Aggregating Energy Flexibilities under Constraints

Emmanouil Valsomatzis and Torben Bach Pedersen
Aalborg University
Email: {evalsoma, tbp}@cs.aau.dk

Alberto Abelló
Universitat Politècnica de Catalunya
Email: aabello@essi.upc.edu

Katja Hose
Aalborg University
Email: khose@cs.aau.dk

***Abstract*—The flexibility of individual energy prosumers (producers and/or consumers) has drawn a lot of attention in recent years. Aggregation of such flexibilities provides prosumers with the opportunity to directly participate in the energy market and at the same time reduces the complexity of scheduling the energy units. However, aggregated flexibility should support normal grid operation. In this paper, we build on the flex-offer (FO) concept to model the inherent flexibility of a prosumer (e.g., a single flexible consumption device such as a clothes washer). An FO captures flexibility in both time and amount dimensions. We define the problem of aggregating FOs taking into account grid power constraints. We also propose two constraint-based aggregation techniques that efficiently aggregate FOs while retaining flexibility. We show through a comprehensive evaluation that our techniques, in contrast to state-of-the-art techniques, respect the constraints imposed by the electrical grid. Moreover, our techniques also reduce the scheduling input size significantly and improve the quality of scheduling results.**

I. Introduction

One of the main goals of the Smart Grid is the energy use increase from Renewable Energy Sources (RES). However, due to RES being characterized by volatile power production (e.g., wind power), Smart Grid takes advantage of the prosumers' inherent flexibility to better match energy demand with supply, termed Demand Response (DR), and thus enables an increased share of RES energy.

In our work, we model flexible demand/supply devices (referred to as *loads* for simplification) using the flex-offer (FO) concept [1]. An FO explicitly captures the flexibility in energy and time of a load, as presented in the following example.

Example 1. The owner (consumer) of an electric vehicle (EV) wants to charge his EV at 20:00 and have it charged by 7:00 the following day. The EV takes 3 hours to be charged and requires 15kWh. Thus, the EV can start its charging between 20:00 and 4:00.

The number of loads that are flexible has recently increased due to new technological achievements (e.g., EVs and heat pumps). The existence of appropriate information and communication technology (ICT) infrastructure [2] and a suitable hierarchical control architecture, offer the capability to market actors to command the DR [3]. Moreover, the establishment of a flexibility market [4] will provide flexibility with the opportunity to be traded [5]. However, the energy captured by individual FOs from small load devices cannot be directly traded in the market [6]. For instance, the power required to participate in the ancillary service market in Denmark is in the magnitude of few hundreds of kW where the consumption capacity of an EV is few kW [6]. Thus, in order to trade flexibility, it is essential to aggregate FOs and produce commodities that can be traded in the emerging energy flexibility markets. Furthermore, aggregation of FOs, applied before scheduling, is essential to reduce the highly complex Unit Commitment (UC) problem [7]. According to the UC problem, FOs are scheduled, i.e., the operational time and amount is defined, based on an objective function.

On the other hand, flexible loads and, consequently, their corresponding FOs are connected to an electrical grid. However, the grid is characterized by power capacity limitations and the high power requirements of new devices, such as EVs, might lead to grid congestions. Grid sensitive load locations (bottlenecks) are in different voltage elements. They could be in low (local distribution) and in high voltage elements (supra-regional distribution). For instance, a bottleneck might be a distribution transformer (0.4-1kV) with a maximum power value of few hundred kW. Such a transformer might serve from few (e.g., in North America) to several hundred households (e.g., in Europe) [8].

In our work, we follow the mapping applied in [9] and map a bottleneck to the root of a tree, see Ⓡ in Figure 1. The root is characterized by an amount constraint that defines the tolerable operational power range. For instance, the power of a distribution transformer (0.4kV) shall be in the interval [-300kW, 300kW] [3]. We also map all FOs, which belong to the bottleneck, to the leaf nodes, see ① in Figure 1. The leftmost circle in the figure illustrates an FO corresponding to the load of an EV. The x-axis represents time and the y-axis represents power. The energy required for charging the EV is expressed by three slices (one per time unit). The dark-shadowed parts represent the minimum energy requirements. The light-shadowed parts represent optional charging levels. For instance, the EV owner is satisfied when charging level is in the range $[60\%, 100\%]$. Moreover, as we see in the figure, charging of the EV can start at time 1 at the earliest (t_{es}) and at time 5 at the latest (t_{ls}). Thus, the FO profile, which consists of the three slices, can be time-shifted.

Using traditional aggregation techniques [10], the FOs are aggregated resulting in aggregated FOs (AFOs). As illustrated in Figure 1, the four FOs ① are aggregated into two AFOs ②. Each profile of an AFO is produced by summing up one or more profiles of the 4 FOs. Without considering constraints, loads might be placed at the same time since it may be more beneficial, e.g., from a financial point of view. However, this could lead to violations. For instance, we see that the power

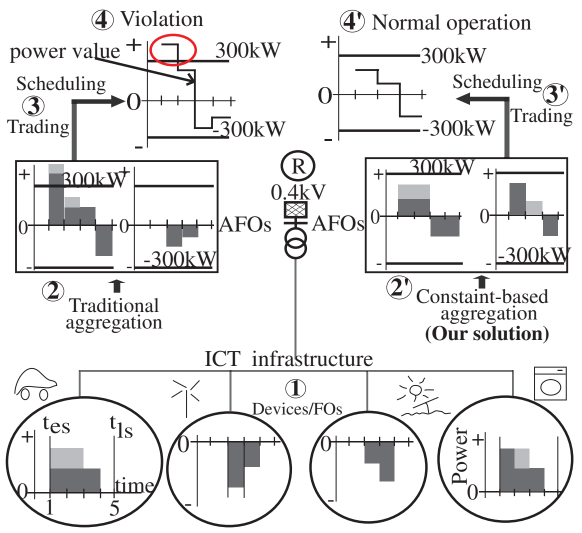


Fig. 1: Traditional vs Constraint-based aggregation.

of the left AFO (first dark-shadowed slice in ②) exceeds the constraint imposed by the grid. After being aggregated, the AFOs are traded and scheduled, see ③. Scheduling transforms AFOs into *assignments* and forms the root power value. However, it is impossible to schedule the output of traditional aggregation and to respect the constraint. Thus, scheduling leads to a constraint violation due to inappropriate aggregation, see ④ where the power value exceeds 300kW in the first time slot (red circle). Consequently, FO aggregation techniques that take into account grid constraints are required. In this paper, we propose such constraint-based aggregation which produces AFOs that can be further scheduled and support a normal grid operation, see ②'-④' in Figure 1.

Contributions. First, we demonstrate the problems that occur with traditional FO aggregation. Second, we introduce the objectives of constraint-based FO aggregation and propose two heuristic aggregation techniques that reduce the input by more than 90% while retaining flexibility. Third, we evaluate the proposed techniques in complex use case scenarios. We show that our techniques lead to normal grid operation where the existing state-of-the-art approaches lead to grid constraint violations at more than 15% of the examined time horizon. Finally, we show that in cases where scheduling *cannot* provide a schedule that respects grid constraints within a certain time period, our aggregation techniques efficiently narrow down the solution space and thus lead to valid scheduling results.

The remainder of the paper is structured as follows. Section II introduces relevant concepts and definitions. Section III discusses the problems of traditional aggregation and introduces constraint-based aggregation objectives. In Section IV, the two constraint aggregation solutions are proposed. Their experimental evaluation is described in Section V. In Section VI related work is discussed. Finally, the paper concludes and points to future work in Section VII.

II. Background and Preliminaries

Based on [10] and using two discrete dimensions, i.e., time and amount, we define the following.

Definition 1. An *FO* f is a tuple $f = (T(f), P(f))$ where $T(f)$ is the start time flexibility interval and $P(f)$ is the amount profile. $T(f) = [t_{es}, t_{ls}]$ where t_{es} and t_{ls} are the *earliest start time* and *latest start time*, respectively. The *amount profile* is a sequence of ($m \in \mathbb{N}_{>0}$) consecutive slices, $P(f) = \langle s^{(1)}, \dots, s^{(m)} \rangle$ where a *slice* $s^{(i)}$ is an amount range $[a_{min}, a_{max}]$. The duration of slices is 1 time unit. For instance, Figure 2 illustrates FO $f = ([1,5], \langle [3,5], [2,3] \rangle)$.

We distinguish two types of flexibilities associated with an FO that are used as individual measures taking into account time and amount separately. We consider *time flexibility* $tf(f)$ of an FO f to be the difference between its latest and earliest start time, i.e., $tf(f) = t_{ls} - t_{es}$. Moreover, we consider *amount flexibility* $af(f)$ of an FO f to be the difference between the sum of all the maximum and minimum values of all its slices, i.e., $af(f) = \sum_{s \in P(f)} (s.a_{max} - s.a_{min})$. Time flexibility is measured in time units and amount flexibility in amount units.

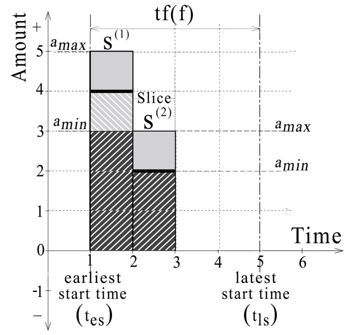


Fig. 2: A flex-offer f

An FO captures all possible amount demands and/or supplies of a device for a given time horizon. However, during the scheduling process, an FO is assigned to a specific amount at a specific time resulting in an *assignment* of the FO defined as follows:

Definition 2. An *assignment* of an FO f is a sequence of $|P(f)| \in \mathbb{N}_{>0}$ consecutive slices, $as\_f = \langle s^{(1)}, ..., s^{(|P(f)|)} \rangle$. Each slice is a 2-tuple, $s^{(i)} = (ts, am), i \in [1, |P(f)|]$. The first element, ts, indicates the actual starting time and the second one, am, the actual amount of the slice. The duration of each slice is 1 time unit.

The starting time of the first slice of the *assignment* must be within the start time flexibility interval of the FO, i.e., $f.t_{es} \leq as\_f.s^{(1)}.t_s \leq f.t_{ls}$. Each slice of the *assignment* has an amount value in the range of the corresponding slice of the FO, i.e., $f.s^{(i)}.a_{min} \leq a\_f.s^{(i)}.am \leq f.s^{(i)}.a_{max}$, $\forall i = [1 \dots |P(f)|]$. There is a finite number of assignments of an FO. We denote the set of all the assignments of an FO f by $L(f)$.

III. Problem Formulation

In this section, we discuss how aggregation is applied through traditional aggregation and introduce the concept of constraint-based aggregation.

A. Traditional FO aggregation

We consider, based on [10], traditional aggregation of FOs to be the function that given a set of FOs returns an aggregated one, taking into account the time and amount flexibilities of the FOs. Given a set of FOs, there are different alignments

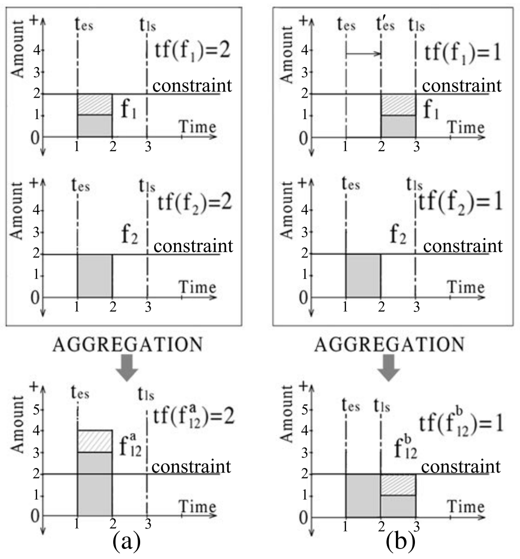


Fig. 3: Different alignment examples for aggregation.

that lead to different AFOs due to their time flexibility. In particular, given $|F|$ FOs with time flexibility $tf(f_1), \ldots, tf(f_{|F|})$ respectively, the number of the aggregation results (AFOs) that can be produced is: $\prod_{i=1}^{|F|} tf(f_i) + 1$. For instance, the 2 FOs, f_1 and f_2 in Figure 3, can be differently aligned and result in different AFOs. Thus, for f_1 and f_2 with both obtaining 3 different start times, there are $3 \cdot 3 = 9$ alignments that lead to 9 aggregation results (AFOs). We show 2 of them in Figure 3.

The time flexibility interval of an AFO is determined by the chosen alignments. In particular, the amount profile of an FO does not have any specified starting time until the FO is assigned. However, an FO captures all the different starting times in the start time flexibility interval, see Definition 1. As a result, when aggregation is applied, FOs that participate in aggregation are aligned (a starting time among the interval is chosen for every FO) and the amount ranges of each aligned slice are summed, see Figure 3. We denote the aggregation that aligns FOs according to their earliest start time as *Start Alignment* (SA) aggregation, see Figure 3a. According to SA aggregation, the earliest starting time of the AFO is the minimum earliest starting time of the non-aggregated FOs. The latest starting time of the AFO is the sum of its earliest starting time and the minimum time flexibility among the FOs. As a result, the AFO respects all the starting time intervals of the non-aggregated FOs that produced it. For instance, the AFO f^a_{12} in Figure 3 has earliest starting time 1 ($f^a_{12}.t_{es} = min(f_1.t_{es}, f_2.t_{es})$). The latest starting time (t_{ls}) of f^a_{12} is equal to 3 ($f^a_{12}.t_{ls} = f^a_{12}.t_{es} + min(tf(f_1), tf(f_2))$).

B. Constraint aggregation objectives and complexity

As mentioned in Section I, the value of a node (actual load) is given by the assignments of the FOs that belong to the node. In particular, during scheduling each FO is turned into an assignment and the result is a set of assignments. Consequently, the sum of the slice amounts with the same time forms the node value at that time. However, in order to guarantee a normal grid operation, the actual loads of the grid must be within the bounds imposed by the constraint, e.g., [-300kW, 300kW]. For instance, concurrently charging a high number of EVs can lead to transformer overload.

We assume that FOs f_1 and f_2 in Figure 3 belong to a node with constraint value 2. Moreover, we see that the aggregation result (f^a_{12} last row column a) of SA *does not* enable an assignment that respects the constraint. When scheduling is applied on f^a_{12}, there are several potential assignments of f^a_{12}, e.g., $as\_f^{a1}_{12} = (1,3)$ and $as\_f^{a2}_{12} = (2,4)$, see Figure 3a. However, the constraint value is 2 and the amounts of all the assignments are greater than the constraint. They should have been within the range [-2,2]. Conversely, we see that when FO aggregation takes into account the constraint, it produces AFO f^b_{12} (Figure 3b) that contains assignments which respect the constraint, e.g., $as\_f^b_{12} = \langle(2,2),(3,1)\rangle$. In this paper, we evaluate an aggregation result through the objectives of constraint-based aggregation.

Constraint-based FO aggregation has 3 objectives. The produced AFOs (1) shall enable scheduling results that respect the constraint of the node where the FOs belong (hard constraint). Moreover, (2) aggregation should retain as much flexibility as possible and (3) at the same time reduce the number of FOs that belong to a specific node.

1) Respect node constraints. All node constraints should be respected. A node constraint violation corresponds to a grid malfunction at the point where the node is. That results in service cutoff of FOs that belong to the violated node and thus the prosumers might not be served.

2) Minimize flexibility losses. Flexibility of FOs is important for scheduling because the more flexible FOs are, the more degrees of freedom the scheduling has to find the optimal solution. Moreover, AFOs capture larger flexibilities and can more easily be traded in the energy market. We use flexibility as a quality measure to evaluate our proposed techniques, as AFOs might lose flexibility during aggregation.

3) Minimize the number of AFOs. FOs are part of the scheduling input that takes place after aggregation. Therefore, it is important for constraint aggregation to reduce the number of FOs, because it directly reduces the complexity of the subsequent scheduling. Moreover, unless FOs are aggregated to capture large energy amounts, they cannot be traded in the energy market.

The above-mentioned objectives might be contradictory and cannot be satisfied simultaneously. In particular, as the number of AFOs is reduced, time flexibility losses might increase and time flexibility might be used to respect the constraint. For instance, we see in Figure 3a that f_1 and f_2 have time flexibility 2. However, AFO f^b_{12} has $tf(f^b_{12}) = 1$.

Constraint aggregation complexity. Due to space limitations, we illustrate the computational complexity of constraint-based aggregation through an example. In our example, given a set of FOs, we compute the total solution space, i.e., the number of all the potential aggregation results.

Example 2. Given a set F of 4 FOs, f_1, f_2, f_3, f_4, with $tf(f_1)$=3, $tf(f_2)$=2, $tf(f_3)$=4, $tf(f_4)$=5, there are

$B_4 = \sum_{k=1}^{4} \left\{ {4 \atop k} \right\} = \frac{1}{1!}(-1)^1\binom{1}{0}0^4 + \frac{1}{2!}\sum_{j=0}^{2}\binom{2}{j}j^4 + \frac{1}{3!}\sum_{j=0}^{3}\binom{3}{j}j^4 + \frac{1}{4!}\sum_{j=0}^{4}\binom{4}{j}j^4 = 1+7+6+1 = 15$, partitions of F [11]. Moreover, there are $\prod_{i=1}^{4} tf(f_i) = 3 \cdot 2 \cdot 4 \cdot 5 = 60$ alignments. Thus, there are $15 \cdot 60 = 900$ possible aggregation results.

Adding a fifth FO to the set with $tf(f_5) = 5$, there are B_5=52 partitions of F and $\prod_{i=1}^{5} tf(f_i)$ =$60 \cdot 5$=300 alignments. Thus, there are $52 \cdot 300 = 15600$ possible aggregation results. Therefore, we can notice a *combinatorial explosion* of the aggregation results depending on the size of the input and its average time flexibility.

IV. Constraint-based FO Aggregation

Due to the high complexity of constraint-based aggregation, we analyze two variations of a greedy solution to tackle the problem. In particular, the greedy approaches process FOs that belong to a node incrementally by evaluating binary aggregations. Evaluation is based on different metrics in order to examine whether further aggregation is favored or not. The metrics take into account both the capacity limitations of the node and the objective of the market actor who controls the FOs of the node.

A. Constraint and target related distances

As mentioned in Section I, in order to guarantee a normal grid operation, the node value shall be within the bounds imposed by the constraint. In this paper, we handle the constraint as a function.

Definition 3. We define a (constant) positive constraint function $c(t) = y$, $t \in \mathbb{Z}$, $y \in \mathbb{N}_0$, where t is the time and y the amount.

For instance, given a constraint function $c(t)$=300, the valid amount range is [-300, 300]. In cases where the node value is outside the constraint bounds, a node violation occurs, see for instance ④ in Figure 1. When this happens, the electrical grid is not reliable and the distribution system operator, who is responsible for the grid, needs to expand and update the power system infrastructure. Updating the grid is a very expensive and time consuming procedure. In our work, since the node value is formed by the assignments of the FOs, we correlate an assignment to the constraint function. We consider the distance of each slice of an assignment (positive or negative) to be zero when it is within the range because no grid problems occur. Otherwise, we take into account the distance to the constraint function.

Definition 4. We define the *distance of a slice of an assignment*, $D_c(as\_f.s^{(i)})$, to a constraint function c as equal to zero if the absolute slice amount is smaller or equal to c. Otherwise, $D_c(as\_f.s^{(i)})$ is equal to the difference between the absolute amount value of the slice and the constraint, i.e., $D_c(as\_f.s^{(i)}) = max(0, |s^{(i)}.am| - c(s^{(i)}.ts))$ where $as\_f = \langle s^{(1)}, ..., s^{(|P(f)|)} \rangle$ and $i \in [1, |P(f)|]$. Consequently, we define *distance of an assignment* as_f, $D_c(as\_f)$, to a constraint function c to be the sum of all its slice distances to c, i.e., $D_c(as\_f) = \sum_{i=1}^{|P(f)|} D_c(as\_f.s^{(i)})$.

The objective of the market actor controlling the FOs of a node, e.g., an aggregator, is formulated through a target function. Target expresses the optimal schedule, without considering the constraint, and can be used to represent an optimal business goal, e.g., optimal price/amount correlation. Target might contradict the constraint and it could lead to AFOs with assignments that violate the constraint, see for instance ② in Figure 1. We define both the target function and the assignment distance to the target function as follows:

Definition 5. We define a (constant) signed target function $g(t) = a$, where $t \in \mathbb{Z}$ is the time and $a \in \mathbb{Z}$ the amount.

Definition 6. We define the *distance of an assignment* as_f to a target function g, $D_g(as\_f)$, as equal to the sum of the absolute differences between g and the amount values of all the slices of the assignment, i.e., $D_g(as\_f) = \sum_{i=1}^{m} \left(|g(s^{(i)}.ts) - s^{(i)}.am|\right), as\_f = \langle s^{(1)}, ..., s^{(m)} \rangle$.

In our work, we take into account both the capacity limitations of the grid and the market actor's objective. Thus, we consider both the distance to the constraint and the target function to evaluate our results. In particular, we take into account the sum of the distances (target and constraint) and we use weights (coefficients) to prioritize the constraint violation. Of course, when constraint is respected, only the distance to the target function is taken into account.

Definition 7. We define the *distance of an assignment* as_f to a target function g and a constraint function c, $D_{g,c}(as\_f)$, as the weighted sum of its target and constraint distances with weights α and β respectively, i.e., $D_{g,c}(as\_f) = \alpha \cdot D_g(as\_f) + \beta \cdot D_c(as\_f)$, $\alpha, \beta \in \mathbb{R}$.

As mentioned in Section II, since an FO f captures a set of assignments ($L(f)$), there is at least one assignment of f that has the smallest distance.

Definition 8. We define the *target_to_constraint distance of a FO* f to a target function g and a constraint function c, $D_{g,c}(f)$, as the minimum distance among all its assignments to g and c, i.e., $D_{g,c}(f) = \min_{as\_f \in L(f)} D_{g,c}(as\_f)$.

Example 3. For instance, given $\alpha = 1$, $\beta = 10$, $c(t) = 2$, and $g(t) = 3$, an assignment of f_{12}^a in Figure 3 with the minimum distance is: $as\_f_{12}^a = [1, 3]$ where $D_{g,c}(as\_f_{12}^a) = 1 \cdot 0 + 10 \cdot 1 = 10 = D_{g,c}(f_{12}^a)$. On the contrary, an assignment of f_{12}^b with the minimum distance is: $as\_f_{12}^b = \langle [1,2], [1,2] \rangle$ where $D_{g,c}(as\_f_{12}^b) = 1 \cdot (1+1) + 10 \cdot 0 = 2 = D_{g,c}(f_{12}^b)$.

B. Aggregation techniques

We now present our 2 heuristic constraint-based FO aggregation techniques. Both the techniques are variations of the same abstract Greedy algorithm (Algorithm 1). They start by selecting (Line 2) the FO (f_{nom}) with the *maximum target_to_constraint distance* ($max_{f \in SF}(D_{g,c}(f))$). The reason is that apart from reducing the number of the AFOs, aggregation shall also produce FOs that are closer to the target in order to improve scheduling results. Thus, starting aggregation with FOs with high "distances" (used instead of target_constraint distance for simplification) is desirable and increases the chance of reducing the overall distance. Then, the selected FO, f_{nom}, is removed from the initial set (Algorithm 1, Line 2).

Algorithm 1 Abstract Greedy

Input: SF - set of FOs; g,c - a target and a constraint function
Output: SF - set of AFOs

```
1: $f_{tmp} \leftarrow$ null; $f_a \leftarrow$ null;
2: $f_{nom} \leftarrow$SelectNomFO($SF$); $SF \leftarrow SF \setminus f_{nom}$;
3: while $\exists f \in SF$ not aggregated do
4:   $\{f_a, f_{tmp}\} \leftarrow$BestAggregation($SF, f_{nom}$)
5:   if $D_{g,c}(f_a) < D_{g,c}(f_{nom})$ then
6:     $SF \leftarrow SF \setminus f_{tmp}$; $f_{nom} \leftarrow f_a$
7:   else
8:     AnnotateAsAFO($f_{nom}$)
9:     $SF \leftarrow SF \cup f_{nom}$
10:    $f_{nom} \leftarrow$ SelectNomFO($SF$); $SF \leftarrow SF \setminus f_{nom}$;
11: return $SF$
```

Algorithm 2 Simple Greedy extends Greedy (same input and output as Greedy)

```
1: function BestAggregation($SF, f_{nom}$)
2:   $f_{tmp} \leftarrow$ClosestToZeroDistance($SF$)
3:   $f_a \leftarrow$BinaryAggregation($f_{nom}, f_{tmp}$)
4:   return $\{f_a, f_{tmp}\}$
```

Afterwards, algorithm continues until all FOs are aggregated (Line 3). The two variations of Greedy examine different FOs to produce an AFO, i.e., f_a (Line 4). If there is an AFO (f_a) with smaller distance than f_{nom}, the algorithm continues aggregation with the aggregated one and removes f_{tmp} from the initial set SF (Line 6). Otherwise, it annotates f_{nom} as AFO and continues by selecting another f_{nom} from the non-aggregated ones (Lines 8–10). The algorithm stops when all the FOs are annotated as AFOs (Line 3) and returns set SF with the AFOs (Line 11).

Simple Greedy (SG). Apart from f_{nom}, SG also selects a single FO f_{tmp} to examine whether it will aggregate them or not (Algorithm 2, Line 2). In particular, it selects the FO (f_{tmp}) among the set that has the closest to zero distance to increase the chances of reducing the distance of f_{nom}. Then, in each step, it examines all the potential aggregations between the two FOs, i.e, f_{nom} and f_{tmp} to identify the AFO that reduces the distance of f_{nom} (Algorithm 2, Line 3).

Exhaustive Greedy (EG). EG explores a larger solution space than SG. In particular, during each step, it examines *all* the potential binary aggregations between f_{nom} and all the FOs in set SF (Algorithm 3, Line 3) compared to SG that examines only the binary aggregations among f_{nom} and one FO from SF. EG then stores the AFO with the smallest distance (Line 6). When the comparisons finish, it returns the AFO with the minimum distance (f_a) and the FO (f_{tmp}) that participated in the production of f_a (Line 7).

Algorithm 3 Exhaustive Greedy extends Greedy

```
1: function BestAggregation($SF, f_{nom}$)
2:   $f_a \leftarrow f_{nom}$; $f_{tmp} \leftarrow null$;
3:   for each $f \in SF$ do
4:     $f_y \leftarrow$BinaryAggregation($f_{nom}, f$)
5:     if $D_{g,c}(f_y) < D_{g,c}(f_a)$ then
6:       $f_a \leftarrow f_y$; $f_{tmp} \leftarrow f$;
7:   return $\{f_a, f_{tmp}\}$
```

Algorithm 4 Best binary aggregation function

Input: f_{nom}, f_{tmp} - FOs
Output: f_a - an AFO

```
1: function BinaryAggregation($f_{nom}, f_{tmp}$)
2:   $f_a \leftarrow f_{nom}$
3:   for each alignment $al$ of $\{f_{nom,}, f_{tmp}\}$ do
4:     $f_x \leftarrow$AGG-2-to-1($f_{nom}, f_{tmp}, al$)
5:     if $D_{g,c}(f_x) < D_{g,c}(f_a)$ then
6:       $f_a \leftarrow f_x$
7:   return $f_a$
```

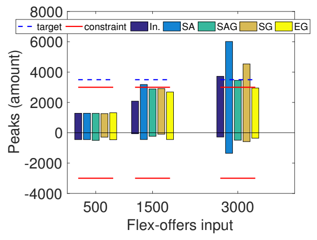
Constraint allocation feature. Since aggregation should lead to a valid schedule, it is desirable to examine, after each step, whether the node constraint is respected or not. However, this would require to schedule during each step the current FOs/AFOs, i.e., solve the UC problem. Due to the fact that the UC problem is an NP-complete problem [12], our aggregation algorithms instead act preventively in terms of constraint handling. In particular, it is possible for both algorithms to consider a constraint value lower than the original one. For instance, we typically allocate the constraint to 50% of its original value. As a result, the allocation feature obstructs aggregation to violate the constraint. Consequently, in cases where more than one AFOs have slice amounts closer to the constraint, it increases the chance for scheduling to form a node value that respects the constraint.

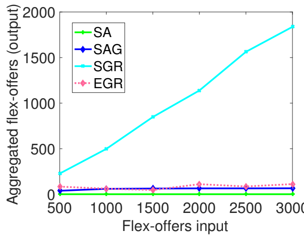
V. Experimental evaluation

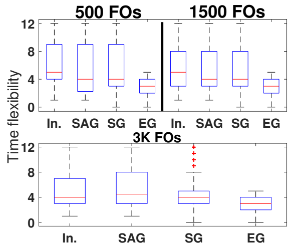
A. Experimental setup

We experimentally evaluate the proposed techniques in complex congestion scenarios. Our experiments are based on power characteristics from real loads (e.g., [13], [14]) that show similar use behavior and are complemented with potential flexibility, e.g., [15]. One amount unit corresponds to 0.5kW. The grid power capacity constraint used in the experiments represents medium voltage grids, e.g., [16]. We use a mixed portfolio of FOs that represents a variety of devices and characteristics regarding flexibility and power demand/supply. In particular, we generate 6 datasets of FOs with different sizes to be able to examine the scalability of the techniques in terms of input. The sizes of the datasets follow an arithmetic progression with both initial term and common difference equal to 500 FOs. Thus, the last dataset has 3000 FOs. In order to create imbalances and congestion situations, the number of the negative FOs is 10% of every dataset.

In particular, 40% of the positive FOs represent electrical vehicles (EVs), 30% represent heat pumps (HPs), and 30% clothes washers (CWs). The negative FOs represent wind turbines (WT) and photovoltaics (PV) that are less flexible


(a) Peaks


(b) #aggregated FOs


(c) Time flexibility box-plots

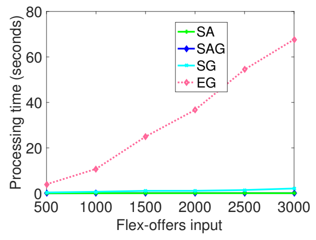

(d) Processing time

Fig. 4: $500-3$K FOs, target=3.5K, constraint=3K, constraint aggregation allocation $=1.5$K, $D_{g,c}(f) = 1 \cdot D_g + 10000 \cdot D_c$

Device	EST	tf	#slices	Min amounts	af
EV (day)	6	5	4	$U^*\{5,7\}$	$U\{0,2\}$
EV (night)	$N^*(18,1)$, $[17,20]$	$N(10,1)$, $[8,12]$	$U\{3,4\}$	$U\{5,7\}$	$U\{0,2\}$
CW (day)	$N(16,1)$, $[15,17]$	$U\{1,3\}$	$U\{2,4\}$	$U\{3,4\}$, $U\{1,2\}$	0
CW (night)	$N(20,1)$, $[19,21]$	$N(8,1)$, $[5,10]$	$U\{2,4\}$	$U\{3,4\}$, $U\{1,2\}$	0
HP (day)	$N(13,1)$, $[12,14]$	$N(3,1)$, $[1,5]$	$U\{4,7\}$	$U\{5,8\}$	$U\{0,2\}$
HP (night)	17	3	$U\{3,6\}$	$U\{5,8\}$	$U\{0,2\}$
WT, PV (day)	$N(14,1)$, $[13,15]$	$U\{3,4\}$	$U\{4,10\}$	$U\{8,10\}$	$U\{0,2\}$
WT, PV (night)	$N(23,1)$, $[22,24]$	$U\{1,4\}$	$U\{5,8\}$	$U\{8,10\}$	$U\{0,2\}$

TABLE I: Flex-offers characteristics, U^*: uniform distribution, N^*: Gaussian distribution

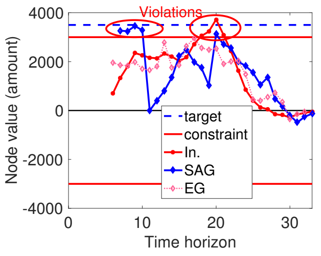


Fig. 5: 3K FOs input

with longer profiles. We allocate FOs during day time and night time for all the devices (50%-50%). Details about the characteristics of the datasets are shown in Table I.

Moreover, for comparison reasons we use two baseline aggregation techniques. We compare our techniques with Start Alignment (SA) aggregation [10] (see Section III-A where all the FOs are aggregated into one FO). We also use a *Start Alignment* with *Grouping* (SAG) aggregation technique where a grouping phase is used in advance [10]. Consequently, FOs with the same earliest start time and the same time flexibility are grouped together and SA is applied on each group. As a result, for each group of FOs, a single AFO is produced.

In order to examine whether an aggregation result allows the constraint to be respected or not, we implemented a stochastic scheduling technique based on the Evolutionary Algorithm (EA) proposed in [17]. EA is applied on a set of FOs (aggregated or not) and forms the node value with the possible minimum distance to target and constraint function. We see in Figure 5 how the node value is formed when EA is applied on the results of each aggregation technique along with the constraint and the target function values.

The experiments were conducted on a 2.9 GHz Intel core i7 processor with two cores, physical memory of 8 GB, and MacOS. The techniques are implemented in Java 1.8.

B. Use case

We examine our techniques in a case where target is greater than the constraint so that a bottleneck appears. Target is 3500 and constraint is 3000 (e.g., 6.6kV - 11kV substation [16]). We also use the constraint allocation feature. Thus, the constraint value used by the aggregation techniques is 1500. We set the target coefficient to 1 and we use a very high value for the constraint coefficient (10000) when the distance is computed, in order to prioritize the constraint respect.

In Figure 4a, we see how the highest and the lowest node values (amount peaks) are formed when EA is applied on the initial ("In." label) non-aggregated set and on the aggregation result of each technique. For a better illustration we show the cases where the input is 500, 1500 and 3000 FOs. We see in Figure 4a that when the input size is 500, the peaks formed by EA (scheduling) are quite far away from the constraint and as the input size is increased, the peaks approach and finally exceed the constraint. In the same figure, we also observe that when the input size is small (500), all the techniques lead to scheduling that respects the constraint. When the size is increased to 1.5K FOs, SA violates the constraint, and in the last case of 3K FOs only EG respects the constraint.

Regarding the number of the AFOs, we see that SA produces only one AFO in all the input cases (Figure 4b) and its time flexibility is always 1, i.e., the minimum among all the FOs. We also notice that SAG, due to the grouping that applies, produces a low number of AFOs and also achieves a similar to the initial time flexibility distribution among the AFOs, see Figure 4c. However, the aggregation result of SAG violates the constraint when the input is increased to 2.5K and 3K (shown in Figure 4a). Furthermore, in Figure 5, we see how the node value is formed based on EA when the aggregation input is 3K. In the same figure, we notice that SAG not only violates the constraint, but there is also violation in 5 out of the 28 scheduling points (approximately in the 18% of the time horizon).

Regarding SG, the number of AFOs scales linearly with the input size and achieves a time flexibility higher than EG. However, EG is the *only algorithm* that forms a node value that *respects* the constraint **in all the input cases.** It maintains the number of AFOs low (93% input reduction on average) and uses time flexibility to lead to a schedule that respects the constraint. That is why it has the lowest average time flexibility and a distribution with low boundaries, see Figure 4c.

Regarding the processing time, EG is the slowest algorithm due to the high number of comparisons it requires. It shows a similar to linear growth rate behavior, see Figure 4d. SG is fast since it only compares just two FOs in every step. Similarly, SA and SAG are the fastest algorithms due to the very low number of aggregations they perform.

Experimental summary. We observe that SG is fast and respects the constraint while SA does not. When size increases ($>$2K FOs), both SG and SAG violate the constraint. On the other hand, EG examines a larger solution space and leads to results that respect the constraint when the input size is large, see Figure 5, case of 3K FOs. It is indicative that even when we apply EA on the initial set of 3K FOs for 10 minutes, it still cannot provide a result that respects the constraint, see Figure 5 label "In.". On the contrary, EG uses approximately 67 seconds for its execution and EA applied afterwards produces the first result that respects the constraint in approximately two seconds. That means that EG is able to provide filtered inputs to scheduling so that initially unsolvable cases can be solved. However, when the input is large, EG requires high processing times. It requires 24.08 minutes to process a dataset of 10K FOs.

VI. Related Work

The role of an aggregator that handles flexible loads has been investigated in many previous works, e.g., [18], [19]. Such works use highly complex models and focus on controlling and scheduling methods. Their main characteristic is that the aggregator operates as an aggregated load controller that tries to follow a power reference and eventually tackles the scheduling problem to offer DR and ancillary services, e.g., [20]. On the contrary, in our work we use a low complexity generic model to represent energy flexibilities, namely flex-offers (FOs). Moreover, the main goal of our techniques is to produce flexible and non-scheduled AFOs that can be traded as commodities in emerging energy flexibility markets. Thus, our proposed techniques, SG and EG, produce AFOs that can lead to normal grid operation and use a generic target function that can capture overall business case scenarios.

Furthermore, there is an extensive literature tackling the unit commitment (UC) problem (scheduling), e.g., [21], [22]. In [17] the aggregation of FOs before scheduling showed an improvement of scheduling results compared to applying scheduling individually. Our work can be also applied in advance of scheduling process and not only reduces the complexity of the UC problem, but in addition, partially handles scheduling goals as it "filters" invalid results and improves their quality.

VII. Conclusion and future work

This paper introduces constraint-based aggregation over a generic data model that captures flexibilities in time and amount dimensions. It proposes two techniques that take into account the power capacity constraint limitations imposed by the grid. Moreover, the paper evaluates the proposed techniques in complex congestion scenario. The experimental evaluation shows that the proposed techniques can efficiently aggregate FOs and at the same time enable scheduling to respect the grid constraints, unlike existing techniques.

In our future work, we will focus on enhancing our techniques by automating the setting of aggregation parameters through sampling techniques. Moreover, we will extend our proposed algorithms to investigate the financial perspective of constraint-based aggregation on the future energy market.

Acknowledgment

This work was supported in part by the TotalFlex project sponsored by the ForskEL program of Energinet.dk.

References

[1] M. Boehm, L. Dannecker, A. Doms, E. Dovgan, B. Filipič, U. Fischer, W. Lehner, T. B. Pedersen, Y. Pitarch, L. Šikšnys, and T. Tušar, "Data management in the mirabel smart grid system," in *EnDM*, 2012.

[2] M. Albano, L. L. Ferreira, L. M. Pinho, and A. R. Alkhawaja, "Message-oriented middleware for smart grids," *Computer Standards & Interfaces*, vol. 38, 2015.

[3] I. Diaz De Cerio Mendaza, I. Szczesny, J. Pillai, and B. Bak-Jensen, "Demand response control in low voltage grids for technical and commercial aggregation services," *Smart Grid*, 2015.

[4] L. Ferreira, L. Siksnys, P. Pedersen, P. Stluka, C. Chrysoulas, T. le Guilly, M. Albano, A. Skou, C. Teixeira, and T. Pedersen, "Arrowhead compliant virtual market of energy," in *ETFA*, 2014, pp. 1–8.

[5] B. Neupane, T. B. Pedersen, and B. Thiesson, "Evaluating the value of flexibility in energy regulation markets," in *e-Energy*. ACM, 2015.

[6] B. Biegel, M. Westenholz, L. H. Hansen, J. Stoustrup, P. Andersen, and S. Harbo, "Integration of flexible consumers in the ancillary service markets," *Energy*, vol. 67, 2014.

[7] N. Padhy, "Unit commitment-a bibliographical survey," *Power Systems*, vol. 19, 2004.

[8] R. Mitra, V. Arya, B. Sullivan, R. Mueller, H. Storey, and G. Labut, "Using analytics to minimize errors in the connectivity model of a power distribution network," in *e-Energy*. ACM, 2015.

[9] D.-J. Won and S.-I. Moon, "Optimal number and locations of power quality monitors considering system topology," *Power Delivery*, vol. 23, 2008.

[10] L. Siksnys, E. Valsomatzis, K. Hose, and T. Pedersen, "Aggregating and disaggregating flexibility objects," *TKDE*, vol. 27, 2015.

[11] M. Aigner, "A characterization of the bell numbers," *Discrete Mathematics*, 1999.

[12] X. Guan, Q. Zhai, and A. Papalexopoulos, "Optimization based methods for unit commitment: Lagrangian relaxation versus general mixed integer programming," in *Power Engineering Society General Meeting*, 2003.

[13] J. Acosta, K. Combe, S. Djokic, and I. Hernando-Gil, "Performance assessment of micro and small-scale wind turbines in urban areas," *Systems Journal*, vol. 6, 2012.

[14] M. van der Kam and W. van Sark, "Smart charging of electric vehicles with photovoltaic power and vehicle-to-grid technology in a microgrid; a case study," *Applied Energy*, vol. 152, 2015.

[15] I. Sajjad, G. Chicco, and R. Napoli, "Demand flexibility time intervals for aggregate residential load patterns," in *PowerTech*, 2015.

[16] W. P. Distribution, "Generation capacity register," https://www.westernpower.co.uk/Connections/Generation/Generation-Capacity-Map/Generation-capacity-register.aspx, accessed: 2016-05-05.

[17] T. Tušar, L. Šikšnys, T. B. Pedersen, E. Dovgan, and B. Filipič, "Using aggregation to improve the scheduling of flexible energy offers," in *BIOMA*, 2012.

[18] X. Geng and P. Khargonekar, "Electric vehicles as flexible loads: Algorithms to optimize aggregate behavior," in *SmartGridComm*, 2012.

[19] H. Hao, B. Sanandaji, K. Poolla, and T. Vincent, "Aggregate flexibility of thermostatically controlled loads," *Power Systems*, vol. 30, 2015.

[20] H. Cai, A. Hutter, E. Olivero, P. Roduit, and P. Ferrez, "Load shifting for tertiary control power provision," in *POWERENG*, 2015.

[21] T. Logenthiran, D. Srinivasan, and A. M. Khambadkone, "Multi-agent system for energy resource scheduling of integrated microgrids in a distributed system," *Electric Power Systems Research*, vol. 81, 2011.

[22] T. Logenthiran, D. Srinivasan, A. Khambadkone, and H. N. Aung, "Multiagent system for real-time operation of a microgrid in real-time digital simulator," *Smart Grid*, vol. 3, 2012.