# Flexibility Informed Profile Steering using Shapley Values

Felix van der Waals, s2835371
*Department of EEMCS, University of Twente, Enschede, The Netherlands*
Email: f.w.j.vanderwaals@student.utwente.nl

***Abstract*—Profile Steering (PS) is a Demand Side Management (DSM) method that can schedule devices in order to reduce peak demands on the power grid. In PS, a coordinator iteratively asks all devices (or nodes) to optimize their power profile and accepts the best candidate profile presented. PS is unscalable as all nodes are asked to compute a candidate profile in each iteration.**

**We propose Flexibility Informed Profile Steering (FIPS). An algorithm that uses flexibility information to determine the order in which nodes are asked to optimize on the problem. To quantify flexibility we use previous research on a solution concept called 'Shapley values' from game theory. This concept considers all contributions of a node with different combinations with other nodes.**

**In this paper we investigate the performance of FIPS compared to PS by simulating power demand of different sizes of neighborhoods for a year. We found that FIPS has better scalability and an improved computational efficiency. Furthermore, the way we use the flexibility information influences the scalability, computational efficiency and the reached objective score.**

***Index Terms*—Profile Steering, Shapley Value, Demand Side Management**

## I. Introduction

The amount of electrical power demanding devices in households is increasing because of the demand for lower carbon emissions [1]. This increase in electric vehicles (EV), battery energy storage systems (BESS) and photovoltaics (PV) results in larger peak demands of the power grid. The power grid itself can only handle a certain amount of power at a time and therefore it imposes a maximum permissible power peak. As increasing the capacity of the power grid, by upgrading or replacing assets, is a rather inefficient way of reducing the pressure on the grid, different approaches of preventing power peaks have been explored in literature. One of which is Demand Side Management (DSM); by managing the demand, the needed capacity can be reduced.

Profile Steering (PS) is such a DSM method and effectively schedules the power demand of devices in order to 'shave' power peaks by asking all devices to optimize their own power profile after which the best optimization is chosen [2]. However, because the algorithm has a quadratic computational complexity and its computational efficiency is low it is unscalable and it potentially uses more computations than needed [3].

In this paper we propose an improved version of PS where a quantification of the flexibility of devices is used to determine the order in which the devices are asked to optimize their own power profile. To quantify the flexibility we use the Shapley value solution concept as was studied by Varenhorst et al. [4]. The concept is used to assign a value to a device based on the average of its marginal contributions in the past. A contribution is quantified by the amount of improvement a device could introduce.

In Section II we first explain the precise workings and limitations of PS and some altered versions of PS. Then we elaborate in Section III on our proposed algorithm: Flexibility Informed Profile Steering (FIPS). Subsequently, in Section IV, we analyze FIPS in comparison to PS where we focus on the peak-shaving performance, empirical computational complexity and computational efficiency. Lastly, we discuss our findings and draw conclusions from them in Sections V and VI.

## II. Background and related works

In this section, the precise workings and the limits of PS are explained. Then, some different versions are discussed that optimize PS in some way after which a method to quantify flexibility is discussed.

### A. Profile Steering

PS is a DSM method to use the grid more efficiently proposed by Gerards et al. [2]. The algorithm uses devices their flexibility in planning to steer the power profile $\vec{x}$ to a desired profile $\vec{p}$. It does this by first asking each node $m \in \{1, ..., M\}$, where a node is a device or a household (PS can work at any level in a coordinator follower hierarchy), to change its own power usage $\vec{x}_m$ to be as close as possible to the zero profile i.e. it minimizes $||\vec{x}_m||_2$ (Euclidean distance, 2-norm). These profile vectors are representations of power usage at a specific time. All these profiles are added to create the total profile: $\vec{x} = \sum_{m=1}^{M} \vec{x}_m$.

After the total power profile is created, every node is asked to create a candidate profile $\vec{\hat{x}}_m$ that minimizes the Euclidean 2-norm distance between the total power profile $\vec{x}$ and the desired profile $\vec{p}$. An improvement $e_m$ of this candidate profile $\vec{\hat{x}}_m$ compared to the previous profile $\vec{x}_m$ is calculated and the node that presents the candidate profile that can improve the profile the most, can change its profile. This chosen node calculates the difference between its current profile $\vec{x}_m$ and the candidate profile $\vec{\hat{x}}_m$. The change in profile is added to the total profile $\vec{x}$. Then all nodes are asked again to create a new candidate profile based on the new total profile $\vec{x}$ and

this is repeated until no significant changes to the total profile are made, meaning $max(e_m)$ is below a certain threshold $\epsilon$.

PS is able to reduce power peaks, but for every iteration that is made, all nodes need to receive and transmit a message with data. This means that if one node responds slow it becomes a bottleneck for the algorithm. Furthermore, increasing the amount of nodes makes the performed computations grow quadratically [3].

### B. Versions of Profile Steering

Several methods to improve certain characteristics of profile steering exist. Two relevant papers are discussed in this subsection.

In the first paper, written by Pappu et al. [5], the authors proposed an asynchronous and distributed method based on profile steering. This method removes the coordinator role from the network effectively having a distributed network instead of a centralised or decentralised network. The algorithm runs on every node in the system and all nodes optimize their own profile periodically. When a node is optimizing its own profile it acquires the global problem and notifies the other users that it is optimizing such that no other nodes optimize on the same problem. This method has shown that it can be 11 times faster in terms of performed computations while having a 1.35% deviation in its final objective score compared to PS. The increase in the final objective score indicates that the order in which nodes can optimize on the global problem has some effect on the local minimum found for the objective score. In addition to the improvement in the amount of computations, with this method the slowest node in the system is no longer a bottleneck for the algorithm.

In the second paper Hoogsteen et al. [3] did research on the scalability of profile steering. By combining improvements of PS and parallelization techniques the authors were able to bring the complexity of the algorithm from quadratic to linear while obtaining similar results. The optimization problem was distributed among clusters making parallelization possible. PS was improved by accepting multiple profiles each iteration and pruning nodes that had small improvements in the previous iteration. These changes improve the computational efficiency of PS because for each iteration less improvements are calculated and more candidate profiles are accepted, resulting in less redundant calculations. As the computational efficiency is increased with these changes a considerable amount of calculations is still redundant because not all calculated candidate profiles are accepted.

### C. Quantifying flexibility

The flexibility of a node is a metric of considerable importance for the PS algorithm. In PS, the possible improvement of a node is calculated during each iteration, resulting in a precise improvement $e_m$ specific to that iteration. The found improvement $e_m$ values are then used to determine the node that can replace its profile $\vec{x}_m$ with its candidate profile $\vec{\hat{x}}_m$. Optimizing the profile of a node each iteration is energy-consuming and therefore, having an estimation of this improvement for each node is of use for an improved version of PS. As the improvement of a node is partly determined by its flexibility some quantification method for the flexibility of a node can serve as a proxy estimate of the actual possible improvement of a node.

Varenhorst et al. [4] propose a way of quantifying device flexibility by making use of the Shapley value solution concept. Calculating a Shapley value is done by taking the average of all marginal contributions of all the possible combinations of devices. The marginal contribution of the node is calculated by taking the difference between the Root Mean Square (RMS) of the power profile without steering and the RMS of the power profile with (profile) steering. The average of these marginal contributions is taken since the flexibility of a device is partly determined by its synergy with other devices. The computations needed for calculating the Shapley value for a node grows exponentially with the amount of nodes making it unscalable. However, an approximation of the Shapley value can still be made by using the results obtained by Varenhorst et al.

## III. Algorithm

In Section II-B the improvements and the deficiencies of the two versions of PS were discussed. In this section we propose an improved version of PS that tries to combine the lessons learned from literature: Flexibility Informed Profile Steering (FIPS). The steps of this version are described in Algorithm 1.

The initialization of this new method remains the same as for PS; the coordinator asks every node to optimize its own profile $\vec{x}_m$ and combines these profile to create a total profile $\vec{x}$. After the initialization phase is done, only a single node is asked to optimize on the problem and its candidate profile is accepted without any consideration of the possible optimal solutions of the other nodes. As a result, no redundant profiles are optimized and thrown away and a different local minimum can be found. To pick the node that is allowed to create a new optimal planning, we create an ordered list of nodes ($Msort[]$) after the initialization phase. The order of the nodes on the created list determines the order in which nodes can optimize their own profile $\vec{x}_m$ to change the total profile $\vec{x}$. When we reach the end of the list, we start at the beginning of the list again until the stopping criterion is met.

### A. Ranking methods

The way we sort the list $Msort$ influences the amount of iterations needed to converge and the final objective score reached as we saw in the paper written by Pappu et al. [5]. Therefore, in this paper we consider three methods that sort the nodes on the list in a different way.

*1) Flexible nodes last:* The first method uses the previously discussed Shapley values. The nodes are ranked from least flexible to most flexible where flexibility is estimated by a Shapley value as explained by Varenhorst et al. [4]. We expect that when the least flexible nodes are asked to optimize first, a better objective score is achieved compared to other methods. This would make sense because a node that offers low flexibility is more likely to lose its flexibility when a node

**Algorithm 1** Pseudocode of FIPS

**for** m $\in$ {1, ..., M} **do**
  Request node $m$ to minimize $||\vec{x}_m||_2$
$\vec{x} := \sum_{m=1}^{M} \vec{x}_m$
Create $Msort[]$ and sort nodes based on one of the methods in Section III-A
$i := 0$
**repeat**
  $\vec{d} := \vec{x} - \vec{p}$
  $m := sorted\_nodes[i \mod |Msort|]$
  $\vec{p}_m := \vec{x}_m - \vec{d}$
  For node $m$, find a planning $\vec{\hat{x}}_m$ that minimizes $||\vec{\hat{x}}_m - \vec{p}_m||_2$
  $e_m := ||\vec{x}_m - \vec{p}_m||_2 - ||\vec{\hat{x}}_m - \vec{p}_m||_2$
  $\vec{x} := \vec{x} - \vec{x}_m + \vec{\hat{x}}_m$ {update total profile}
  $\vec{x}_m := \vec{\hat{x}}_m$ {update profile of node m}
  $i := i + 1$
**until** $\max(e_m) < \epsilon \wedge i >= |Msort|$

that offers high flexibility is asked first. We refer to this method as FIPS flex. last.

*2) Flexible nodes first:* The second method also uses flexibility Shapley values. For this method the nodes are ranked from most flexible to least flexible. We expect that this sorting method performs similar to PS in terms of final objective score because, like PS, it is a greedy strategy as it chooses the nodes that can optimize the most over the other nodes. We refer to this method as FIPS flex. first.

*3) Random:* The third method randomly sorts the nodes each time the list is created. This method gives an indication of the importance of the order of the nodes because when the performance is similar to the two previously described sorting methods it indicates that sorting based on the flexibility does not influence performance and vice versa. We refer to this method as Uninformed Profile Steering (UPS) because it does not use any flexibility information.

### *B. Stopping criterion*

The stopping criterion cannot remain the same because the way in which the nodes are chosen is different than PS; when a node is able to improve on the problem below the threshold value, it does not mean that the other nodes cannot improve on the problem anymore. To solve this we store the last improvements of each node and compare the maximum of the stored improvements to a threshold value. When the maximum improvement is smaller than the threshold value it indicates that it is likely that the point of convergence is reached or that it is close (if the threshold value is sufficiently small). To make sure that the algorithm does not stop prematurely all nodes need to have calculated an improved at least once.

## IV. Evaluation

To evaluate the performance of the proposed algorithm we perform a simulation where all algorithms receive the same use-case.

TABLE I
Flexibility Shapley values of devices in the simulation [4]

| | |
|---|---|
| EV (11 kW, 50 kWh) | 719 |
| EV (3.7 kW, 10 kWh) | 615 |
| BESS (3 kW, 13.5 kWh) | 560 |
| HP (2 kW, 3 kWh) | 453 |
| BESS (3 kW, 7 kWh) | 413 |
| BESS (3 kW, 3.5 kWh) | 219 |
| DW | 26 |
| WM | 5 |

To simulate a real life power demand we use the Artificial Load Profile Generator (ALPG), by Hoogsteen et al. [6]. The algorithm simulates demands of households which consist of EVs, BESSs, dishwashers (DWs), washing machines (WMs), heat pumps (HPs) and a static load.

The power usage of EVs, DWs and WMs can be planned as needed between two given time intervals. BESSs can charge themselves and supply power to the grid to optimize the given problem. HPs are similar to BESSs as they can store some energy. However, they also have a heat demand where they need to generate a specific amount of heat at given time intervals. The HPs can decide when to charge to meet the demand. In our simulation we use heat demand profiles from 2020 of different houses.

Our power demand simulation has a length of 366 days (one year) where every day is composed of 96 intervals. For every day in the year a 2-day-ahead planning is made by the profile steering algorithms. We shift the 2-day-ahead planning with steps of 1 day resulting in overlapping days. All devices for which the two given time intervals lie within the current two day window can be scheduled. Once a device is scheduled and the window shifts by a day there are two possibilities. If the two given time intervals also lie within the new two day window the device can be rescheduled. If at least one of the two given time intervals does not lie within the new two day window the profile of the device is added to the simulation but can not be rescheduled.

The power usage of all the combined households is measured, meaning that a BESS can supply power to another household without contributing to the power usage. Another way to look at this is that we measure the power usage from the point of view of the supplier. This is not realistic because the exchange of power between houses does put some pressure on the power grid. However, in this evaluation we focus at the differences between the different algorithms and not the performance of the algorithms their implementation in the real world.

We use the flexibility Shapley values obtained by Varenhorst et al. [4]. In Table I the devices present in the simulation accompanied by their Shapley values are found. The values found in the table are the values that we use to sort the list of devices. As the static load is not plannable, it is not included in the table.

We perform four power demand simulations of a year, where each simulation has a neighbourhood with a different amount of households with a step size of 7 households. The composition of households remains the same meaning that

TABLE II
AVERAGE OBJECTIVE SCORES PER DAY FOR ALL ALGORITHMS

| Households | 7 | 14 | 21 | 28 |
|---|---|---|---|---|
| PS | 30639 | 57907 | 88103 | 123115 |
| FIPS flex. last | 30634 | 57896 | 88087 | 123093 |
| FIPS flex. first | 30640 | 57907 | 88103 | 123116 |
| UPS | 30638 | 57902 | 88095 | 123103 |

the offered flexibility also remains the same. The aim of all algorithms is to minimize $||\vec{x}||_2$. The stopping threshold value for all algorithms is $\epsilon = 1e{-}4$. Having gathered data from the simulations, we look at the difference between the algorithms in peak-shaving performance, empirical computational complexity and computational efficiency.

### A. Peak-shaving performance

To compare the peak-shaving performance of the algorithms the average objective score per day is calculated as follows: $\frac{1}{366}\sum_{g=1}^{366}||\vec{x}_g||_2$ where $\vec{x}_g$ is the profile of a single day $g$ of the year. The lower the score is, the better the algorithm performed. The final average objective scores per day for all performed simulations can be found in Table II.

From this table it can be seen that all sorting methods slightly outperform PS for all neighbourhood sizes. The average objective score relative to PS of FIPS flexible first, UPS and FIPS flexible last is $+0.000\%$, $-0.008\%$ and $-0.018\%$ respectively. This shows that all proposed algorithms perform similar or slightly better compared to PS when it comes to peak-shaving performance.

The slight improvement in the objective score of UPS and FIPS flexible last is explained by the order the nodes are chosen. When we plan the more flexible devices first, the less flexible devices have even less flexibility than before resulting in a slightly worse objective score.

Based on these results we conclude that the order in which we ask nodes to optimize has a minimal influence on the finding of a different local minimum. Asking flexible nodes last to optimize can result in the finding of a slightly better local minimum. However, this difference in the reached objective score is minimal.

### B. Empirical computational complexity

When we talk about empirical computational complexity of the algorithm we mean how much the amount of resources needed to run the algorithm increases as the complexity of the problem itself increases. To compare the empirical computational complexity of the algorithms we look at the average number of nodes asked to optimize because this number grows in the same fashion as the total resources needed to run the algorithm. The complexity of the problem is increased by increasing the number of households.

Retrieving the average and the standard deviation of the number of plannings made from the performed simulations results in Fig. 1. It can be seen that for all sorting methods the empirical computational complexity is significantly reduced compared to PS. Between the sorting methods there is a difference in empirical computational complexity. Asking nodes randomly to optimize results in a similar empirical computational complexity as asking the most flexible nodes first. Furthermore, asking the flexible nodes last to optimize results in a slightly higher empirical computational complexity compared to the other sorting methods.

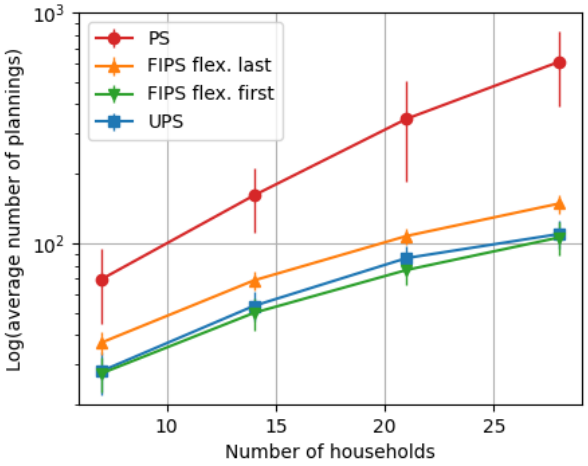


Fig. 1. Average plannings as a function of the number of households

These results show that the order in which nodes are asked to optimize matters for the empirical computational complexity of the algorithm.

### C. Computational efficiency

When we talk about computational efficiency in this paper we mean how much resources are used to achieve a certain objective score. Because the amount of resources, as previously described, grows in a similar fashion as the amount of nodes asked to optimize we consider this as our metric.

In Fig. 1 we already see that all sorting methods have better computational efficiency than PS. To further analyze the computational efficiency we look at how the reached objective score changes with respect to node plannings.

We visualize this by plotting the average objective score after a certain number of plannings $i$: $\frac{1}{366}\sum_{g=1}^{366}||\vec{x}_{g,i}||_2$ where $\vec{x}_{g,i}$ is the profile of the 2-day window visible on day $g$ after $i$ plannings. Because the stopping criterion is based on a threshold, the total amount of plannings differs for each day. This means some $\vec{x}_{g,i}$ elements do not exist for larger $i$. To still be able to take the average we extend all vectors $\vec{x}_{g,i}$ for a specific day $g$ by repeating the last objective score reached until all vectors have the same length of the maximum number of plannings made across all days. PS asks multiple nodes to plan each iteration and the number of devices present across all days $g$ is not the same. Therefore the rounded average number of present devices across all days is taken. This average is used as a step size for the number of plannings $i$ for PS.

We add 1 to the number of plannings $i$ to make a logarithmic scale possible. The resulting graph of the simulation with

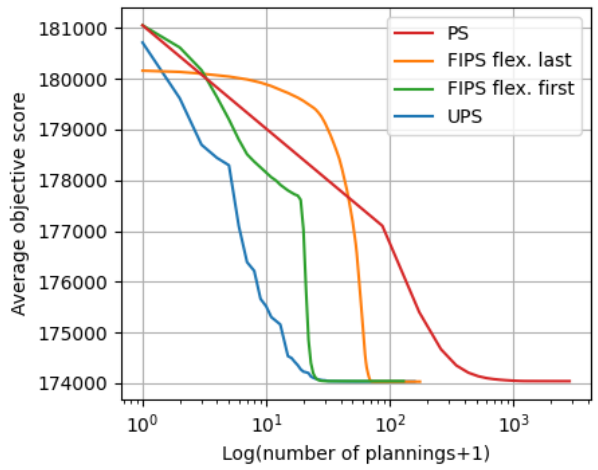


Fig. 2. Objective score as a function of devices asked

28 households is found in Fig. 2. All other simulations with different amount of households showed similar behaviour.

Fig. 2 shows that all sorting methods ask less nodes to optimize to reach a certain score compared to PS. Asking the flexible nodes last needs more computations to overtake PS. This is an inherent characteristic of asking the flexible nodes last because it is not a greedy algorithm like PS and FIPS flex. first that aim to converge as fast as possible.

Another observation we made is that asking the most flexible devices first does not result in the steepest descent. The Shapley values that we use are not taking into consideration that multiple devices of the same kind can have the same coordinator. In effect, the same kind of devices are asked to optimize on the problem in the first few iterations. In our case these first devices are the EVs. Because the EVs must be planned in a similar window the improvement of the device decreases significantly after having asked other EVs. UPS asks nodes randomly to optimize so it does not have this problem. As a result, UPS has the highest computational efficiency on average compared to the other algorithms.

A remarkable difference between the algorithms is that the average objective starting score differs. This is possible because a 2-day-ahead planning is made and this window of two days shifts with steps of one day. The planning of the previous day is saved. Therefore, the algorithm can plan devices on the second day such that the objective starting score is lower.

In short, the results show that the all methods have greater computational efficiency than PS and the order the devices are asked to optimize influences the slope of the descent to a local optimum. Asking devices randomly results in the best computational efficiency on average.

## V. Discussion

In this section we discuss the reliability of the found results, possible improvement and the implications of the results regarding computation time.

Firstly, about the reliability of the results. It needs to be mentioned that changing the threshold value can have a big impact on the results. A small change in planning of a single node might cause other nodes to improve their profile even more resulting in a series of similar events. In Section IV we used $1e{-}4$ as the threshold value. Although this value is rather small, it might still have been possible that such events were suppressed. In addition to this, the stopping criterion is not the same for PS and FIPS causing the possibility for these events to be different for both methods. Furthermore, the given use-case also has an effect on the reliability of our results. In this paper we used the ALPG algorithm to simulate demand of a neighbourhood. However, there might be scenarios where PS, a greedy algorithm, outperforms FIPS and the other way around. The aforementioned matters can have some impact on the results.

In Section IV-C we found that asking flexible nodes first to optimize results in a worse computational efficiency compared to UPS. This is due to the same type of devices being asked to optimize in the first iterations. Calculating the actual Shapley value for all nodes could result in a better computational efficiency.

From Section IV-B and IV-C it is clear that FIPS and UPS outperform PS in terms of empirical computational complexity and computational efficiency. However, in terms of computation time to find a local minimum, PS can still have the advantage. If all nodes add computing power, PS can use this increase by making all nodes perform calculations. In contrast, FIPS and UPS only use the computing power of one node. PS is able to converge faster in terms of iterations because it always chooses the highest improvement. This means that, if no nodes become a bottleneck, PS is still able to be faster in terms of computation time than FIPS or UPS.

## VI. Conclusions

In this paper we proposed a new version version of Profile Steering (PS), called Flexiblity Informed Profile Steering (FIPS), in order to improve on computational efficiency, computational complexity and finding a local minimum.

In PS, every node is asked to find a candidate profile during every iteration. Only one of these candidate profiles is accepted. We proposed to determine the node that can optimize on the problem with an ordered list. We used the Shapley values found by [4] to determine the order of this list by either asking the flexible nodes first or the least flexible nodes first.

Asking nodes to optimize and accepting their profile without consideration of the other nodes their possible candidate profile has resulted in improved empirical computational complexity and efficiency compared to PS. In addition to this, we were able to find a different local minimum compared to PS, although the difference was minimal.

The order in which we ask nodes influences the found local minimum, the empirical computational complexity and the computational efficiency.

Asking the least flexible nodes first to optimize on the problem results in a slower convergence to a local minimum and a higher empirical computational complexity than asking

the most flexible nodes first. Asking nodes to optimize randomly results in a similar empirical computational complexity compared to asking the most flexible nodes first. Furthermore, asking nodes randomly results on average in the best computational efficiency compared to all other methods.

The computational efficiency of FIPS flexible first is lower compared to asking the devices at random. This is likely due to the estimation of Shapley values for nodes instead of calculating this value for every node.

In future work, we look at dynamically changing the order devices are asked to optimize by updating their Shapley values based on historical events.

## ACKNOWLEDGEMENTS

I would like to extend my sincere thanks to my supervisors Gerwin Hoogsteen and Aditya Pappu. Their feedback, knowledge and expertise helped me create this thesis.

## DECLARATIONS

During the preparation of this work the author used ChatGPT as a tool in order to write code more efficiently. After using this tool/service, the author reviewed and edited the content as needed and takes full responsibility for the content of the work.

## REFERENCES

[1] M. Hasanuzzaman, U. S. Zubir, N. I. Ilham, and H. Seng Che, "Global electricity demand, generation, grid system, and renewable energy polices: a review," *WIREs Energy and Environment*, vol. 6, no. 3, p. e222, 2017.

[2] M. E. T. Gerards, H. A. Toersche, G. Hoogsteen, T. van der Klauw, J. L. Hurink, and G. J. M. Smit, "Demand side management using profile steering," in *2015 IEEE Eindhoven PowerTech*, pp. 1–6, 2015.

[3] G. Hoogsteen, M. E. Gerards, and J. L. Hurink, "On the scalability of decentralized energy management using profile steering," in *2018 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, pp. 1–6, 2018.

[4] I. A. M. Varenhorst, J. Hurink, and M. E. T. Gerards, "Quantifying device flexibility with shapley values in demand side management," in *Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems*, e-Energy '24, (New York, NY, USA), p. 147–157, Association for Computing Machinery, 2024.

[5] A. Pappu, M. E. T. Gerards, G. Hoogsteen, and J. L. Hurink, "Asynchronous Distributed Energy Management with Co-operative Agents," 2024.

[6] G. Hoogsteen, A. Molderink, J. Hurink, and G. Smit, "Generation of flexible domestic load profiles to evaluate demand side management approaches," in *2016 IEEE International Energy Conference (ENERGYCON)*, (United States), p. 1279, IEEE, Apr. 2016. eemcs-eprint-27137 ; 2016 IEEE International Energy Conference, ENERGYCON 2016 ; Conference date: 04-04-2016 Through 08-04-2016.