed by the experiment. Then the fixed input and the behavior space over which the experiment is carried out will be explained. At the end of each section what the output should look like to accept or refute the hypotheses will be stated. Experiment 5 is an exception to this structure.

4.4.1 Experiment 1

This experiment addresses the neighborhood related hypotheses, 3a and 3b. All the parameters that do not affect the neighborhood formation are kept fixed.

1. `random_change  = 0.3`
2. `copy_best_change = 0.5`
3. `learn_change = 0.2`
4. `majority_vote = False`
5. `institutional_success_rate = 50 %`

The variable of interest is the variable which defines how the neighborhoods are set up.

1. `copy_all = [All, horizontal, vertical]`

To accept the hypotheses a significant increase in the number of failures in a run should occur when copying or learning is increased. The two types of failures are an agent not reaching 100 % charge within one week, and an agent not being able to charge sufficiently and its battery charge dropping to zero.

4.4.2 Experiment 2

This experiment addresses the first three institutional hypotheses, 2a, 2b, and 2c. The behavior of the individual is not important to those hypotheses, therefore the parameters influencing the individual behavior and their connections are fixed. The parameter for copy_all is based on the results of experiments one.

1. `random_change  = 0.3`
2. `copy_best_change = 0.5`
3. `learn_change = 0.2`
4. `copy_all = best(copy_all)`

The variables which are of interest are the variables which influence how the agents come up with rules collectively.

1. `majority_vote = [True, False]`
2. `institutional_success_rate = [10 %, 30 %, 50 %, 80% ,100%]`

Similar to experiment 1, the hypotheses will be accepted based on increase/decrease of the number of two failures.

4.4.3 Experiment 3

This experiment addresses the behavioral hypotheses, 1.a and 1.b. Since the parameters which do not influence individual behavior, are not essential to the hypotheses, they are kept fixed. The parameters for copy_all and institutional_success_rate are based on the results of experiments one and two.

1. `majority_vote = True`
2. `copy_all = best(copy_all)`
3. `institutional_success_rate = best(insitutional_success_rate)`

The variables which are of interest are the variables which influence the individual behavior of the agents.

1. `random_change`
2. `copy_best_change`
3. `learn_change`

Similar to experiment 1, the hypotheses will be accepted based on increase/decrease of the number of two failures.

4.4.4 Experiment 4

This experiment addresses the last two institutional hypotheses, 2d and 2e. The two hypotheses are a comparison between majority and borda count, therefore all the other parameters are fixed. Further the initial active rule of each agent has to be set to a greedy rule.

1. `random_change  = 0.5`
2. `copy_best_change = 0.3`
3. `learn_change = 0.2`
4. `active_rule = [1,1,20,5,5,0]`
5. `copy_all = best(copy_all)`
6. `institutional_success_rate = best(insitutional_success_rate)`

The only variable parameter is the majority_vote parameter.

1. `majority_vote = [True, False]`

Testing the two hypotheses consists of two parts. The first part is to show that there is a difference in number of failures between majority and borda count, to show that borda count is better at overcoming greedy rules. The second part is to show how much the rules the model comes up with differ from the greedy rule. A greater distance of the institutional rule from the initial greedy rule is expected for the borda count compared to majority voting.

4.4.5 Experiment 5

This experiment addresses the first contextual hypothesis, 4a. This experiment does not really have an behavior space, therefore all the parameter are kept fixed.

1. `random_change  = best(random_change)`
2. `copy_best_change = best(copy_best_change)`
3. `learn_change = best(learn_change)`
4. `majority_vote = best(majority_vote)`
5. `copy_all = best(copy_all)`
6. `institutional_success_rate = best(insitutional_success_rate)`

The difference between the runs is the grid structure that is used. The model will first be run with the default grid, grid1. The best performing rule will be extracted from this model run. This rule will be used as the active rule for all the agents in the second run. The second run will be done without the development of new rules to just see the effect of this rule. To accept the hypothesis there has to be a significant difference in the number of failure in the second run compared to the first run with the default grid.

4.4.6 Experiment 6

This experiment addresses the second contextual hypothesis, 4b. This experiment is to introduce noise to the system. The other parameters will be set to default values.

1. `random_change  = 0.3`
2. `copy_best_change = 0.5`
3. `learn_change = 0.2`
4. `majority_vote = False`
5. `copy_all = best(copy_all)`
6. `institutional_success_rate = best(insitutional_success_rate)`

Different noise levels will be used to explore their effect on how well the agents deal with background noise. The best rules that the agents voted on under noisy situations will then be used in a second run, in which , similar to experiment 5, the agents no longer develop new rules but live with this run for a longer time.

1. `Noise_level = [10%, 30% , 50 %, 80%]`

To accept this hypothesis there has to be a significant increase in the performance difference between the two consecutive runs for increasing noise levels. This would mean that the rules were selected more randomly and therefore performed worse when they were run for longer times.

5 Results and Discussion

6 Conclusion

6.1 Model conclusion

6.2 Stakeholder perspective

6.3 Insight for software agent systems for resource appropriation

A PYPOWER Model

To have some grid to work with a low voltage case has to be created from the ground up, since MATPOWER/PYPOWER does not include any generic low voltage cases. Since there is no data on a real grid, reasonable values have to be assumed based on literature. To create the model the size and topology of a grid has to be set, impedance values for connections and load values for the buses have to be found and all the data has to be converted to per unit values. Based on literature[25] a radial network with a central transformer with a power of between 100 and 500kVA is chosen. There will be 24 homes, base load of 6kVA, connected to the transformer with 4 lines. Between each household there will be a line of 100 meters. This way the total base load is about $S_{base} = 144kVA$ and the total length of each cable is 600 meters. The base voltage of the grid is 400V.

$$Z_{base} = \frac{V_{Base}^2}{S_{Base}} = \frac{(400V)^2}{144kVA} = 1.11\frac{V}{A} \tag{4}$$

The low voltage cables are taken from real world cable data and have the following values[24].

1. A= 500 mm^2
2. l = 100m = 0.1km
3. R = 0.1590 $\Omega/km * 0.01km = 0.001590$
4. X = 0.0814 $\Omega/km * 0.01km = 0.000814$

The per unit values for the lines therefor are $r = \frac{R}{Z_{Base}} = 0.001431$ and $x = \frac{X}{Z_{Base}} = 0.00073226$ The load profiles are based on normalized profiles with an average usage of 8000kWh a year, see figure 9.

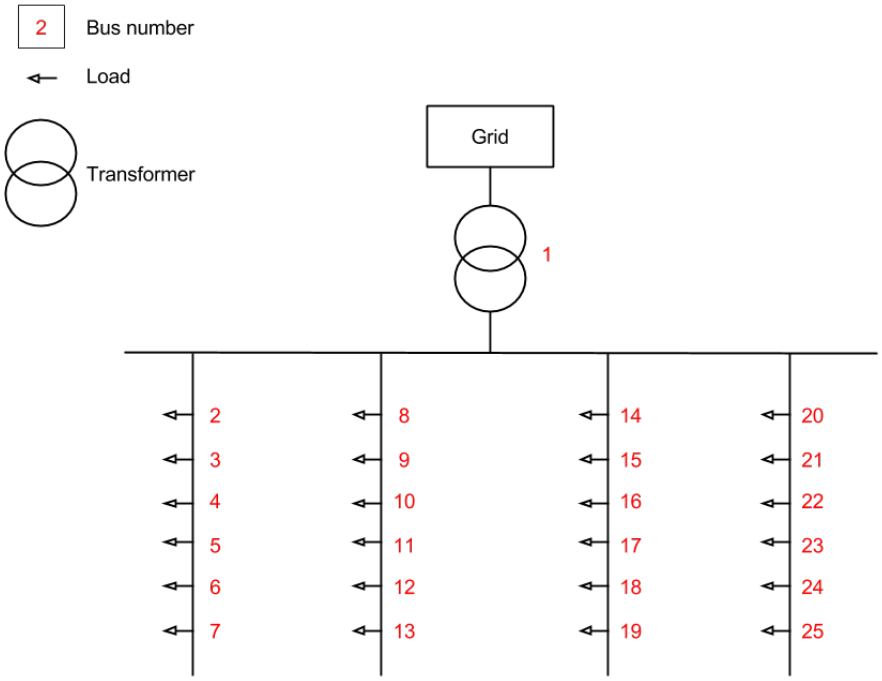

Figure 8: Topology

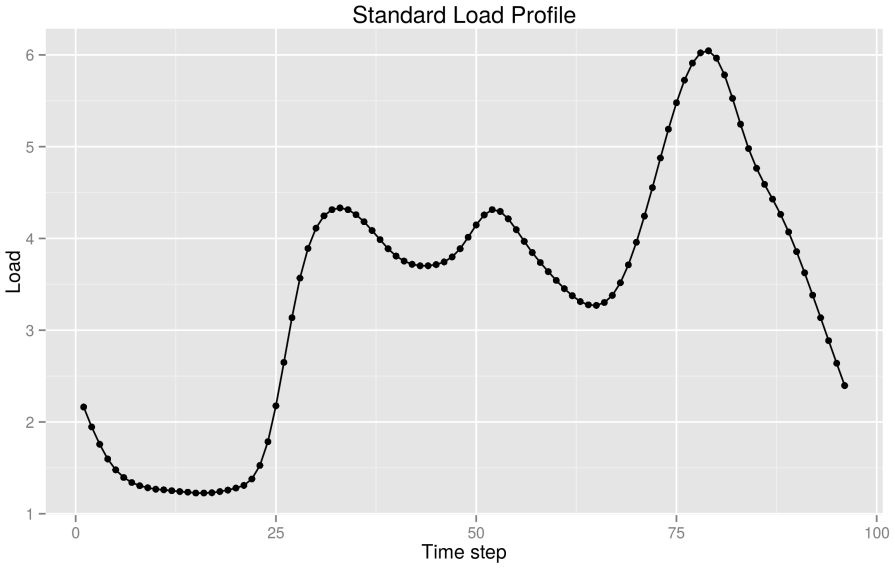

Figure 9: Load Profile

B Verification

The verification of the model is done in three steps. The functions that the agent can execute by itself are verified, then the functions that only work by interacting with other agents or other parts of the model are verified and finally the pypower model, in particular the constraints.

B.1 Verification of Functions

Each function and how it was tested will shortly be explained.

B.1.1 Verification of simple functions

1. `Init_simple_test:`

   Simple test if an initialized agent is set to the correct state of charge and time.

2. `update_situation:`

   Updates the count values and sets a new preference value.
   Agent was initialized with a specific state of charge. State of charge was reset to different values and update situation was called every time. The preference value was calculated by hand and compared to the preference value the agent held

3. `Reset_values:`

   Resets the count values (count_steps,count_soc,preference) to 0.
   Agent was initialized and updated the state of charge was then updated several times and the reset values function was called and the count values were checked to be 0.

4. `power_used:`

   Receives an amount of power and updates the state of charge of the battery accordingly.
   Agent was initialized with a specific state of charge, power used was called and the new state of charge of the agent was compared to the state of charge that was calculated by hand. Agent was also initialized to a state of charge close to full charge, power used was called and it was checked if the state of charge would be bound to the maximum soc.

5. `get_action:`

   Returns the action, the agent will take based on the agents state.
   Agent was initialized with specific state (voltage,time and SOC). Agent was given 4 different voltage rules,differing in their voltage and state of charge threshold, resulting in different actions. The actions the agent was returning was compared to the action it was supposed to take. The same was done with 4 different time based rules.

6. `update_memory:`

   Takes the active rule and the preference value and adds it to the memory.
   Agent was initialized and the memory of this agent was set. An active rule,

which was new to the memory was set. Two test were made, one where the preference value for the active rule was set in such a way that it was worse than the worst rule in the memory and one where it was better than the worst preference value in the memory. The update memory function was called and it was checked if the new rule was in the correct spot in the memory. Then a active rule was set which was already known to the memory. The update memory rule was called and it was checked if the preference value was reset to the average of the two preference values.

7. `weighted_time_rules:`

   Creates a new time based rule based on the rules and preference values in the memory.
   Agent was initialized. Memory only consisting of time rules was set and the weighted time rules function was called. A new rule was calculated by hand and compared to the rule that was created by the function. The same procedure was repeated with a mixed memory of time and voltage based rules.

8. `weighted_voltage_rules:`

   Creates a new voltage based rule based on the rules and preferences in the memory.
   Same procedure as the weighted time rules function.(see above)

9. `Arrive_at_home:`

   Changes the at_home variable to True and discharges the battery.
   Agent was initialized and its at_home variable was set to False. The arrive at home function was called and the at_home variable was checked to be True. Further the state of charge was set to "-2" and the arrive at home function was called . The state of charge of the agent was checked to be reset to 0.

B.1.2 Verification of interacting functions:

1. `copy_best(all):`

   Searches through all other agents and copies the rule with the highest preference value.
   24 Agents were initialized and each agent expect one was given the same memory. The one exception was give a memory with a rule with a preference value that was higher than the others. The copy best function was called from one of the other agents. The new active rule of this agent was compared to the one rule with the highest preference in the special agent.

2. `copy_best(vertical):`

   Searches through all other agents, in the same branch, and copies the rule with the highest preference value.
   24 agents were initialized and each agent except for two were given the same memory. The copy_all variable was set to "vertical". There is one original agent on which the copy_best function will then be called on. One exception was in the same branch as the original agent and given a better rule. The second exception was given a global best rule in its memory and

placed outside of the original agents branch. The two exceptions are on the same level but in different branches. The agent should when the copy best function is called on it find the best rule in its branch but ignore the global best rule.

3. `copy_best(horizontal):`

   Searches through all other agents, on the same level, and copies the rule with the highest preference value.
   24 agents were initialized and each agent except for two were given the same memory. The copy_all variable was set to "horizontal".There is one original agent on which the copy_best function will then be called on. One exception was on the same level as the original agent and given a better rule. The second exception was given a global best rule in its memory and placed outside of the original agents level. The two exceptions are on the same branch but in different levels. The agent should when the copy best function is called on it find the best rule in its level but ignore the global best rule.

4. `do_interaction:`

   Receives a pypower model as input and adds power to the node of the agent based on the active rule of the agent.
   24 Agents were initialized. Each agent was given a different node but the same voltage, active rule, state of charge and time. A pypower model was initialized to the same time as the agents. Each agent called the do interaction function on the pypower model. A theoretical load vector was calculated by hand and compared to the load vector in the pypower model after the interaction with the agents.

5. `get_feedback:`

   Receives a pypower output structure as input and updates the state of charge of the battery based on how much power was actually used.
   24 Agents were initialized. The 24 Agents were given the same state and the same active rule. A pypower model was initialized and its voltage constraints were removed to make the amount of power used predictable. Each agent was interacting with the pypower model and the get feedback function was called for each agent. A new state of charge was calculated by hand and compared to the new state of charge of the agents after the function call.

6. `Voting:`

   Takes the memory of all the agents, combines them into an election matrix ,taking the individual memory matrices as borda count votes.
   24 Agents were initialized, each agent was given the same memory. Then the voting algorithm was called. The number each rule should be represented in the election matrix was calculated by hand and compared to the actual number of occurrences in the election matrix. Further the total number of voltage rules and the total number of time rules was calculated by hand and compared to the actual number of time and voltage rules in the election matrix. The k-means clustering algorithm was not tested since it is a function of the numpy python package.

B.2 Verification of Pypower model

The functions that are to be verified are the following.

1. The dispatchable loads are
   (a) changed in such a way that the voltage at each node is always above a specific voltage threshold
   (b) not changed if the voltage is already above the specific voltage threshold
   (c) changed first at the nodes with the highest voltage in the branch

To verify this two pypower models are initialized one with and one without a voltage constraint. Both models are run 20 times. Each time a random dispatchable load is applied to the different node points. Within each run the dispatchable loads of both models are set to the same values. Among other things the voltage is kept track of and compared. The two graphs below show the minimum voltage in each branch. The red line indicates the voltage threshold. In the top graph the unrestricted load causes the voltage to drop below the voltage threshold, in the bottom graph the minimum voltage seems to be bound to the red line while the points above the line appear to be unchanged. This is confirmed by looking at the dispatchable load in the branches which had a minimum voltage above the threshold in the unconstrained case. In those cases the dispatchable loads of both models were exactly equal. To check if the order in which the loads are cut is correct, the percentage of change between the constrained and unconstrained case is checked for each branch. If at any node the change is more than 0 and below 100 percent, this node has to be either the very lowest in its branch , meaning furthest away from the source and therefore lowest voltage, or all the percentage difference that come after it in the branch have to be 100 percent. All the test are run in the "pypower_constraint_test.py" file.

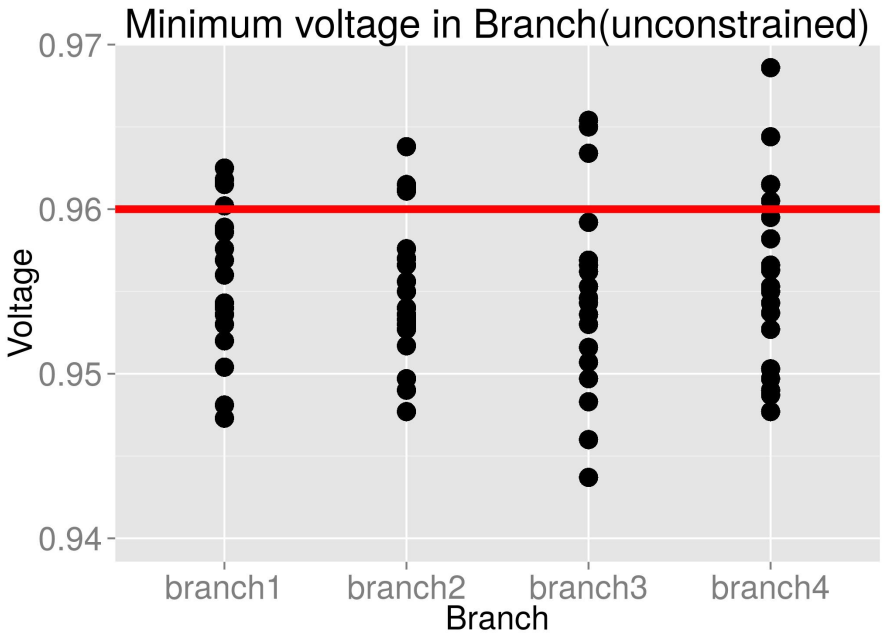

Figure 10: Pypower model without constraints

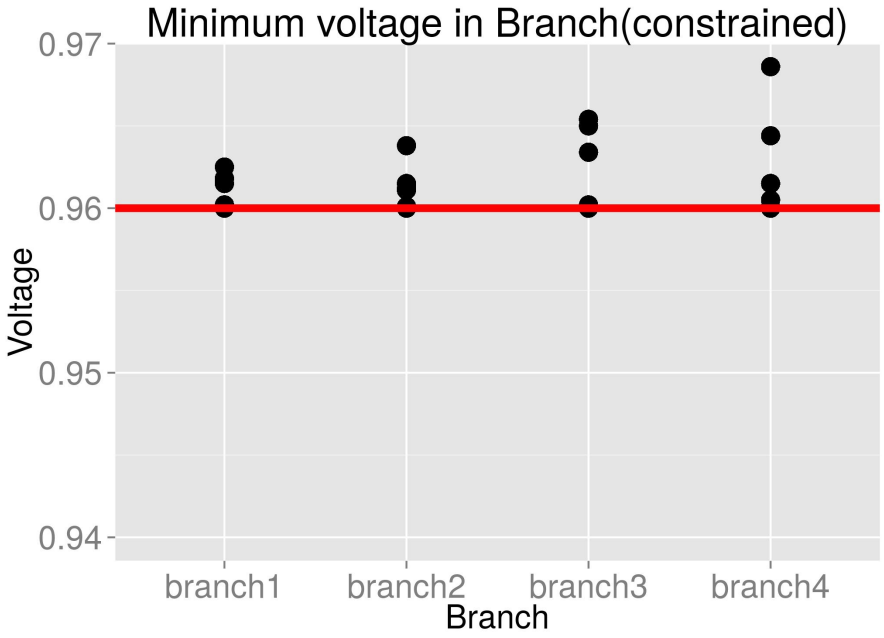

Figure 11: Pypower model with constraints

C Pseudo Code

1. Agents have:
   (a) active_rule
   (b) Memory (7,5 Matrix)
   (c) arrival_time
   (d) leaving_time
   (e) at_home (true/false)
   (f) average_battery_drain
   (g) sd_battery_drain
   (h) SOC
   (i) SOC_max
   (j) Voltage
   (k) Node
   (l) count_steps
   (m) count_soc
   (n) preference
   (o) copy_all
2. Global Variables:
   (a) average_arrival_time
   (b) average_leaving_time
   (c) sd_average
   (d) average_battery_drain
   (e) sd_average_battery_drain
   (f) average_soc
   (g) sd_average_soc
   (h) soc_max_global
   (i) random_change
   (j) copy_best_change
   (k) learn_change
   (l) Action_options
   (m) t_end_options
   (n) t_begin_options
   (o) soc_threshold_options
   (p) voltage_options
   (q) institutional_rule
   (r) institutional_success_rate

In the setup phase of the model the time gets set to 0, the agents are created and initialized and the matpower model is created.

Figure 12: To-setup

During the initialization the agents get assigned their node in the grid and the voltage is set to 1. Each agent then initializes their preference to the current rule to 0, sets leaving and arrival times, sets up their battery and creates an initial rule and an empty Memory.

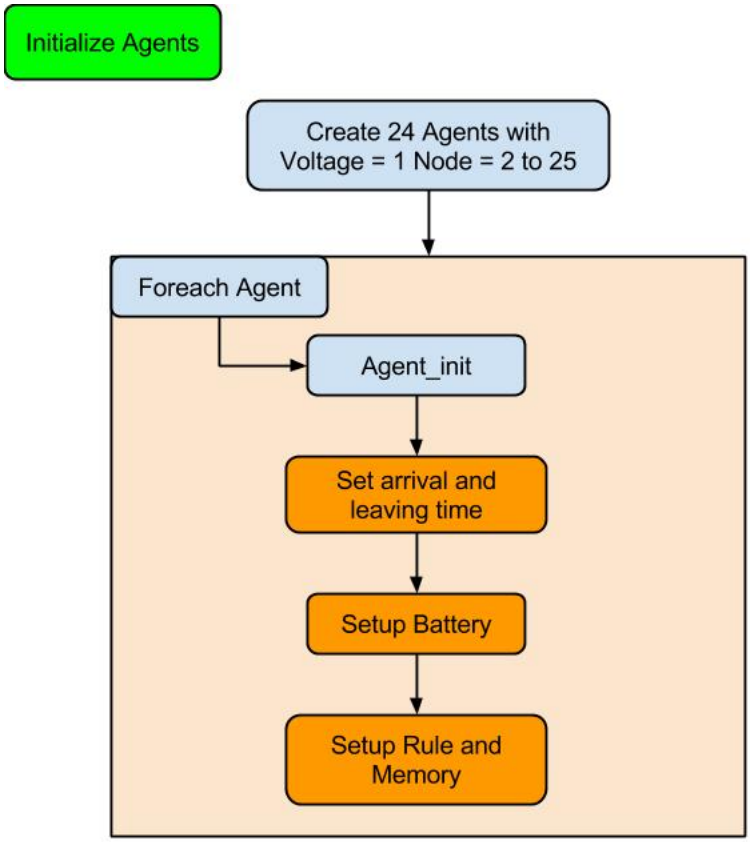

Figure 13: Initialize agents

Pseudo Code:

```
create-turtles agents 24 [set voltage to 1 set node to 2-25]
Foreach Agent
```

```
set count_steps = 0
set count_soc = 0
set preference = 0
Set arrival and leaving time
Setup Battery
Setup Rule and Memory
```

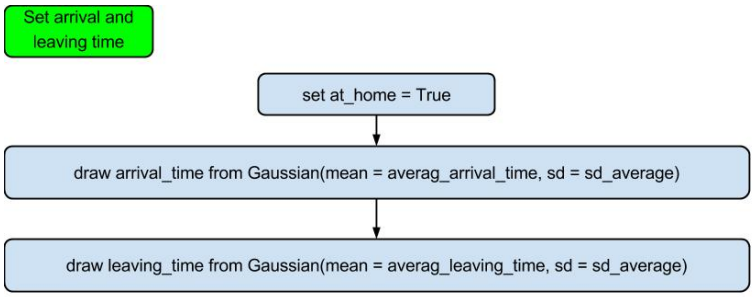

Figure 14: Setup arrival and leaving time

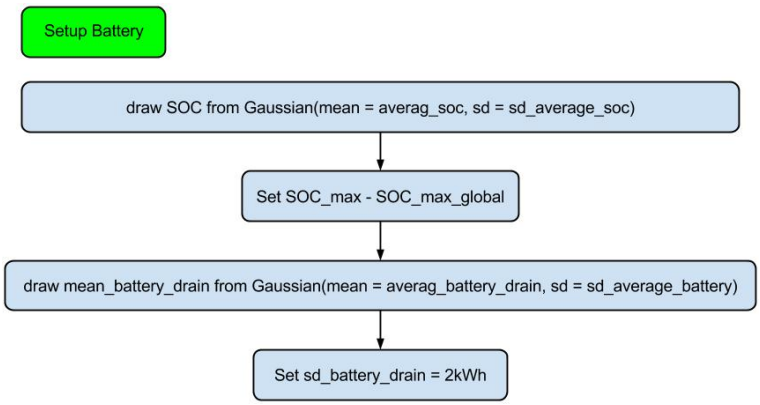

Figure 15: Setup Battery

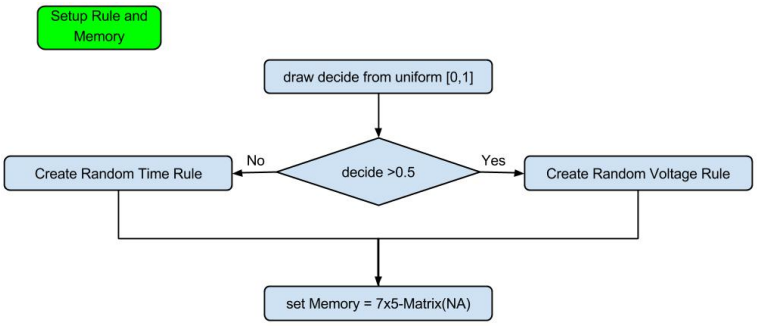

Figure 16: Setup Rules and Memory

Pseudo Code:

```
let decide be a random number between 0 and 1
if decide > 0.5
    let t_begin be randomly picked from t_begin_options
    let t_end be randomly picked from t_end_options
    let soc_threshold be randomly picked from soc_threshold_options
    let action1 be randomly picked from action_options
    let action2 be randomly picked from action_options
    set active_rule to vector[2,t_begin,t_end,soc_threshold,action1,action2]
else
    let voltage_threshold be randomly picked from voltage_threshold_options
    let soc_threshold be randomly picked from soc_threshold_options
    let action1 be randomly picked from action_options
    let action2 be randomly picked from action_options
    set active_rule to vector[1,voltage_threshold,soc_threshold,action1,action2,NA]

let Memory be a 7X5 matrix of NA values
```

To-go

The two things the agents do in the to-go part is to interact with the matpower model and develop rules. The agents interact with the matpower model by in-

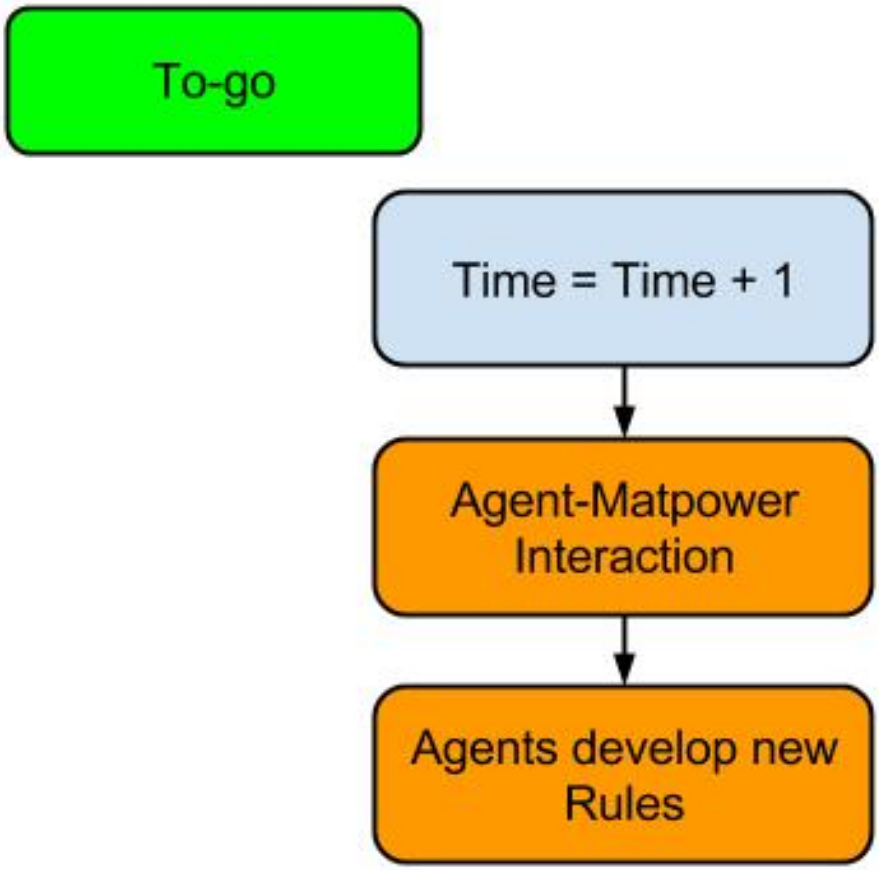

Figure 17: To-go

putting their action into the matpower model, the model then runs and outputs a structured output object. From this output object the agents read the actual power they received. The agents can only interact with the model if they are at home. The agents arrive and leave at specific times, and this state is kept track of by the at_home variable.

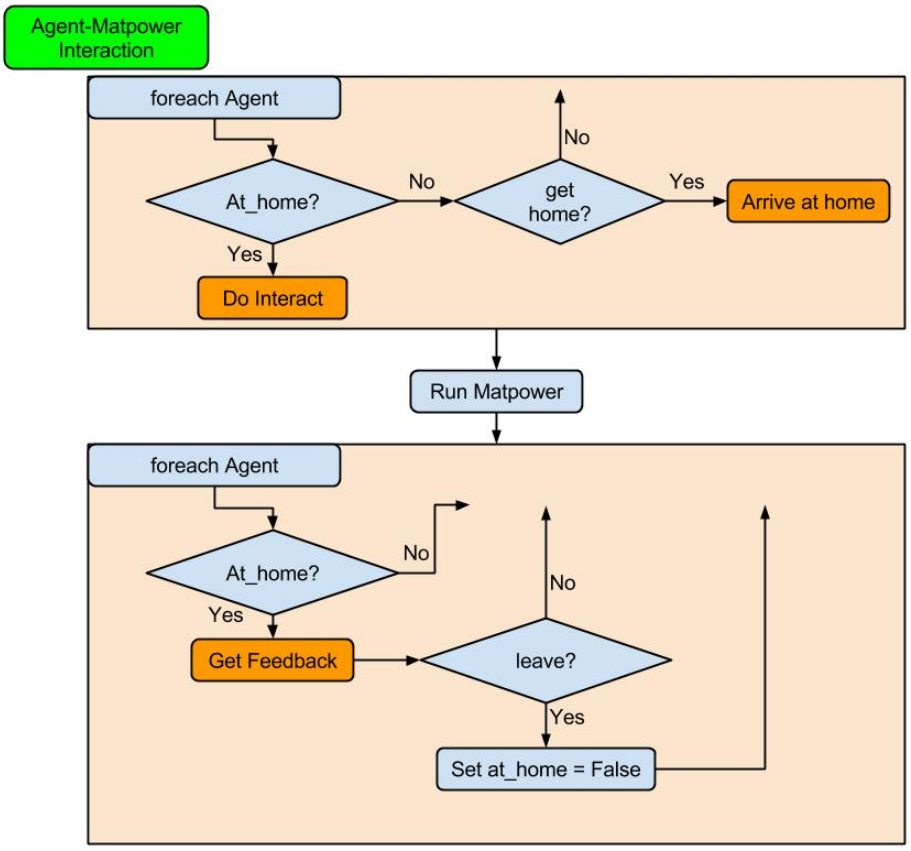

Figure 18: Agent matpower interaction

Pseudo code:

```
foreach agent
    if (at_home ==True)
        Do Interact
    else
        if (time%96 == arrival_time)
            Arrive at home
Run Matpower
foreach agent
    if (at_home == True)
        get feedback
        if (time % 96 == leaving_time)
            set at_home to False
```

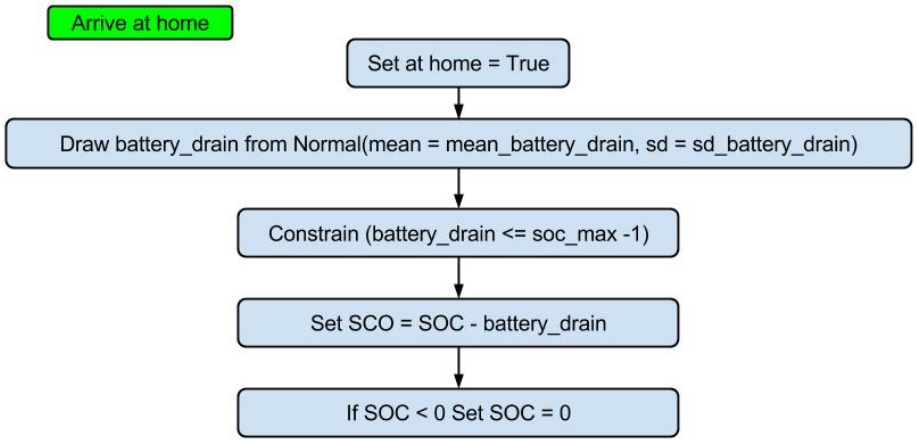

Figure 19: Arrive at home

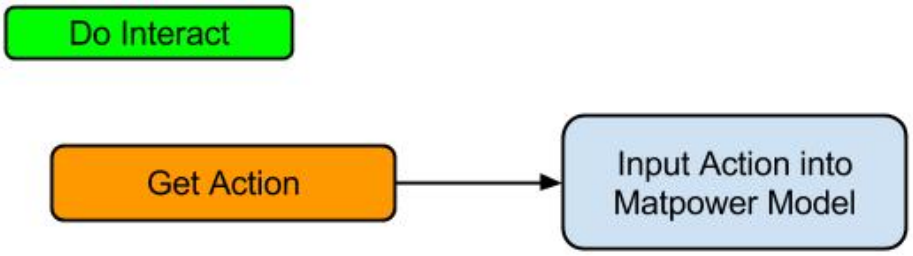

Figure 20: Do Interact

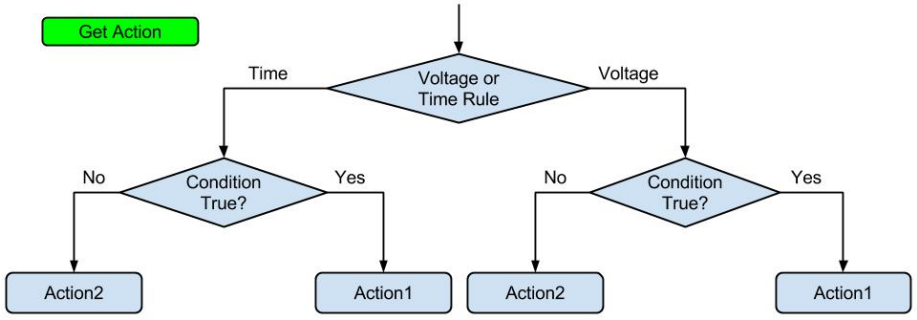

Figure 21: Get Action

Pseudo Code:

```
if (active_rule[index = 1] ==1)
    if (Voltage < active_rule[index = 2]) & (soc > active_rule[index = 1])
        return active_rule[index = 4]
    else
        return active_rule[index = 5]
```

```
else
    if (active_rule[index = 3] > active_rule[index=2])
        if (active_rule[index = 2] < time%96 < active_rule[index = 3]  &  soc > soc_thres.
            return active_rule[index = 5]
        else
            return active_rule[index = 6]
    else
        if  (active_rule[index = 2] < time%96  or time % 96 < active_rule[index = 3]&  soc
            return active_rule[index = 5]
        else
            return active_rule[index = 6]
```

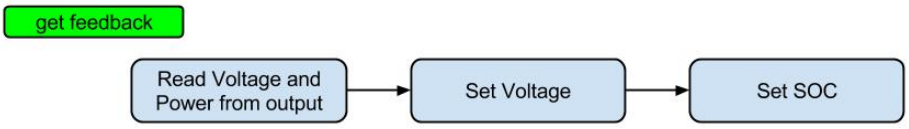

Figure 22: Get feedback

Pseudo Code:

```
let power be  - 1000.0 * output["gen"][node,1] ;; -1000 stems from unit conversions
let volt be output["bus"][node,7]
set voltage = volt
set SOC = SOC + power/4
if SOC > SOC_max
    set SOC to SOC_max
```

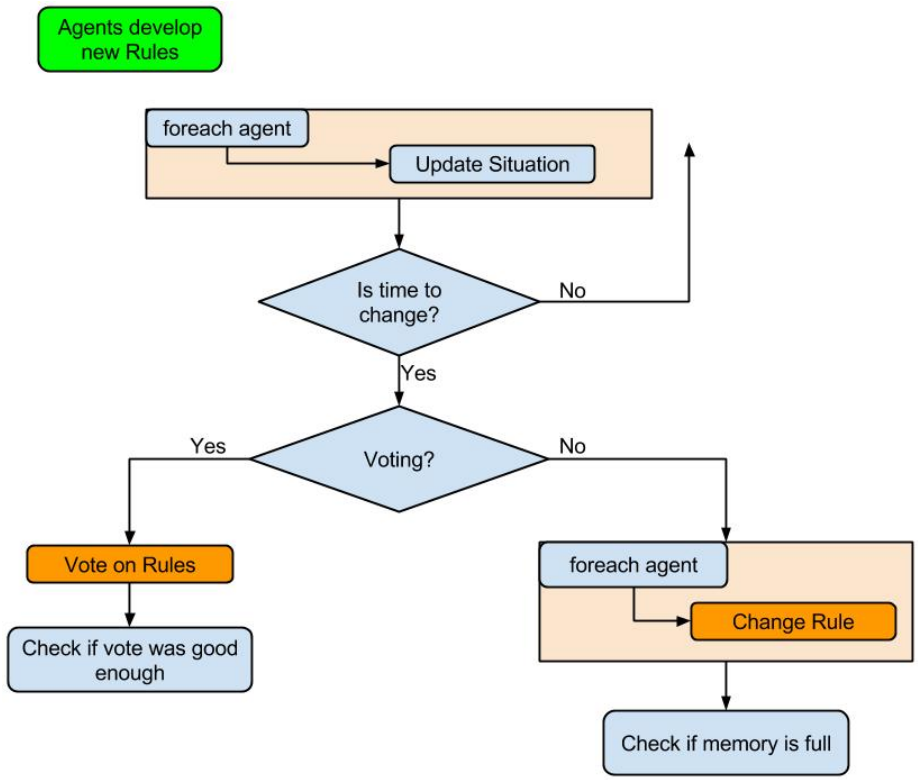

Figure 23: Agents develop new rules

Pseudo Code:

```
foreach agent
    count_steps = count_steps + 1
    count_soc = count_soc + soc
    preference = count_soc / count _steps ;; basically the average soc
voting = False
if (time % 672 == 0)                ;; 672 is one week in 15 minutes ticks
    if (voting == True)
        if (there is already an institutional rule)
            let counter be the number of times the institutional rule is in agent memory
            if (counter < 24 * institutional_success_rate)
                set voting to False
            Vote on rules
else
            Vote on Rules

    else
        foreach agent
            Change Rule
        let in_memory be  True
foreach agent
            if memory is not full
```

```
            set in_memory to False
        if (in_memory == True)
            set voting to True
```

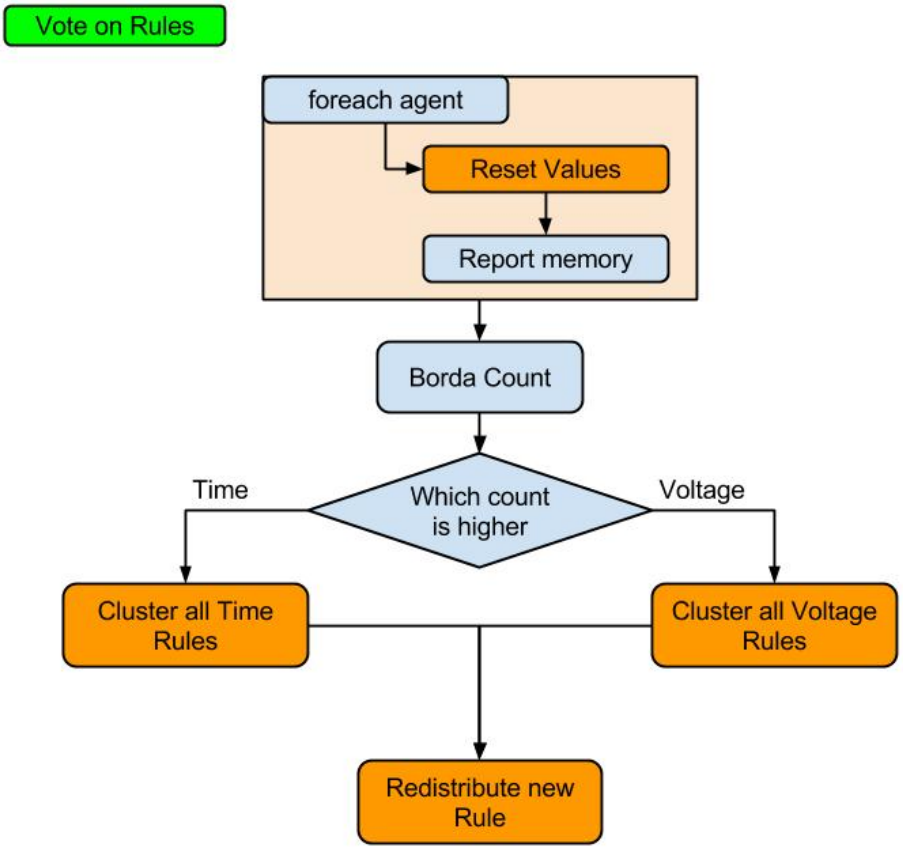

Figure 24: Vote on Rules

Pseudo Code:

```
let election_list be an empty list of matrices
Foreach agent
    reset Values
    add Memory to election_list
let election_matrix be a 240x7 empty  matrix
let n be 5  ;; number of rules for borda count
let i be 1
foreach element in election_list
    for k = 1 to 5
        let multiple be 0
        while multiple < n-k
            set election_matrix[index = all, index = i] to element[k, all]
set i to i + 1
```

```
set multiple to multiple + 1
let number_voltage_rules to number of instances (election_matrix[index = 1, index = all]==
let number_time_rules to number of instances (election_matrix[index = 1, index = all]== 2)
if number_time_rules > number_voltage_rules
    set new_rule to Cluster all Time Rules
else
    set new_rule Cluster all Voltage Rules
foreach agent
    set active_rule to new_rule
```

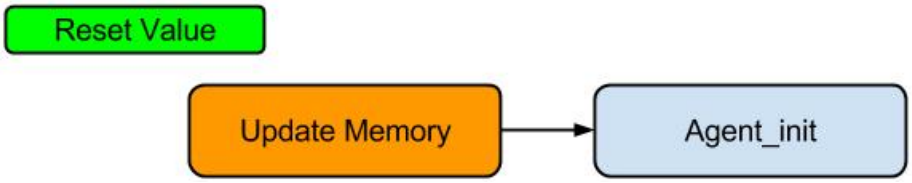

Figure 25: Reset values

Pseudo Code:

```
Update Memory
set count_steps to 0
set count_soc to 0
set preference to 0
```

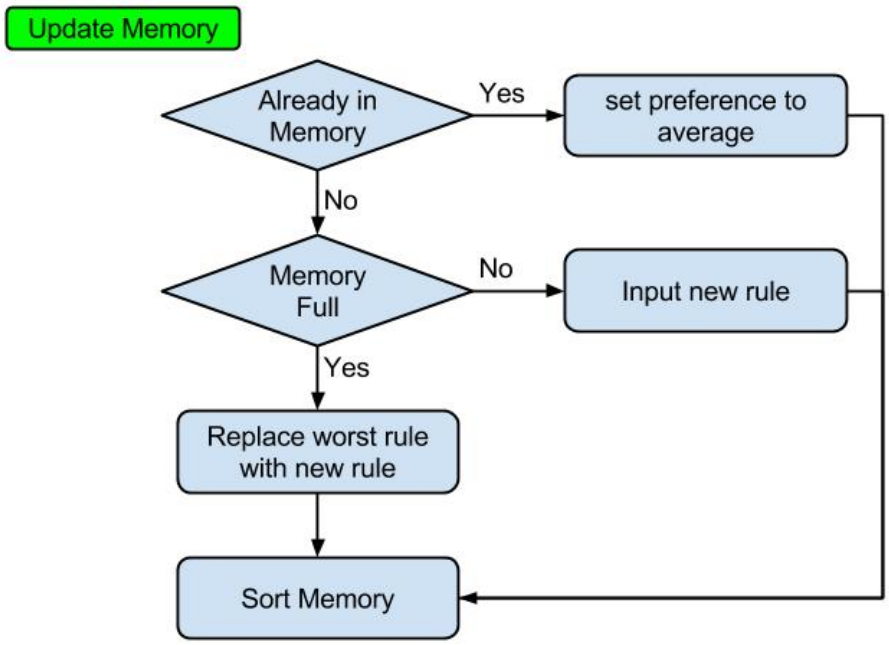

Figure 26: Update Memory

Pseudo Code:

```
 let in_memory be False
for k = 1 to 5
    if memory[k ,index 1 to 6] == active_rule
        set in_memory to True
        let memory_index  be k
if (in_memory == True)
    set memory[memory_index,7] = (memory[memory_index,7] +preference)  / 2
else
    let memory_full be True
    for k = 1 to 5
        if (memory[k ,1] ==NA)
            set memory_full to False
            set memory_index to k
    if (memory_full == True)
        if (active_rule[index = 7] > memory[index = 5m index = ])
            set memory[index = 5, index = 1 to 6] to active_rule
            set memory[index = 5, index = 7] to preference
    else
        set memory[index = memory_index,index = all] to active_rule
        set memory[index = memory_index, index = 7] to preference
sort memory by 7th column
```

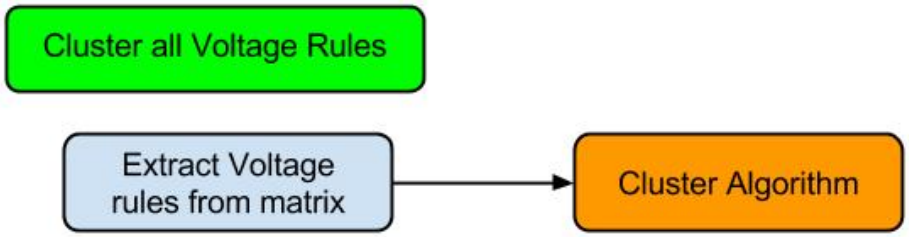

Figure 27: Cluster Voltage Rules

Pseudo Code:

```
let cluster_matrix be a (number_voltage_rules)x7 matrix
let i be 1
for elements in election_matrix
    if (element[index = 1] == 1)
        set cluster_matrix[index = all, index = i] = element
        i = i + 1
Cluster Algorithm(cluster_matrix)
```

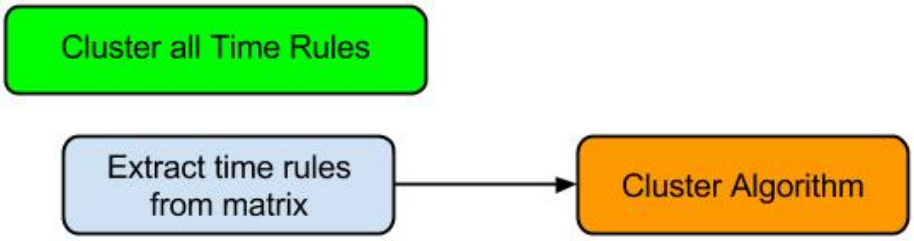

Figure 28: Cluster time Rules

Pseudo Code:

```
let cluster_matrix be a (number_time_rules)x7 matrix
let i be 1
for elements in election_matrix
    if (element[index = 1] == 2)
        set cluster_matrix[index = all, index = i] = element
        i = i + 1
Cluster Algorithm(cluster_matrix)
```

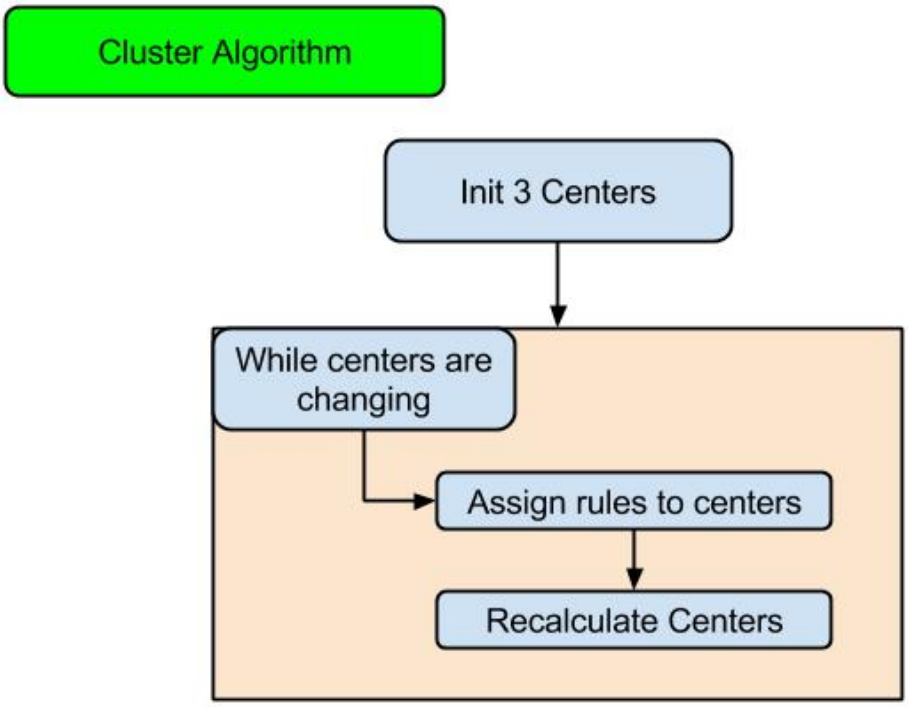

Figure 29: Cluster algorithm

Pseudo Code:

```
let centers be a 3x6 matrtix
if (cluster_matrix[index = 1, index =1] ==1)
```

```
for k = 1 to 3
        set centers[k,all] to random_voltage_rule
else
    for k = 1 to 3
        set centers[k,all] to random_time_rule
let old_centers be a 3x6 matrix (NA)
while old_centers != centers
    foreach element in cluster_matrix
        let lowest_distance be 99
        let lowest_index be 0
        for k = 1 to 3
            if calc_distance(centers[k,all],element[1 to 6] < lowest_distance
                set lowest_distance to calc_distance(centers[k,all],element[1 to 6]
                set lowest_index to k
        set element[7] to lowest_index
    set old_centers = centers
for k = 1 to 3
    set centers[k,all] to arithmetic_mean(cluster_matrix[1 to 6] with (index = 7 ==k)
sort centers by 7th index
return centers[1, all] ;; returns the most voted on mean rule
to report random_voltage_rule
let voltage_threshold be randomly picked from voltage_threshold_options
    let soc_threshold be randomly picked from soc_threshold_options
    let action1 be randomly picked from action_options
    let action2 be randomly picked from action_options
    return vector[1,voltage_threshold,soc_threshold,action1,action2,NA]

to report random_time_rule
let t_begin be randomly picked from t_begin_options
    let t_end be randomly picked from t_end_options
    let soc_threshold be randomly picked from soc_threshold_options
    let action1 be randomly picked from action_options
    let action2 be randomly picked from action_options
    return vector[2,t_begin,t_end,soc_threshold,action1,action2]
```

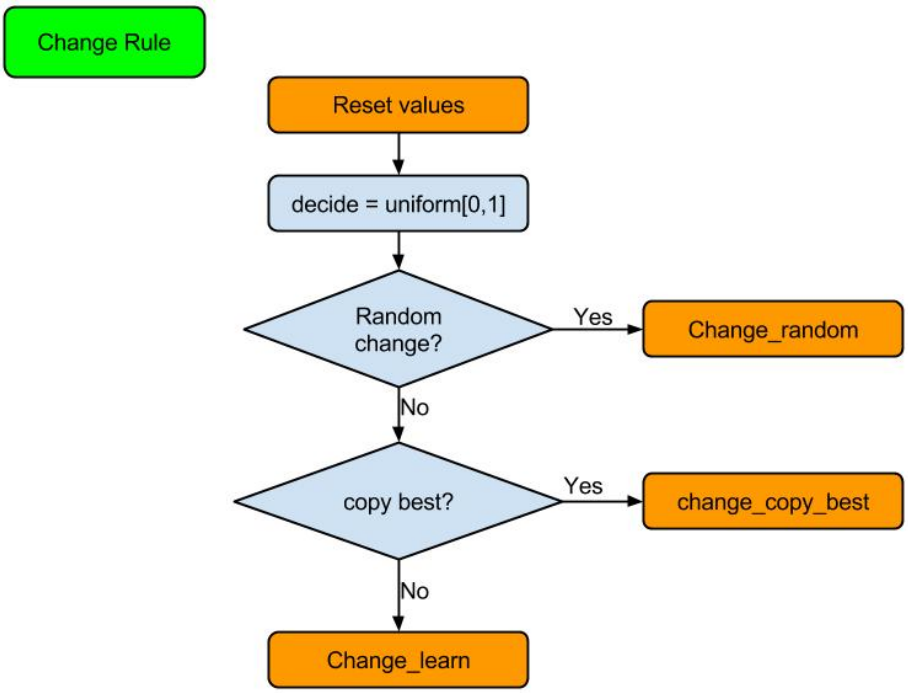

Figure 30: Change Rule

Pseudo Code:

```
Reset values ;; see above
let decide be a random number between 0 and 1
if (decide > 1 - random_change)
    Change_random
else
    change_copy_best
    else
        Change_learn
```

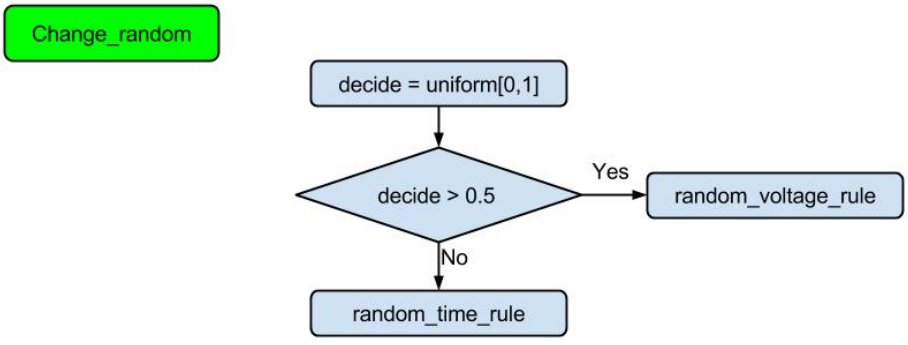

Figure 31: Change random

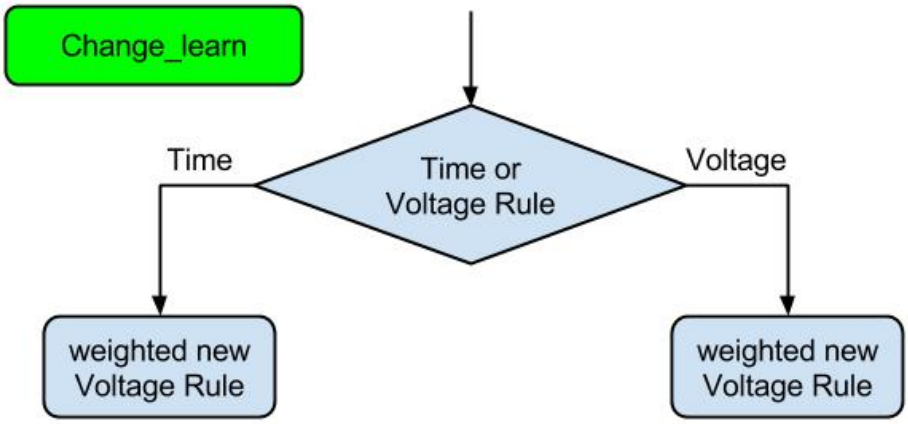

Figure 32: Change learn

Pseudo Code:

```
let index = 0
let number_voltage be 0
let voltage_preference be 0
let number_time be 0
let time_preference be 0
for k = 1 to 5
    if (memory[k,1] == 1)
        set number_voltage = number_voltage + 1
        set preference_voltage = preference_voltage + memory[k,7]
    if (memory[k,1] == 2)
        set number_time = number_time + 1
        set preference_time = preference_time + memory[k,7]
    if (memory[k,1]==NA)
        set index to k - 1
if (index ==1)
    set active_rule to matrix[1, all]
else
    if (number_voltage==0 or number_time==0)
        if (number_voltage > number_time)
            set active_rule to weighted_new_voltage_rule
        else
            set active_rule to weighted_new_time_rule
    else
        if (preference_voltage/number_voltage > preference_time/number_time)
            set active_rule to weighted_new_voltage_rule
        else
            set active_rule to weighted_new_time_rule

to report weighted_new_voltage_rule
let averaged_rule be vector[0,0,0,0,0,0]
let count be 0
```

```
for k = 1 to 5
    if (memory[k,1] == 1)
        averaged_rule = averaged_rule + memory[k, 1 to 6]
        count = count + 1
return (averaged_rule  / count)

to report weighted_new_time_rule
let averaged_rule be vector[0,0,0,0,0,0]
let count be 0
for k = 1 to 5
    if (memory[k,1] == 2)
        averaged_rule = averaged_rule + memory[k, 1 to 6]
        count = count + 1
return (averaged_rule  / count)
```

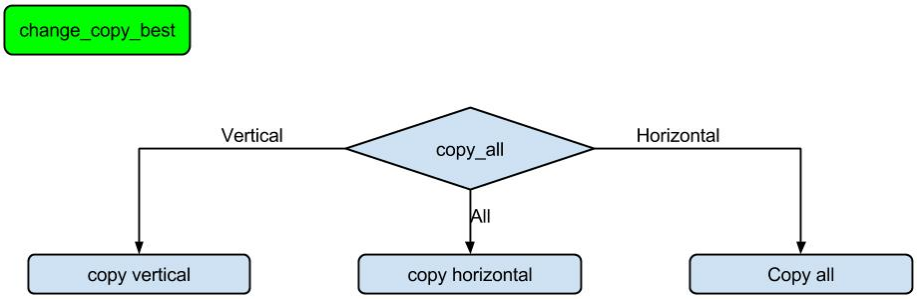

Figure 33: Change copy best

Pseudo Code:

```
If copy_all = \all"
    let best_value be  -1
            let best_rule be empty rule
            for i in agents:
                if (node of i-th agent != node of original agent):
                        if preference of i-th best rule> best_value:
                            best_rule = best rule of i-th agent
                            best_value = preference of i-th agents best rule
            set active_rule  to  best_rule

elsif (copy_all == "vertical")
            let best_value be -1
            let best_rule be empty rule
            let branch be branch in which agent is in
            for i in agents:
                let branch2 be branch in which  i-th agent is in
                if ((branch == branch2) &(node of i-th agent != node of original agent)):
                        if preference of i-th best rule > best_value:
                                best_rule = best rule of i-th agent
                                best_value = preference of i-th agents best rule
```

```
        self.active_rule = best_rule
else:
        let best_value be -1
        let best_rule be empty rule
        level = level of agent
        for i in agents:
            level2 = level on which i-th agent is on
            if ((level == level2) & (node of i-th agent != node of original agent)):
                    if preference of i-th best rule > best_value:
                            best_rule = best rule of i-th agent
                            best_value = preference of i-th agents best rule
         set active_rule to  best_rule
```

References

[1] Ostrom, E., Gardner, R., & Walker, J. (1994). Rules, games, and common-pool resources. University of Michigan Press.

[2] Lund, H., Andersen, A. N., Østergaard, P. A., Mathiesen, B. V., & Connolly, D. (2012). From electricity smart grids to smart energy systems–a market operation based approach and understanding. Energy, 42(1), 96-102.

[3] Wolsink, M. (2012). The research agenda on social acceptance of distributed generation in smart grids: Renewable as common pool resources. Renewable and Sustainable Energy Reviews, 16(1), 822-835.

[4] Stragier, J., Hauttekeete, L., & De Marez, L. (2010, September). Introducing Smart grids in residential contexts: Consumers' perception of Smart household appliances. In Innovative Technologies for an Efficient and Reliable Electricity Supply (CITRES), 2010 IEEE Conference on (pp. 135-142). IEEE.

[5] Geelen, D., Reinders, A., & Keyson, D. (2013). Empowering the end-user in smart grids: Recommendations for the design of products and services. Energy Policy, 61, 151-161.

[6] Park, C. K., Kim, H. J., & Kim, Y. S. (2014). A study of factors enhancing smart grid consumer engagement. Energy Policy, 72, 211-218.

[7] Kostyk, T., & Herkert, J. (2012). Societal implications of the emerging smart grid. Communications of the ACM, 55(11), 34-36.

[8] Fhom, H. S., & Bayarou, K. M. (2011, November). Towards a holistic privacy engineering approach for smart grid systems. In Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on (pp. 234-241). IEEE.

[9] Sandel, M. J. (2000). What money can't buy: the moral limits of markets. Tanner Lectures on Human Values, 21, 87-122.

[10] Pedrasa, M. A. A., Spooner, T. D., & MacGill, I. F. (2010). Coordinated scheduling of residential distributed energy resources to optimize smart home energy services. Smart Grid, IEEE Transactions on, 1(2), 134-143.

[11] Fattori, F., Anglani, N., & Muliere, G. (2014). Combining photovoltaic energy with electric vehicles, smart charging and vehicle-to-grid. Solar Energy, 110, 438-451.

[12] Deilami, S., Masoum, A. S., Moses, P. S., & Masoum, M. A. (2011). Real-time coordination of plug-in electric vehicle charging in smart grids to minimize power losses and improve voltage profile. Smart Grid, IEEE Transactions on, 2(3), 456-467.

[13] Shaaban, M. F., Eajal, A. A., & El-Saadany, E. F. (2014). Coordinated charging of plug-in hybrid electric vehicles in smart hybrid AC/DC distribution systems. Renewable Energy.

[14] Galus, M. D., & Andersson, G. (2008, November). Demand management of grid connected plug-in hybrid electric vehicles (PHEV). In Energy 2030 Conference, 2008. ENERGY 2008. IEEE (pp. 1-8). IEEE.

[15] Valogianni, K., Ketter, W., & Collins, J. (2013, July). Smart Charging of Electric Vehicles using Reinforcement Learning. In AAAI Workshop: Trading Agent Design and Analysis.

[16] Lauer, M., & Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In In Proceedings of the Seventeenth International Conference on Machine Learning.

[17] Turner, R. M. (1993). The tragedy of the commons and distributed AI systems. Department of Computer Science, University of New Hampshire.

[18] Tajima, K. (2007). The theory of institutions and collective action in Adam Smith's Theory of Moral Sentiments. The Journal of Socio-Economics, 36(4), 578-594.

[19] Alpaydin, E. (2014). Introduction to machine learning. MIT press.

[20] Vlassis, N. (2007). A concise introduction to multiagent systems and distributed artificial intelligence. Synthesis Lectures on Artificial Intelligence and Machine Learning, 1(1), 1-71.

[21] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas, Matpower: Steady-State Operations, Planning and Analysis Tools for Power Systems Research and Education," Power Systems, IEEE Transactions on, vol. 26, no. 1, pp. 12-19, Feb. 2011

[22] `http://www.powerworld.com/files/S01SystemModeling.pdf`

[23] Wooldridge, M. (2009). An introduction to multiagent systems. John Wiley & Sons.

[24] `http://ffcpg.com.my/portal/Products.nsf/2d248e16d8d12df4825719b00301d50/5801a1bfde86e113482577e40009d126/$FILE/LV+TD.pdf`

[25] Willis, H. L. (1997). Power Distribution Planning Reference Book. CRC Press.