Investigating the Effects of Changing the Optimization Order in Profile Steering

Dominic Pinto (2678462)
Faculty of EEMCS
University of Twente

***Abstract*—Energy management is crucial for the current energy transition. It can be employed to maintain a steady load profile, thereby optimizing grid efficiency. One way to achieve a steady load profile is through load management algorithms such as Profile Steering. Profile Steering is used to shape the power consumption profiles of devices towards an overall desired profile. However, the algorithm does not currently distinguish an optimization order for devices during each iteration and is not guaranteed to reach the optimal solution. Changing the order of optimization of devices within the algorithm could yield better results. One possible method to order devices based on their flexibility was implemented and tested. Sorting devices in ascending order of flexibility allows more flexible devices to compensate for devices with lower flexibility. This implementation can achieve a similar load profile to the original algorithm whilst reducing the computation time by a factor of 2.**

***Index Terms*—Power quality, scheduling, smart grids, algorithms.**

I. Introduction

The current energy transition denotes a paradigm shift, moving away from the traditional top-down centralized energy grid to an increasingly distributed and decentralized bottom-up grid hierarchy. The steady decline in the cost of distributed energy resources, coupled with supportive government policies and an increased desire for energy security, has resulted in a noticeable increase in electrification. However, the existing grid infrastructure was not designed to accommodate this gradually increasing load of bidirectional energy flow resulting from increasing deployment of distributed renewable energy sources. Accommodations need to be made to avoid overloading the current infrastructure. Either, the current infrastructure needs to be updated to support the increased load, which is a long process and would be expensive; or smart planning techniques in the form of Demand Side Management (DSM) algorithms. Profile Steering (PS) [1] is one such DSM algorithm that can be used to plan and optimize the energy profiles for sets of devices.

II. Problem and Research Question

Profile Steering (PS) is a heuristic algorithm designed to optimize the power profile for each grid hierarchy level. The algorithm currently requests improved profiles from all devices before accepting only a single improved profile that corresponds to the greatest improvement to the aggregated profile. One way around requesting all devices for improved profiles is to request for profiles in a certain order. To establish an order, the characteristics of devices that are beneficial to the algorithm have to be quantified. The main device characteristic to consider is flexibility, thus an optimization order based on flexibility could be a solution.
How does implementing an optimization order of the PS Algorithm affect its performance when considering the Euclidean distance to the desired profile?

- How do you calculate and implement a flexibility quantifier into the decision-making process of the Profile Steering Algorithm?
- How does the performance and efficiency of the ordered implementation of the PS Algorithm compare with the standard PS Algorithm?
- How does this affect the scalability of PS in terms of number of devices and number of houses?

III. Background Work

DSM approaches such as Price Steering and Profile Steering can be used to create desirable planning profiles tailored to individual consumers. Controllers that collect and steer the planning profiles of devices, called aggregators, use these algorithms to steer devices under their purview to shape the aggregated load profile to match the desired profile as closely as possible. Price Steering uses time-varying prices for a unit of energy to guide the aggregator. The aggregator then incentivizes devices to adjust their profiles to utilize cheaper intervals first. On the other hand, the PS Algorithm is a form of DSM where an aggregator uses steering signals based on the norm distance to the desired profile to guide devices.

1) Price Steering: An aggregator running the price steering algorithm can either use 'uniform pricing', where all houses get the same price signals, or 'differentiated dynamic pricing', where individual houses receive different price signals. Uniform pricing can serve as an incentive for devices to shift their consumption towards off-peak hours. However, using the same price signal for several houses can result in peaks simply being shifted in time. Therefore, different price signals are used for individual houses, incentivizing power consumption at different times. This approach is called 'Differentiated dynamic pricing' and results in individual houses having peak loads at different times and therefore a few steps closer to flat profiles. Nevertheless, when zooming into the individual profiles there are additional problems. The main issue with Price Steering is that it encourages extreme behaviour, such as high peaks during cheap intervals.

A. Profile Steering [1]

Another possible approach is Profile Steering. Given a device X which has a power profile $\vec{x} = [x_1, ..., x_N]^T$ and a desired profile $\vec{p} = [p_1, ..., p_N]^T$ both valid for N intervals, the PS Algorithm attempts to shorten the norm distance $\|\vec{x} - \vec{p}\|_*$ between $\vec{x}$ and $\vec{p}$. Consider a set of M devices with profiles $\vec{x}_m$ within a household with combined profile $\vec{x}$ and a desired profile $\vec{p}$ valid for N intervals. First, all devices individually minimize $\|\vec{x}_m\|$. Next, a difference vector is calculated as $\vec{d} = \vec{x} - \vec{p}$. The device planning profile $\vec{p}_m$ is then chosen such that $\|\vec{\hat{x}}_m - \vec{p}_m\|_2$ is minimized, where $\vec{p}_m = \vec{x}_m - \vec{d}$ and $\|\vec{\hat{x}}_m$ is a candidate planning for the device . The device which is able to provide the greatest improvement $e_m = \|\vec{x}_m - \vec{p}_m\|_2 - \|\vec{\hat{x}}_m - \vec{p}_m\|_2$ is chosen. Finally, the total consumption $\vec{x}$ is recalculated, and $\vec{x}_m$ is assigned its new value. This process is then repeated until $e_m < \epsilon$, where ϵ is an arbitrary set threshold. This algorithm can then be applied to every level of the grid hierarchy using a bottom-up approach.

The algorithm currently requests improved profiles from all devices and chooses the best, wasting the rest of the calculations. If there was some way to order the devices such that the algorithm could automatically request the device that would produce the best result in order, this would significantly reduce the required number of calculations. This can be achieved by sorting the devices. The rest of the paper explores the idea and implementation of sorting the device list by flexibility.

B. How do you quantify Flexibility?

1) Shapley Values [2]: One possible method to quantify flexibility and device worth to an aggregator is Shapley Values which we will now review using the scenario and equation given in [2]. Given a set of M devices, the marginal contribution $\xi(i)$ of a certain device $i \in M$ to a subset, or coalition, $S \subseteq M$ is used to quantify its worth. The marginal contribution can be defined as $\xi(i) = v(S \cup \{i\}) - v(S)$, with $v(S) = \|\vec{x_0}\|_2 - \|\vec{x}\|_2$, where $\vec{x_0}$ is the uncontrolled profile. However, devices interact differently with other devices and other coalitions. Therefore, Varenhorst et al. define an interaction index $I(v, \{i, j\})$ (shown in Equation 1) between two devices i and j, using their marginal contributions $\Delta_{\{i,j\}}$ (shown in Equation 2) to a coalition of T assets, to be used to determine the value of their synergy.

$$\Delta_{\{i,j\}}v(T) := v(T \cup \{i,j\}) - v(T \cup \{i\}) - v(T \cup \{j\}) + v(T) \tag{1}$$

$$I(v, \{i,j\}) := \sum_{T \subseteq M \setminus \{i,j\}} \frac{1}{m-1} \left(\; m - 2|T| \; \right)^{-1} \Delta_{\{i,j\}}v(T) \tag{2}$$

Using these defined terms, the Shapley value of i can be calculated for all possible subsets, $S \subseteq M$ assets, shown in Equation 3. However, this also means that the complexity of calculating the Shapley value scales exponentially with the number of assets M.

$$\phi_i(v) = \sum_{S \subseteq M \setminus \{i\}} \frac{|S|!(m - |S| - 1)!}{m!} [v(S \cup \{i\}) - v(S)] \tag{3}$$

2) FlexOffers [3]: FlexOffers [3] use power and time flexibility data to create a tuple, which is a potential method to quantify the flexibility of devices. A tuple can be defined for a device m as $f = ([t_{es}, t_{ls}], \vec{p})$; where t_{es} is the earliest possible start time, t_{ls} is the latest possible start time for the device, and p represents data about the maximum and minimum power consumption of the device.

Another key idea presented is the classification of flexibility into Time Flexibility (TF), Amount Flexibility (AF), and Time and Amount Flexibility (TAF). TF devices such as washing machines (WM) and dishwashers (DW) only show flexibility as time shifting, i.e. they have flexible start times but their power profiles cannot be changed. On the other hand, AF devices like heat pumps (HP) can have varying power profiles depending on thermostat settings and other factors, but do not have a clear start time and thus do not display time flexibility. TAF devices, like electric vehicles (EV), have clear start times and also show controllability in their power profiles.

IV. Analysis

A. Why sort by flexibility?

First, taking a step back, we first need an understanding of why we sort by flexibility. Looking at a simple scenario, including just a WM and an EV, we can find an intuitive sense of sorting in ascending order of flexibility. In standard PS, the device that provides the largest improvement during each iteration is selected to submit an improved profile. The EV, which has greater flexibility has a higher probability of having the greatest improvement during the first iteration. However, this does not allow the WM to aid in the improvement process. Let's assume the base profile is N discrete time intervals long where each interval is 1 hour and has a valley that is 3 intervals long at $t = 5$ to $t = 8$ and another valley that is 3 intervals long at $t = 10$ to $t = 13$. Both valleys have a depth of $3kW$ and can therefore store a total of $9kWh$ of energy each. Now let's also assume that $t_{es} = 0$ and $t_{le} = 7$ for the washing machine; and $0 \le t_{es} = 3$ and $10 \le t_{le} \le N$ for the EV. The washing machine has a fixed profile that is 3 intervals wide and $1kW$ high, for a total consumption of $3kWh$. The EV has a flexible profile but needs to charge a total of $15kWh$ before departure. During the initialization stage of the algorithms, both devices are assigned arbitrary profiles, usually based on greedy charging, resulting in peaks near the start of the profile. A sketch of the situation can be seen in Figure 1. For standard PS, the EV would win best improvement for the first iteration by moving its load from the start to partially fill both valleys by distributing its energy consumption equally between them. Both valleys would now only be $0.5kWh$ deep. In the next iteration, the WM would move its load to fill the first valley resulting in a $0.5kWh$ surplus. The second valley remains as is. A third iteration is then needed to shift some of the EV's consumption from the first to the second valley. Since every device is requested for an improvement profile during each iteration this means that 6 improvement requests ($3 iterations \times 2 devices$) were required to obtain the desired profile. However, if we sorted by flexibility and then requested improvements sequentially, the WM would move all consumption to the first valley and then the EV would fill the remainder of the two valleys. Thus requiring only 2 improvement requests instead of 6. Sorting by increasing flexibility allows higher-flexibility devices to compensate for lower-flexibility devices. How can this be implemented?

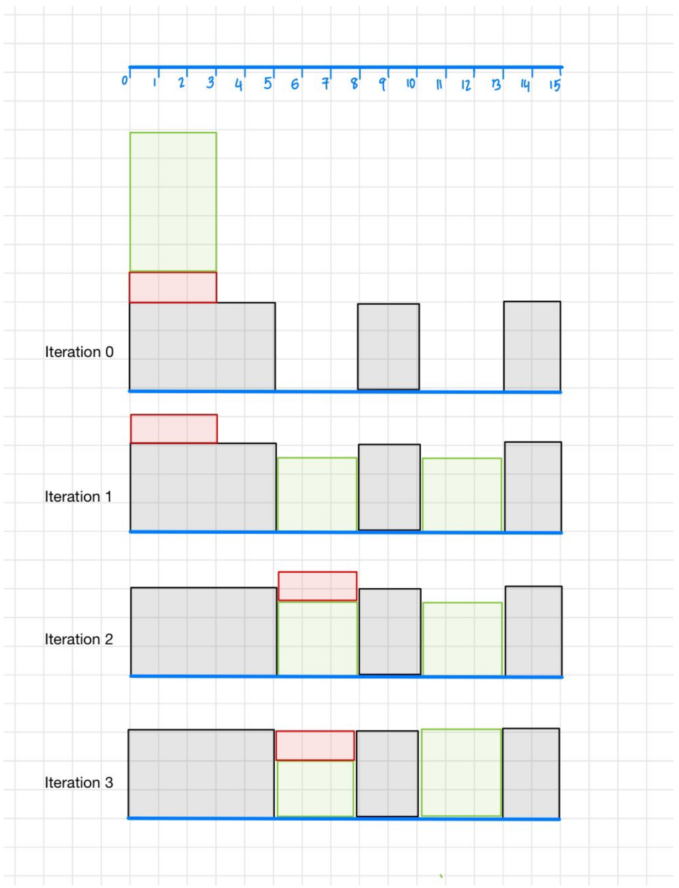

Fig. 1: Example Situation with PS (Black: Baseload, Red: WM, Green: EV)

B. Quantifying Flexibility using FlexValues

To create an optimization order based on flexibility, we first need to quantify flexibility for devices so that they can be sorted. By building on the foundation laid by FlexOffers [3] from Pedersen et al., we can define a flexibility quantifier for devices called FlexValues. Taking inspiration from FlexOffers, we see that the key device parameters to consider when quantifying the flexibility of a device are its power comsumption and start and end times. Additionally, another key concept drawn from Flexffers is that devices can be split into TF, TAF, and AF devices. Using these concepts, we now define a single value to sort devices called a FlexValue. A FlexValue, represented by V, is a unitless flexibility quantifier that is calculated uniquely for TF, TAF, and AF devices and is based on the device's power and time flexibility. The FlexValues for TF, TAF, and AF devices can be computed as follows:

a) Time Flexible Devices: For TF devices, the main device parameter to consider is their start time range. The Flexvalue (V) for TF devices can be calculated as:

$$V = (t_{ls} - t_{es}) \tag{4}$$

b) Amount Flexible Devices: For AF devices, which are continously available, the FlexValue can be calculated based on the maximum possible power consumption p_{MAX} and minimum possible power consumption p_{MIN}. FlexValue (V) for AF devices is therefore defined as:

$$V = (p_{MAX} - p_{MIN}) \tag{5}$$

c) Time and Amount Flexible Devices: Lastly, for TAF devices, we combine the two calculations previously done.

$$V = (p_{MAX} - p_{MIN})(t_{ls} - t_{es}) \tag{6}$$

C. Profile Steering Adapted (PSA)

To implement the optimization order based on flexibility, we must first adapt the PS algorithm such that it follows the sorted list of devices when requesting devices for improved profiles. This new variation on the PS algorithm will be called Profile Steering Adapted (PSA). An overview of the process of PSA can be found in Figure 2and is described in detail next.

First, devices are sorted by their device types, in order TF→TAF→AF, with TF devices at the start of the list and AF devices at the end. TF devices have the lowest flexibility (not always available and fixed power profile), then TAF in the middle (not always available but flexible device power profile) and lastly AF devices have the highest flexibility (always available and flexible device power profile). Next, devices of the same type are sorted based on their FlexValue V, which can be calculated using the equations from subsection IV-B. The controller then follows this list. It first creates an arbitrary planning for all devices in the list and computes an aggregated profile. Next, The controller requests devices, in order, for improved profiles and implements the improvement before moving on to the next device. A key distinction is made here between PS and PSA; in PS the algorithm requests optimized profiles from all devices for every iteration of the algorithm whilst only accepting the improved profile of the device that provides the most improvement to the overall profile. For M devices, over α iterations, where the number of computations to create a new optimal profile for device m is defined as Y_m; the total number of computations C_{Total} can be calculated as:

$$C_{Total,PS} = \sum_{\alpha} \sum_{M} Y_m \tag{7}$$

Since only one device is selected per iteration and all other computations are discarded, the number of wasted computations can be calculated:

$$C_{Wasted,PS} = \sum_{\alpha} \sum_{M-1} Y_{m,!best} \tag{8}$$

Equation 8 PSA on the other hand accepts all requested improvements which could reduce the number of wasted computations. Once an improvement from every device has been implemented, a single iteration has been completed. The controller then moves back to the start of the list and restarts, requesting profiles from lowest to highest flexibility once again. It continues to do this until the overall improvement in an iteration falls below a preset threshold, e_{min}.

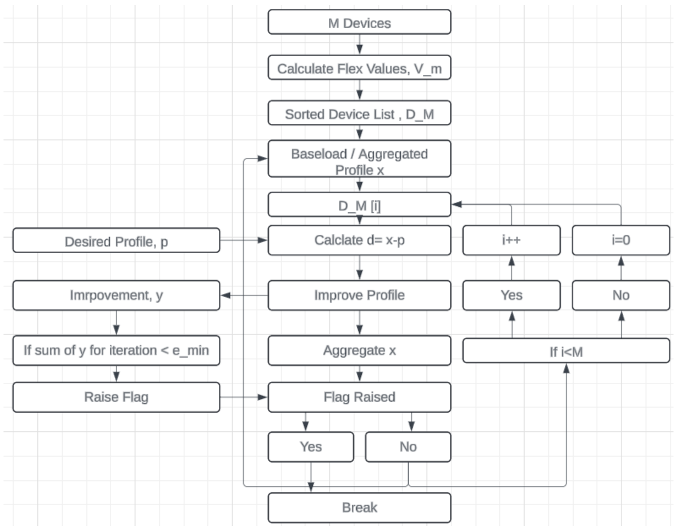

Fig. 2: Profile Steering Process

D. Scaling up FlexValues with FlexProfiles

To calculate FlexValues for larger systems, such as houses or neighborhoods, is more complicated since they contain several devices within them. Therefore we must define a method to calculate a flexibility quantifier for systems which hold multiple devices, that is able to account for the flexibility of the devices within that system. To be able to calculate a FlexValue for such a system, we first introduce FlexProfiles, represented by $F(V)$. Flexprofiles are vectors containing the maximum and minimum power consumption of the flexible devices encompassed by the system. FlexProfiles can be generated for all devices and then summed together to get combined FlexProfiles for larger systems, such as houses and neighbourhoods. The final FlexProfile for a system can then be summed to get a FlexValue for the system. For a set of intervals N, with M devices within a house, where $p_{max,m,n}$ is the maximum possible power consumption of a device m during interval n.

$$F(V)_{max,m} = [p_{max,m,0}, ..., p_{max,m,N-1}] \tag{9}$$

The FlexProfile for the entire house can then be given to be

$$\vec{V}_{max} = [\sum_{m \in M} p_{max,m,1}, ..., \sum_{m \in M} p_{max,m,N}] \tag{10}$$

$$\vec{V}_{min} = [\sum_{m \in M} p_{min,m,1}, ..., \sum_{m \in M} p_{min,m,N}] \tag{11}$$

The final FlexValue for a house is then given by:

$$V_{house} = \sum_{n \in N} F(V)_{max,n} - F(V)_{min,n} \tag{12}$$

FlexProfiles are necessary to calculate FlexValues for higher levels of the grid hierarchy and thus must be calculated and stored for every level of the hierarchy. For example, FlexProfiles of devices are used to calculate FlexProfiles of their houses, which are used to calculate the FlexValues of those houses. FlexProfiles houses are then used to calculate FlexProfiles of neighborhoods, which can be used to calculate the FlexValues of the neighbourhoods. The calculated FlexValues can then be used to sort houses within a neighborhood, or neighbourhoods within cities.

V. Evaluation

To test the effectiveness of the algorithms and provide fair comparisons it is key that any profiles used for testing are realistic and representative of actual households, and that the same profile is used when comparing. Therefore testing was conducted based on data generated by the ALPG (Artificial Load Profile Generator) [4]. This tool allows realistic modelling of environmental and lifestyle factors of various household types with varied number of occupants and domestic situations. Additionally, it is also able to provide flexibility data to allow planning of smart devices. Therefore it is ideal to create test scenarios which can be used to evaluate and compare the performance of the PSA algorithm.

A. Phase 1: Single Household Planning

The initial implementation, intended as a foundational proof of concept, will use artificial load profiles generated by the ALPG to create a simple test scenario. To do this a custom household with a single worker occupant was set up for the ALPG, including an EV, WM, and DW. The driving distance, initial state of charge (SoC), battery capacity, start times and end times, are all determined by the algorithm based on the household type and environmental factors such as the weather, day of the week, etc. Then load profiles and flexibility data are generated for a single household of this type and passed on to the PSA code

PSA was implemented using the PS Lite framework which already includes structures to represent various device types. Whilst device characteristics and planning methods vary drastically from device to device, each structure essentially includes three main functions: 'initialize', 'plan', and 'accept'. The 'initialize' function creates an arbitrary planning, usually choosing to start at the first available start time. The 'plan' function finds the optimal device profile given a steering signal. Finally, the 'accept' function changes the current profile to the one created by the 'plan' function. A layer of abstraction is achieved by standardizing function names across device structures; allowing different structures to be placed in lists. Since the algorithms do not require knowledge about lower-level optimisation, this abstraction reduces code complexity. Lists containing these devices can then be generated using profile and flexibility information from the ALPG. These structure lists are given as inputs to the PSA algorithm along with the desired power profile $\vec{p}_{desired}$, the minimum improvement

required per iteration to continue e_{min}, and the maximum number of iterations I_{max}. The PSA algorithm takes the device list and calculates the FlexValue V for all devices using the equations shown in subsection IV-B. It then sorts the by device type and category. The rest of the PSA code is implemented as described by Figure 2 with the improve profile step being done using the 'plan'+'accept' functions for each device.
To evaluate the performance of PSA, we can compare it to the standard PS algorithm. Key indicators of performance to be compared will be time taken, number of computations, and Euclidean norm of final profile.

a) Time Taken: Time taken to to run the algorithm from start to finish can be measured and compared. However, this brings some concerns. Firstly, whilst some process can be run in parallel in PS, all processes are sequential in PSA. In PS, during each iteration the controller could theoretically request all devices for an improved profiles simultaneously. If there is enough computational power available, multi-threading could be used to possibly speed up the PS algorithm. Additionally, if computation can be offloaded to devices the time required for each iteration now reduces to the slowest improvement calculation with some added communication time. PSA on the other hand, needs to sequentially request for profiles and therefore would benefit from parallelization, but would not benefit from offloading computation. However, in this scenario, we can assume that computation is not offloaded to devices as most smart devices do not have much computation power. Due to these effects, the absolute time taken cannot be used as a metric, but rather as a general trend; i.e. see which algorithm is faster for a given situation.

b) Number of Computations: Number of computations can be estimated by counting the number of times the algorithm call the 'plan' function. Ideally, the number of computations should be as low as possible which would correspond to an algorithm with a higher efficiency i.e. the algorithm that requires fewer computations to achieve the same result is more efficient.

c) Euclidean Norm: The Euclidean norm is a measure of profile flatness. For this scenario the flatter the profile, the better, as this is one of the main goals of Profile Steering. A flatter profile means a lower euclidean norm. Therefore the euclidean norm should be as small as possible. Ideally, PSA would be able to produce the same, if not lower, norm as PS.

B. Phase 2: Scalable Planning

The next phase entails implementing measures to scale up the PSA and PS algorithms. This can be done by nesting. Another device class is created (with the same functions 'initialize', 'plan', and 'accept') to represent larger systems which hold several devices. For simplicity, we will only consider houses rather than higher grid hierarchy levels. Since this new class is identical in structure to the way the devices are defined, the algorithm is unable to distinguish between a list of 'Houses' and a list of devices. This allows the higher-level algorithm to control the houses, whilst an algorithm nested within each house controls devices within the given house and attempts to conform to a desired profile specified by the higher-level algorithm. This is shown in Figure 3. Therefore the 'House' class serves as both an aggregator as well as a controller. One key distinction to be made here is the difference in the way the nesting must be done between both algorithms.

For PSA, nesting is much simpler because the improvements requested are always accepted. For PS, this is significantly harder. Since only the best improvements are accepted, records of unimproved profiles need to be kept to ensure that all unaccepted changes can be reverted. Bookkeeping can be achieved by saving chosen profiles once the 'accept' function of the higher-level PS algorithm. At the start of every iteration of the algorithm, all devices can then be restored to the last chosen profile.
To evaluate the performance of the PSA algorithm when using nesting, the same metrics from Phase 1 can be used. They can also be used show how scalable the PS algorithm is as compared to the PSA algorithm.

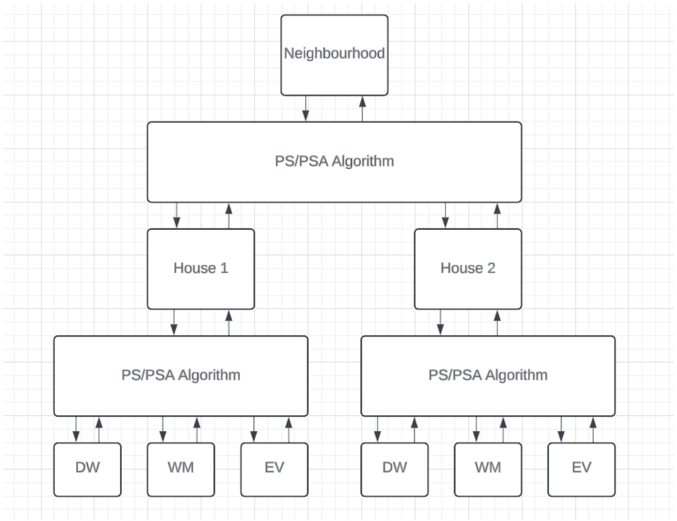

Fig. 3: Nesting in the Grid Hierarchy

VI. Results

A. Simulation Parameters

To simplify testing and verification of the implementation of both phases, we must define a few of the test parameters. Interval length is set at 1 minute. This is chosen so that the power profiles have enough precision to represent real household consumption, including devices that may only be on for a few minutes at a time (for example kettles). Additionally, the intervals are long enough that the computational complexity is still manageable. The number of iterations for any algorithm is limited to 20. This keeps the time taken for computations down even when scaled. In theory this could be varied to achieve better results but additional research is required for this.

	House 0 Norm	House 1 Norm	Time (s)
PSA	22887	31507	0.7899
PS	22910	31419	1.8315

TABLE I: Phase 1 Results, Two Houses

B. Phase 1: Single Household Planning

A simple scenario can be generated using the ALPG to test the phase 1 PSA implementations. The scenario includes 2 houses, each with a dishwasher, washing machine, and electric vehicle. The flexibility information for these devices is also generated imported into the test program. The test program creates devices for the houses and assigns flexibility data from the ALPG to them. It is also able to call code to compute the FlexValue of each device. The list of devices for each house is then sorted, first by type of device (in the order $TF \rightarrow TAF \rightarrow AF$) and then by ascending order of FlexValue. Therefore devices with the lowest flexibility are the lowest in the list. Both houses are then initialized and run through both the PS and PSA algorithms. The results of this test are shown below in Table I. These initial tests clearly show that both algorithms achieve approximately the same final profile Euclidean norm. However, the PSA algorithm can achieve the result in around half the time. this is due to the reduced number of calculations that are wasted.

Next, for a larger scale test, the number of devices within the household was scaled. The time taken,final Euclidean norm, and number of improvement requests of both the PS and PSA algorithms was measured while increasing the number of devices within the household. For a fair comparison, the baseload was kept constant and all devices were randomaly generated using the ALPG. The outcome of this test is plotted in Figure 4, Figure 5, and Figure 6.

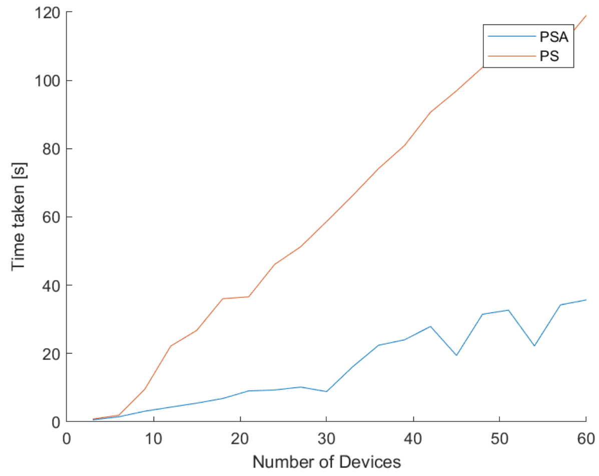

Fig. 4: Computation time vs Number of Devices for PS and PSA

Figure 4 shows that PSA shows around a $2\times$ to $2.5\times$ improvement in speed when compared to PS. Whilst the

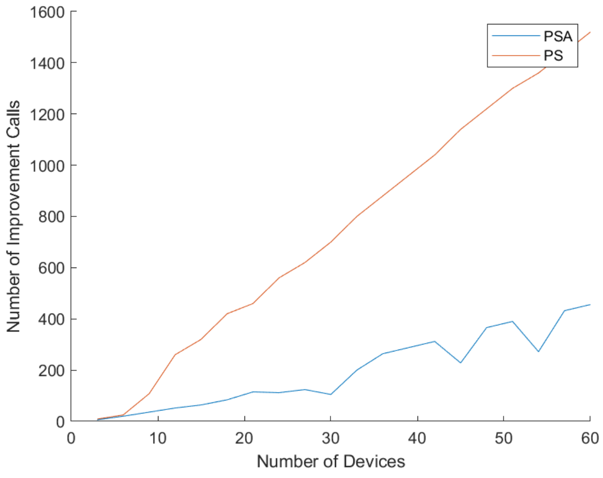

Fig. 5: Number of Computations vs Number of Devices for PS and PSA

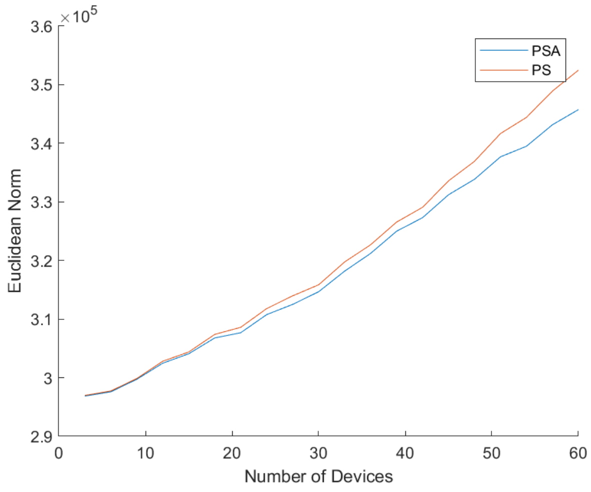

Fig. 6: Output Euclidean Norm vs Number of Devices for PS and PSA

absolute time for the calculation depends on the computation device, it is clear that the PSA algorithm is faster for this particular scenario and test conditions. Additionally, Figure 6 shows that both algorithms give approximately the same final Euclidean norm, showing that both algorithms are equally effective. Figure 5 shows the number of improments requested by the algorithm. The trend clearly shows that PSA needs less than half as many improvement requests to achieve the result. This implies that the PSA algorithm is more efficient than standard PS.

C. Phase 2: Scalable Planning

To test the phase 2 implementations of scalability, the same test profiles were used. However, in this scenario, the houses no longer work independently but rather as a neighbourhood. This means that the houses can compensate for each other and

	House 0 Norm	House 1 Norm	Time (s)
PSA	22887	31507	0.7899
PS	22910	31419	1.8315

TABLE II: Phase 2 Results: Houses compensating for each other

can create a flatter combined profile. The result for this test is shown in Table II. Both algorithms give roughly the same combined Euclidean norm, but PSA achieves it slightly faster. For a larger scale test, the time taken and output Euclidean norms were measure for both algorthms whilst increasing the number of houses. To reduce complexity, all houses have the same 3 devices: an EV, WM, and DW. The mamximum number of iterations for both the higher-level and nested algorithms is set at a lower value of 10 to keep computation time manageable. The flexibility and baseload information for all houses is unique and generated using the ALPG. The results of the test are shown in Figure 7 and Figure 8.
Figure 7 shows that the PSA implementation is generally quicker than PS. However this could be misleading, because in a real world situation a central controller would not perform all computations, but rather offload house level improvement requests to the house controllers. This means that the PS algorithm could simultaneously request improvements from all houses and thus drastically improve the performance of the PS algorithm. Figure 8 shows that both algorithms are equally effective at reducing the Euclidean norm of the houses.

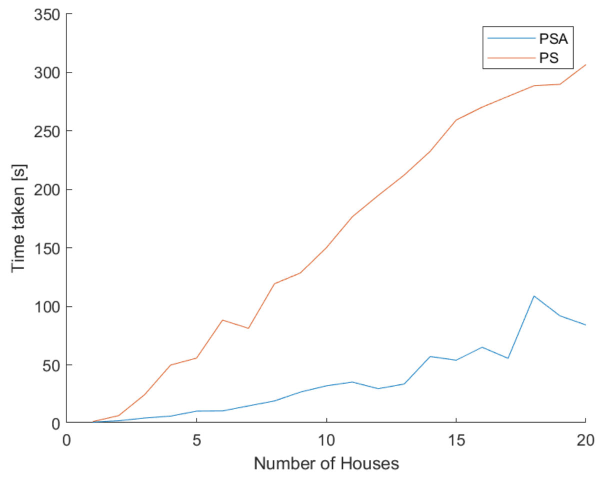

Fig. 7: Computation time vs Number of Houses for PS and PSA

VII. Conclusion

In conclusion, changing the optimization order in Profile Steering based on Profile Steering was effective. Sorting devices based on FlexValues, which can be calculated based on a devices time and power flexibility data, resulted in an increase in 2× reduction in computation time and efficiency

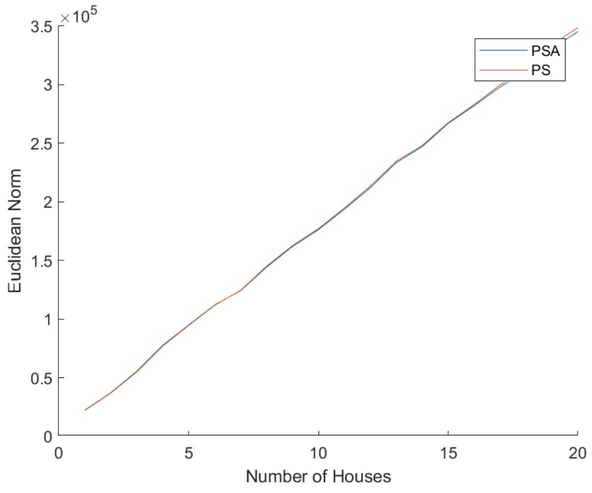

Fig. 8: Output Euclidean Norm vs Number of Houses for PS and PSA

whilst still achieving the same Euclidean norm. Additionally, using FlexProfiles to calculate FlexValues for houses, and using the FlexVales to sort houses for nested algorithms was also successful. Both algorithms were able to reach the the similar Euclidean norms. However, the time data obtained is misleading. Further testing with more realistic conditions is needed to compare the time taken by both algorithms. Additional research is also required into hybrid nesting algorithms for scalability. Since PS allows more parallelization, it may be beneficial to use it for higher levels of the grid higherarchy where computations are offloaded. On the other hand, at the house level, where no computational offloading is possible, it is more beneficial to use PSA. PSA is faster within a house and scales better with increasing devices. Research needs to be done into the most optimal configuration, optimal maximum number of iterations and optimal minimum improvement for PS and PSA based hybrid nested algorithms.

VIII. Declarations

During the preparation of this work, the author used GitHub Copilot for generative coding. Any code added, modified, or suggested by GitHub Copilot has been reviewed by and was according to the intent of the author. Additionally, Grammarly and Overleaf's built-in spell check were used to improve grammar and remove typing errors. After the use of these tools and services, the author has reviewed and edited all content as needed and takes full responsibility for the content of the work.

References

[1] 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*, 2015, pp. 1–6.
[2] I. A. M. Varenhorst, M. E. T. Gerards, and J. L. Hurink, "Quantifying device flexibility with shapley values in demand side management." 2024.

[3] T. B. Pedersen, L. Siksnys, and B. Neupane, “Modeling and managing energy flexibility using flexoffers,” *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, Oct 2018.

[4] G. Hoogsteen, A. Molderink, J. L. Hurink, and G. J. Smit, “Generation of flexible domestic load profiles to evaluate demand side management approaches,” *2016 IEEE International Energy Conference (ENERGYCON)*, Apr 2016.