A Taxonomy for Modeling Flexibility and a Computationally Efficient Algorithm for Dispatch in Smart Grids

M. K. Petersen, K. Edlund, L. H. Hansen, J. Bendtsen and J. Stoustrup

***Abstract*— The word flexibility is central to Smart Grid literature, but to this day a formal definition of flexibility is still pending. This paper present a taxonomy for modeling flexibility in Smart Grids, denoted *Buckets*, *Batteries and Bakeries*.**

We consider a direct control Virtual Power Plant (VPP), which is given the task of servicing a portfolio of flexible consumers by use of a fluctuating power supply. Based on the developed taxonomy we first prove that no *causal* optimal dispatch strategies exist for the considered problem. We then present two heuristic algorithms for solving the balancing task: *Predictive Balancing* and *Agile Balancing*.

***Predictive Balancing*, is a traditional moving horizon algorithm, where power is dispatched based on perfect predictions of the power supply. *Agile Balancing*, on the other hand, is strictly non-predictive. It is, however, explicitly designed to exploit the heterogeneity of the flexible consumers.**

Simulation results show that in spite of being non-predictive *Agile Balancing* can actually out-perform *Predictive Balancing* even when *Predictive Balancing* has perfect prediction over a relatively long horizon. This is due to the flexibility-synergy-effects, which *Agile Balancing* generates. As a further advantage it is demonstrated, that *Agile Balancing* is extremely computationally efficient since it is based on sorting rather than linear programming.

I. Introduction

The introduction of renewable energy production into the existing power system is complicated by the inherent variability of production technologies, which harvest energy mainly from wind and sun. This means that it becomes increasingly challenging to maintain the real-time balance between production and consumption as the ratio of renewable energy production increases. In a Smart Grid system, on the other hand, the inherent flexibility of consumers, such as electric vehicles, heat pumps and HVAC-systems, may be mobilized to play an active part in solving the balancing task.

The flexibility of a given system is a unique, innate, state- and time dependent quality. In conversation it is therefore sometimes said that *flexibility is the ability to deviate from the plan*. That characterization of flexibility is very insightful, but it still leaves us with the problem of defining both *the ability to deviate* and *the plan*.

In this paper we focus on *the ability to deviate* by proposing a taxonomy for modeling flexibility. The numerous constraints that characterize a given flexible system were first investigated in [19]; in the present paper, however, we have chosen to focus on the constraints of

I) Power Capacity,
II) Energy Capacity,
III) Energy level at a specific deadline, and
IV) Minimum runtime,

since these are widely found in practical systems.

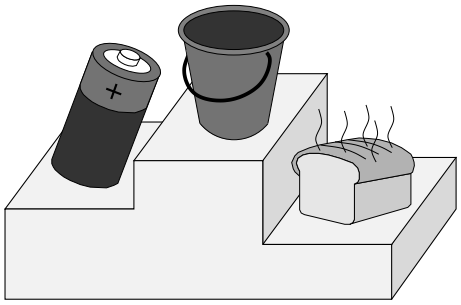

Fig. 1: *Buckets*, *Batteries and Bakeries* is a taxonomy for modeling flexibility in Smart Grids.

Our taxonomy is denoted *Buckets*, *Batteries and Bakeries* and precise definitions are given in Section IV. *The Bucket*, *The Battery* and *The Bakery* are three simple flexibility models, which are constructed based on the constraints I) to IV). The first model, denoted *the Bucket*, is a power and energy constrained integrator. The *Bucket* could be used as a simplified model of a house with a heat pump, which is used for energy storage. The *Battery* is also a power and energy constrained integrator, but with the added restriction that the unit must be fully charged at a specific deadline. The *Battery* could be modeling an electric vehicle, which must be ready for operation at a specific time. Finally the *Bakery* extends the *Battery* with the additional constraint that the process must run in one continuous stretch at constant power consumption. The *Bakery* could be a commercial green house, where plants must recieve a specific amount of light each day. This light must, however, be delivered continuously to stimulate the photosynthesis of the plants.

The suggested framework is a proper taxonomy in the sense that we have imposed a hierarchical relationship between the three models. This means that *a Bucket* provides a better quality of flexibility than *a Battery*, which is again superior to *a Bakery* (see Figure 1). Here, better quality means less restricted, not necessarily more flexible. The reason for this distinction is that the flexibility of a system is not just determined by constraints, but also by the specific parameter values of the system. That is, a "large" *Battery*

Authors M. K. Petersen, J. Bendtsen and J. Stoustrup are with the Department of Electronic Systems, Automation and Control, Aalborg University, Denmark; Mrs. M. K. Petersen is also affiliated with DONG Energy, Denmark as is K. Edlund and L.H. Hansen. Email: {mehpe, kried, larha}@dongenergy.com, {dimon, jakob}@es.aau.dk.
For more information on the PhD-project see Mettematics.com.

could therefore be said to be more flexible than a "small" *Bucket*, even though the *Bucket* is a better quality flexibility than the *Battery*.

Based on the hierarchical relationship between models we will develop an algorithm, *Agile Balancing*, which exploits the heterogeneity of flexible systems. This makes *Agile Balancing* robust against prediction errors and computationally efficient at the same time.

The paper is structured as follows: First, Section II gives an extensive review of how flexibility is modeled in Smart Grid literature today. Next, Section III and IV present the considered optimization problem and the taxonomy. Following this, it is proved formally in Section V how *causality* [16] relates to the taxonomy. Finally, Section VI and VII present *Predictive Balancing* and *Agile Balancing* and give comparative simulation examples.

II. State-of-the-Art

A review of how flexibility is modeled in Smart Grid literature reveals that the generic models of *Buckets*, *Batteries* and *Bakeries* are certainly not novel concepts. Several works have been identified (see Table I), which model flexibility in ways very similar to *a Bucket*, *a Battery* or *a Bakery*. Most existing literature, however, focuses on optimized operation of one particular technology. This means that the advantages of heterogeneity are not investigated.

In [9] a modeling framework for demand response technologies is formulated based on Markov Chain processes. This framework has some similarity to the taxonomy suggested in the present work. The authors of [9] subscribe to the concept of price-signalling, however; possible synergies between heterogenous subsystems are therefore not investigated, since these can only really be exploited though direct control.

The work closest related to the concepts investigated in this paper, is [16]; in fact, the term *laxity*, as used in [16], is almost synonymous with the term *agility* used in [5]. Only the *Battery*-model is investigated [16], however.

In our literature review we have also charted the use of the assumption of perfect prediction[1], which is found to be quite widespread.

III. Problem Formulation

Consider a Virtual Power Plant, which must provide power to a portfolio of flexible systems by dispatching a fluctuating power supply. The fluctuating power supply is denoted $P_{Dispatch}(k), k = 1, 2, \ldots, K$, and the flexible systems are denoted local units. A portfolio of N local units is denoted $\{LU_i\}_{i=1,2,\ldots,N}$. At sample k we let $P_i(k)$ denote the power, which is dispatched to unit i, and any quantity, which cannot be dispatched to the portfolio, is denoted $\mathcal{S}(k)$. The objective is to minimize the residual power, that is $|\mathcal{S}|$.

[1]Paper [13] does assume perfect prediction as indicated in Table I, but the effects of uncertainty are also investigated.

The problem can be formulated as

$$\min_{P_i(\cdot)} \sum_{k=0}^{\infty} |\mathcal{S}(k)| \tag{1}$$

$$s.t.$$

$$P_{Dispatch}(k) \in \mathbb{R}, k = 0, 1, ..., \infty \tag{2}$$

$$\sum_{i=1}^{N} P_i(k) + \mathcal{S}(k) = P_{Dispatch}(k) \tag{3}$$

and also subject to the dynamics and constraints of $\{LU_i\}_{i=1,2,\ldots,N}$.

IV. Taxonomy: Buckets, Batteries and Bakeries.

This section defines the *Buckets*, *Batteries and Bakeries-taxonomy* for modeling flexibility in Smart Grids.

Formal definitions of a *Bucket*, a *Battery* and a *Bakery* are given in Definition 1, 2 and 3 respectively, and the models are further illustrated in Figure 2, 3 and 4. In the following T_s denotes the size of the time step, $\underline{P}_i$ and $\overline{P}_i$ denote limits on consumption rate, $\underline{E}_i$ and $\overline{E}_i$ denote limits on energy storage levels and $v_i(k)$ is a boolean-valued variable stating whether or not a *Bakery* is running at sample k.

Definition 1 (Bucket):
The dynamics and constraints of a *Bucket* are

$$\begin{aligned} \textit{Bucket}_i(k)\text{: } & E_i(k+1) = E_i(k) + T_s P_i(k) \\ & \underline{P}_i \le P_i(k) \le \overline{P}_i \\ & \underline{E}_i \le E_i(k) \le \overline{E}_i \\ & E_i(0) = E_{i,0}, \end{aligned}$$

where $k = 0, 1, \ldots, \infty$, $i = 1, 2, \ldots, N^{\textit{Buckets}}$, $\underline{P}_i \le 0 \le \overline{P}_i$ and $\underline{E}_i \le E_{i,0} \le \overline{E}_i$.

Definition 2 (Battery):
The dynamics and constraints of a *Battery* are

$$\begin{aligned} \textit{Battery}_i(k)\text{: } & E_i(k+1) = E_i(k) + T_s P_i(k) \\ & 0 \le P_i(k) \le \overline{P}_i \\ & 0 \le E_i(k) \le \overline{E}_i \\ & E_i(0) = E_{i,0}, \\ & E_i(T_{end,i}) = \overline{E}_i, \end{aligned}$$

where $k = 0, 1, \ldots, \infty$, $i = 1, 2, \ldots, N^{\textit{Batteries}}$, $T_{end,i} \in \mathbb{N}$, $0 \le \overline{P}_i$ and $0 \le \overline{E}_i$.

Definition 3 (Bakery):
The dynamics and constraints of a *Bakery* are

$$\begin{aligned} \textit{Bakery}_i(k)\text{: } & E_i(k+1) = E_i(k) + T_s P_i(k), \\ & P_i(k) = \overline{P}_i v_i(k) \\ & 0 \le E_i(k) \le \overline{E}_i, \\ & E_i(0) = E_{i,0}, \\ & E_i(T_{end,i}) = \overline{E}_i, \\ & 0 \le \sum_{l=k}^{k+T_{run,i}-1} v_i(l) - T_{run,i}\Big(v_i(k) - v_i(k-1)\Big), \end{aligned}$$

Reference	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]
Bucket	x	x	x	x	x	x	x	x	(x)									
Battery						(x)	x		(x)	x	x	x	x	x	x	x	x	
Bakery								(x)	(x)							x	x	x
Perfect Prediction	Yes	Yes	No	Yes	No	No	Yes	No	No	Yes	Yes	Yes	Yes[1]	No	Yes	No	Yes	Yes

TABLE I: Review of flexibility modeling in Smart Grid literature.

where $k = 0, 1, \ldots, \infty$, $0 \leq \overline{P}_i$, $\overline{E}_i = \overline{P}_i T_{run,i}$, $v_i(k) \in \{0, 1\}$, $i = 1, 2, \ldots, N^{Bakeries}$, $T_{end,i} \in \mathbb{N}$ and $T_{run,i} \in \mathbb{N}$.

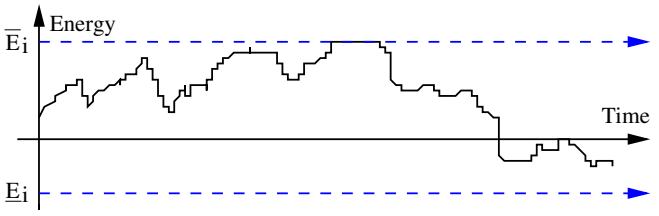

Fig. 2: A *Bucket* is a power and energy constrained integrator.

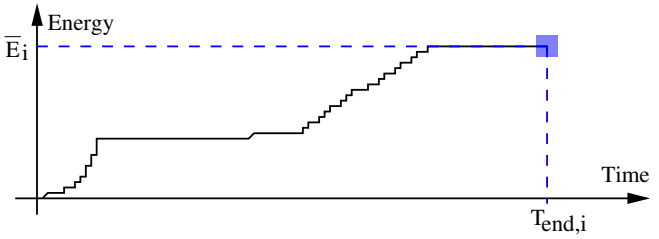

Fig. 3: A *Battery* is a power and energy constrained integrator, which must be "charged" to level $\overline{E}_i$ by time $T_{end,i}$.

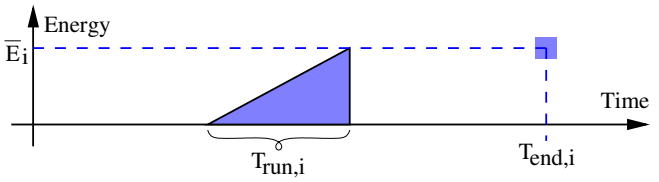

Fig. 4: A *Bakery* is a batch process, which must be finished by time $T_{end,i}$. The process has constant power consumption and the run time is $T_{run,i}$.

V. Causality

In [16] a dispatch strategy was defined as *causal* if it depends only on the information state at time k. The authors of [16] also proved that an optimal *causal* dispatch strategy does not exist for a portfolio of *Batteries*. It was shown in [5] that adding the constraint $\underline{P} = \underline{E} = 0$ for a portfolio of *Buckets* induces that an optimal *causal* dispatch strategy *does* exist. For the sake of completion this section will prove that an optimal *causal* dispatch strategy does not, in general exist for a portfolio consisting of only *Buckets* or only *Bakeries*.

Proposition 1: There does not exist an optimal *causal* dispatch strategy for a portfolio of *Buckets*.

Proof: Proof is done by counterexample. Consider a portfolio consisting of the following two *Buckets*

$$\begin{aligned} \textit{Bucket}_1\text{: } & E_1(0) = 0, \\ & \overline{P}_1 = 1, \overline{E}_1 = 1, \\ & \underline{P}_1 = -1, \underline{E}_1 = -1, \end{aligned}$$

$$\begin{aligned} \textit{Bucket}_2\text{: } & E_2(0) = 0, \\ & \overline{P}_2 = 1, \overline{E}_2 = 3, \\ & \underline{P}_2 = -1, \underline{E}_2 = -3, \end{aligned}$$

Next define the following dispatch profiles

$$\begin{aligned} P^A_{Dispatch} &= (0, 2, 2), \\ P^B_{Dispatch} &= (0, -2, -2). \end{aligned}$$

Observe that it is possible to dispatch sequence $P^A_{Dispatch}$ in such a way that $\sum_{k=0}^{2} |S| = 0$. However, this is only achievable if $P_1(0) = -1$ and $P_2(0) = 1$. Observe also that equivalent arguments hold for $P^B_{Dispatch}$ if $P_1(0) = 1$ and $P_2(0) = -1$. At $k = 0$ a *causal* dispatch strategy must offer allocations based only on information available at time $k = 0$. Notice, however, that $P^A_{Dispatch}(0) = P^B_{Dispatch}(0)$ and since optimal dispatch of $P^A_{Dispatch}$ and $P^B_{Dispatch}$ requires different allocations at time $k = 0$, a *causal* dispatch strategy cannot exist. ■

Proposition 2: There does not exist an optimal *causal* dispatch strategy for a portfolio of *Bakeries*.

Proof: Proof is done by counterexample. Consider a portfolio consisting of the following two *Bakeries*

$$\begin{aligned} \textit{Bakery}_1\text{: } & E_1(0) = 0, \\ & \overline{P}_1 = 1, \overline{E}_1 = 1, \\ & T_{run,1} = 1, T_{end,1} = 2, \\ \textit{Bakery}_2\text{: } & E_2(0) = 0, \\ & \overline{P}_2 = 3, \overline{E}_2 = 3, \\ & T_{run,2} = 1, T_{end,2} = 2. \end{aligned}$$

Next define the following dispatch profiles

$$\begin{aligned} P^A_{Dispatch} &= (2, 1), \\ P^B_{Dispatch} &= (2, 3). \end{aligned}$$

Observe that the optimal dispatch of either sequence $P^A_{Dispatch}$ or sequence $P^B_{Dispatch}$ to the portfolio has $\sum_{k=0}^{1} |S| = 1$. However, for $P^A_{Dispatch}$, this is only achievable if $P_1(0) = 0$ and $P_2(0) = 3$. For $P^B_{Dispatch}$ the

required configuration is $P_1(0) = 1$ and $P_2(0) = 0$. The argumentation that a *causal* optimal dispatch strategy does not exist now follows as in the proof of Proposition 1. ■

VI. Algorithms

Since we have proven that *causal* optimal dispatch strategies do not exist, this section will present two heuristic algorithms for solving problem (1) - (3). The algorithms are denoted *Predictive Balancing* and *Agile Balancing*.

A. Predictive Balancing

A strategy for solving problem (1) - (3) is to use a moving horizon approach. To do this, we assume perfect prediction of $P_{Dispatch}$ over a certain prediction horizon K, and solve

$$\min_{P_i(\cdot)} \sum_{k=1}^{K} w_k |\mathcal{S}(k)| \tag{4}$$

$$s.t.$$

$$P_{Dispatch}(k) \in \mathbb{R}, \tag{5}$$

$$\sum_{i=1}^{N} P_i(k) + \mathcal{S}(k) = P_{Dispatch}(k), \tag{6}$$

where $w_{k_1} > w_{k_2}$ if $k_1 < k_2$. Adding the impatience weights w_k to the cost function ensures that if the problem cannot be solved without introducing slack, then the imbalances will incur as late within the prediction horizon as possible.

B. Agile Balancing

The main objective of the present paper is to investigate heterogenous systems and we do this by introducing agility factors for each class of flexibility. The agility factor of a given unit should express the quality (see [19]) of the flexibility, which the unit represents.

The authors of the present paper first investigated the agility attributes of the *Bucket*-model in [5]. Here agility factors for the *Bucket*-model were defined as

Definition 4 (Agility Factor, Bucket):
Let *Bucket*$_i(k)$ denote a *Bucket*. The *agility factor* of *Bucket* i at sample k is

$$\mathcal{K}_i^{\textit{Bucket}}(k) = \frac{\overline{E}_i - E_i(k)}{T_s \overline{P}_i}.$$

With this definition of the agility factor for the *Bucket*-model we obtain that $\mathcal{K}_i^{\textit{Bucket}}(k)$ denotes the number of samples that the *Bucket* can operate at maximum power without becoming inactive/full.

Introducing *Batteries* and *Bakeries* to the portfolio means that in addition to balancing $P_{Dispatch}$ the Virtual Power Plant must solve a set of fixed tasks, namely charging the *Batteries* and starting the *Bakeries* in due time. This means that as a deadline, T_{end}, approaches, a *Battery* or a *Bakery* can go from being a flexible resource, which can help to minimize our objective, to being a constraint. We therefore define agility factors for the *Battery*- and *Bakery* models, which state how close we are (in terms of samples) to being forced to charge a battery or start bakery:

Definition 5 (Agility Factor, Battery):
Let *Battery*$_i(k)$ denote a *Battery*. The *agility factor* of *Battery* i at sample k is

$$\mathcal{K}_i^{\textit{Battery}}(k) = T_{end,i} - k - \frac{\overline{E}_i - E_i(k)}{T_s \overline{P}_i}.$$

Definition 6 (Agility Factor, Bakery):
Let *Bakery*$_i(k)$ denote a *Bakery*. The *agility factor* of *Bakery* i at sample k is

$$\mathcal{K}_i^{\textit{Bakery}}(k) = T_{end,i} - T_{run,i} - k.$$

Notice that the definition of agility factors for the *Battery* is the same as the definition of a flexibility factors used in [16].

As the deadline of a *Battery* or a *Bakery* approaches the Virtual Power Plant can be forced to charge that *Battery* or start that *Bakery* irrespective of whether this is beneficial to its objective. Forced consumption on LU_i at sample k can, however, be computed based on the agility factors, as

$$P_{Forced,i}^{\textit{Battery}}(k) = \begin{cases} 0 & \mathcal{K}_i^{\textit{Battery}} > 1 \\ \overline{P}_i(1 - \mathcal{K}_i^{\textit{Battery}}) & 1 \geq \mathcal{K}_i^{\textit{Battery}} > 0 \\ \overline{P}_i & \mathcal{K}_i^{\textit{Battery}} = 0 \end{cases}$$

and

$$P_{Forced,i}^{\textit{Bakery}}(k) = \begin{cases} 0 & \mathcal{K}_i^{\textit{Bakery}} > 1 \\ \overline{P}_i & \mathcal{K}_i^{\textit{Bakery}} = 0. \end{cases}$$

The algorithm *Agile Balancing* is based on the principle of *flexibility maximization* [19], where the worst quality units are dispatched first at each sample. The idea is simple: At each sample the Virtual Power Plant will first focus on the set assignments of charging *Batteries* and starting *Bakeries* . The Virtual Power Plant will solve the most pressing task first and the unit with the smallest agility factor is the most critical asset in need of service. At sample k *Agile Balancing* therefore dispatches as much power as possible to the *Batteries* and *Bakeries*, but no more than $P_{Dispatch}(k)$. Secondly, *Agile Balancing* uses the buffer available in the *Buckets* to minimize any remaining imbalance.

Since there are no energy requirements on a *Bucket*, it can only constitute a resource and never a constraint. There are both power and energy constraints on a *Bucket*, however, meaning that only a limited amount of power can be dispatched to the *Bucket*-portion of the portfolio at each sample. The maximum amount of power, which can be dispatched to *Bucket*$_i$ at sample k is denoted $P_{Reserve,i}^{\textit{Bucket}}(k)$ and is given as

$$P_{Reserve,i}^{\textit{Bucket}}(k) = \min\left(\overline{P}_i, \frac{\overline{E}_i - E_i(k)}{T_s}\right).$$

At sample k the upper reserve bound on a portfolio containing $N^{\textit{Buckets}}$ *Buckets* is therefore

$$P_{Reserve}^{\textit{Bucket}}(k) = \sum_{i=1}^{N^{\textit{Buckets}}} \min\left(\overline{P}_i, \frac{\overline{E}_i - E_i(k)}{T_s}\right).$$

Furthermore, *Agile Balancing* handles any dispatch to *Buckets* by implementing the linear cost function given in [5]. Pseudo-code for *Agile Balancing* is given in Algorithm 1.

Algorithm 1 :
Agile Balancing $\Big(\{\boldsymbol{LU}_i\}_{i=1,2,\ldots,N}, \boldsymbol{P_{Dispatch}}\Big)$

for $k = 1$ **to** K **do**

  Compute $P_{Forced}(k) =$

    $\sum_{i=1}^{N^{Batteries}} P^{Batteries}_{Forced,i}(k) + \sum_{j=1}^{N^{Bakeries}} P^{Bakeries}_{Forced,j}(k)$.

  if $P_{Forced}(k) > P_{Dispatch}(k)$ **then**

    $P^{Batteries}(k) = P^{Batteries}_{Forced}(k)$,

    $P^{Bakeries}(k) = P^{Bakeries}_{Forced}(k)$.

  else

    Sort *Batteries* and *Bakeries* according to increasing agility factor.

    Distribute $P_{Dispatch}(k)$ to *Batteries* and *Bakeries* in increasing agility factor order and such that $P^{Batteries}(k) + P^{Bakeries}(k)$ is as large as possible, but less than or equal to $P_{Dispatch}(k)$.

  end if

  Define $P^{Buckets}(k) = \min\Big(P^{Buckets}_{Reserve}(k),$

    $P_{Dispatch}(k) - P^{Batteries}(k) - P^{Bakeries}(k)\Big)$.

  Distribute $P^{Buckets}(k)$ to the *Buckets* as prescribed in [5] that is in decreasing agility factor order.

  Set $S(k) = P_{Dispatch}(k)$

    $-P^{Buckets}(k) - P^{Batteries}(k) - P^{Bakeries}(k)$.

end for

VII. Simulation Examples

This section presents two simulation examples. The first simulation example compares the performance of *Predictive Balancing* and *Agile Balancing*. The second simulation example investigates the computational efficiency of *Agile Balancing*. In all simulations we have $T_s = 1$ and $E_{i,0} = 0$ for all units. Solutions of problem (4) - (6) are computed by use of CPlex, [20]. *Agile Balancing* has been implemented in C#. Computations are performed on a standard laptop.

A. Predictive Balancing vs. Agile Balancing

This simulation example considers a randomly generated portfolio of 105 units, where $N^{Buckets} = 5$ and $N^{Batteries} = N^{Bakeries} = 50$. All units have $\frac{\overline{E}}{T_s \overline{P}} \leq 10$ and $\sum_{Portfolio} \overline{E} = 50$.

The results of running *Predictive Balancing* for $K = 10$ are given in Figure 5. When there is a drop in $P_{Dispatch}$ *Predictive Balancing* attempts to use the *Buckets* as buffer to maintain the balance between supply and demand. Towards the end of each low-period, however, *Predictive Balancing* is forced to use significant slack. This occurs because the prediction horizon is not sufficiently long, and the problem could be mended by increasing the prediction horizon. However, such a modification comes at the price of computation time, which we will explore later in this section.

The results of running *Agile Balancing* are presented in Figure 6. When there is a drop in the power supply *Agile Balancing* is poorly prepared and therefore has too many *Bakeries* started. Since the *Bakeries* cannot be shut down *Agile Balancing* must utilize the buffer in the *Buckets* to maintain the balance. With the given portfolio *Agile Balancing* is able to balance supply and demand without introducing slack until the very end of the simulation.

Computation times and the sum of the absolute value of the slack variable are given in Table II for $K = 10$, $K = 15$ and $K = 20$. Notice that *Predictive Balancing* must have perfect prediction of at least 20 samples to perform better than *Agile Balancing*. As the prediction horizon increases, so does the computation time of *Predictive Balancing*, however; notice that even with a prediction horizon of only 10 samples, *Predictive Balancing* is almost one hundred times slower than *Agile Balancing*. This is because the most computationally demanding task *Agile Balancing* must solve is to sort units according to agility factor. *Predictive Balancing*, on the other hand, solves a series of mixed integer programs, which is far more computationally demanding.

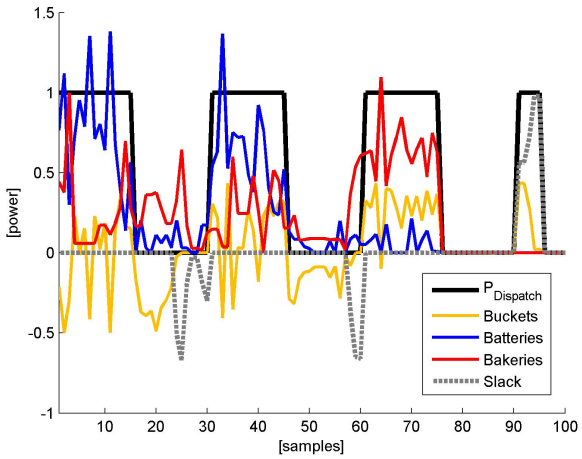

Fig. 5: Power dispatched at each sample for each type of unit by *Predictive Balancing* when $K = 10$.

	Comp. Time [s]	$\sum \lvert\mathcal{S}(\cdot)\rvert$
Agile Balancing	0.03	2.48
Predictive Balancing, K = 10	2.5	7.40
Predictive Balancing, K = 15	4.0	4.29
Predictive Balancing, K = 20	5.8	1.92

TABLE II: Computation time and the sum of numerical imbalances for *Predictive Balancing* and *Agile Balancing*.

B. Large Scale Simulations

This simulation example further investigates the computational efficiency of *Agile Balancing* by considering a

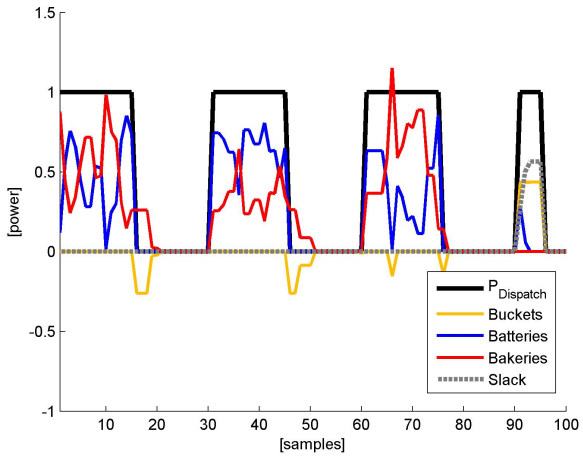

Fig. 6: Power dispatched at each sample for each type of unit by *Agile Balancing*.

Dyn. Ag.	*Buckets*	*Batteries*	*Bakeries*	Comp. Time	$\sum \lvert\mathcal{S}(\cdot)\rvert$
Yes	33%	33%	33%	3 min. 26 sec.	0
Yes	10%	45%	45%	3 min. 25 sec.	19712
No	33%	33%	33%	1 min. 1 sec.	0
No	10%	45%	45%	1 min. 4 sec.	43264

TABLE III: Computation time and the sum of numerical imbalances for large scale simulation.

randomly generated portfolio of 10^6 units. All units have $\frac{\overline{E}}{T_s\overline{P}} \leq 30$.

Figure 7 depicts the simulation results, when one third of each type of unit is included in the portfolio and in Figure 8 only 10% *Buckets* are included in the portfolio. Computation times and the sum of the absolute value of imbalances are given in Table III. In Smart Grid discussions it is often proposed that if only the number of units under the jurisdiction of a Virtual Power Plant is large enough, then *the-law-of-big-numbers* will ensure that the aggregated behavior of the portfolio will be the same as that of a traditional power plant (so essentially proposing that a large portfolio will exhibit *Bucket*-behavior). However, the second simulation (Figure 8) is an example of a case where a large number of units is not in itself enough to warrant that the load can be balanced. This illustrates that care must be taken the ensure that the right combination of units is available in the portfolio.

To further improve the computation time *Agile Balancing* has also been implemented without using dynamic agility factors. This means modifying Algorithm 1 by moving line 8 to the very start of the algorithm (before the for-loop), such that only one sorting is performed. The results of these simulations are given in Figure 9, Figure 10 and Table III. As expected, sorting only once per simulation gives a significant speed up of the computation time, as the modified implementation is more than three times faster than the original. With a portfolio of one third of each type of units, there is no cost of this speed up in terms of performance/optimality. With only 10% *Buckets* in the portfolio, however, not having dynamic agility factors has a significant cost in terms of performance.

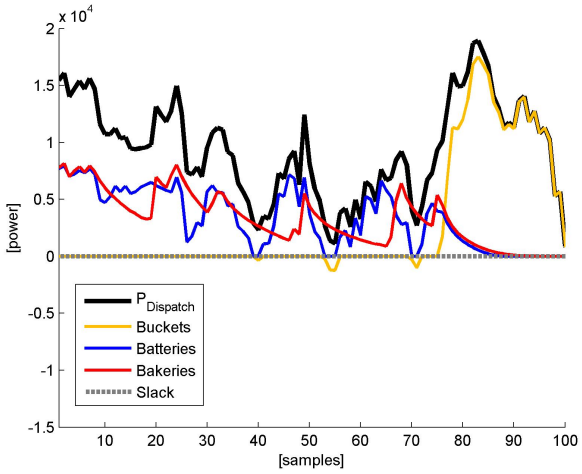

Fig. 7: Power dispatched at each sample for each type of unit by *Agile Balancing* for a portfolio of 1.000.000 units having one third of each type.

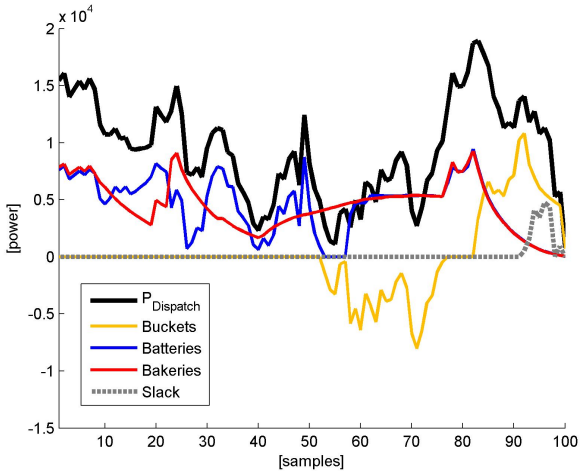

Fig. 8: Power dispatched at each sample for each type of unit by *Agile Balancing* for a portfolio of 1.000.000 units with 10% *Buckets*, 45% *Batteries* and 45% *Bakeries*.

VIII. Conclusion

In this paper we have identified a number of common traits shared by most, if not all, power consuming or -producing units that can be expected to appear in a future Smart Grid system. Most literature to date has focused on only one type of units or one particular technology, although some references have treated more than one type. We proposed a taxonomy that allows the division of units into three distinct categories based on key traits of the unit's primary purpose such as minimum runtime, the ability to consume/release power back to the grid, minimum consumption by a certain time, etc., in a quantifiable manner.

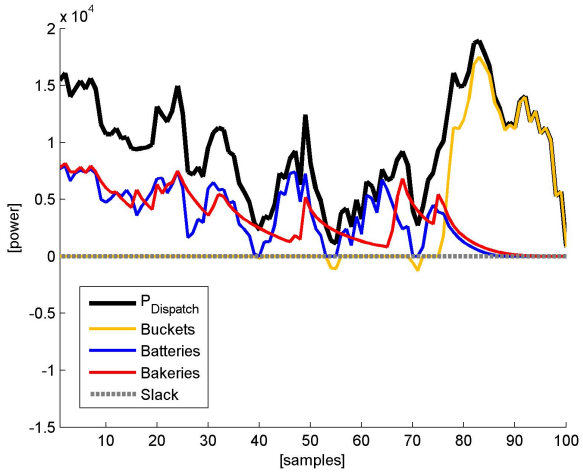

Fig. 9: Power dispatched at each sample for each type of unit by *Agile Balancing* for a portfolio of 1.000.000 units having one third of each type and not using dynamic agility factors.

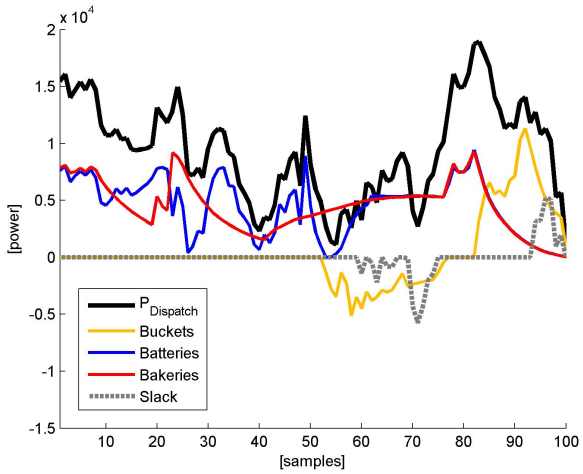

Fig. 10: Power dispatched at each sample for each type of unit by *Agile Balancing* for a portfolio of 1.000.000 units with 10% *Buckets*, 45% *Batteries* and 45% *Bakeries* and not using dynamic agility factors.

We have also presented a suboptimal, but extremely computationally efficient dispatch algorithm, denoted *Agile Balancing*. One of the main challenges in developing the Smart Grid is the sheer size of optimization problems involved. This means that the computation time associated with determining optimal solutions might be unacceptable in practice. An optimal solution available two minutes after market gate closure is far less useful than a suboptimal one available two minutes before market gate closure; thus, even though *Agile Balancing* is not optimal, it might still be the best solution in practice.

REFERENCES

[1] Kai Heussen, Stephan Koch, Andreas Ulbig, Göran Andersson, *Energy Storage in Power System Operation: The Power Nodes Modeling Framework*, IEEE PES Conference on Innovative Smart Grid Technologies Europe, 2010, pp. 1-8.

[2] Benjamin Biegel, Jakob Stoustrup, Jan Bendtsen and Palle Andersen, *Model Predictive Control for Power Flows in Networks with Limited Capacity*, 2012 American Control Conference, 2012, pp. 2959-2964.

[3] Ali Faghih, Mardavij Roozbehani and Munther A. Dahleh, *Optimal Utilization of Storage and the Induced Price Elasticity of Demand in the Presence of Ramp Constraints*, 50th IEEE Conference on Decision and Control and European Control Conference, 2011, pp. 842-847.

[4] T.Y. Lee and N. Chen, *Effect of the Battery Energy Storage System on the Time Of Use Rates Industrial Customers*, IEE Proc.-Gener. Transm. Distrib., Vol 141, No. 5, September 1994.

[5] Mette Petersen, Jan Dimon Bendtsen and Jakob Stoustrup, *Optimal Dispatch Strategy for the Agile Virtual Power Plant*, 2012 American Control Conference, 2012, pp. 288-294.

[6] Matt Kraning, Yang Wang, Ekine Akuiyibo, Stephen Boyd, *Operation and Configuration of a Storage Portfolio via Convex Optimization*, 18th IFAC World Congress, 2011, pp. 10487-10492.

[7] Nikolaos Gatsis and Georgios B. Giannakis, *Residential Demand Response with Interruptible Tasks: Duality and Algorithms*, 50th IEEE Conference on Decision and Control and European Control Conference, 2011, pp. 1-6.

[8] Ioannis Ch. Paschalidis, Binbin Li, Michael C. Caramanis, *A Market-Based Mechanism for Providing Demand-Side Regulation Service Reserves*, 50th IEEE Conference on Decision and Control and European Control Conference, 2011, pp. 1-6., pp. 21-26.

[9] Konstantin Turitsyn, Scott Backhaus, Maxim Ananyev and Michael Chertkov, *Smart Finite State Devices: A Modeling Framework for Demand Response Technologies*, 50th IEEE Conference on Decision and Control and European Control Conference, 2011, pp. 7-14.

[10] B. Daryanian, R.E. Bohn and R.D. Tabors, *Optimal Demand-Side Response to Electricity Spot Proices for Storage-Type Customers*, IEEE Transactions on Power Systems, Vol. 4, No. 3, 1989.

[11] Amir-Hamed Mohsenian-Rad, Vincent W. S. Wong, Juri Jatskevich, Robert Schober and Alberto Leon-Garcia, *Autonomous Demand-Side Management Based on Game-Theoretic Energy Consumption Scheduling for the Future Smart Grid*, IEEE Transactions on Smart Grid, Vol. 1, No. 3, 2010.

[12] Angel Rosso, Juan Ma, Daniel S. Kirschen and Luis F. Ochoa, *Assessing the Contribution of Demand Side Management to Power System Flexibility*, 50th IEEE Conference on Decision and Control and European Control Conference, 2011, pp. 4361-4365.

[13] Changsun Ahn, Chiao-Ting Li and Huei Peng, *Decentralized Charging Algorithm for Electrified Vehicles Connected to Smart Grid*, American Control Conference, 2011.

[14] Anthony Papavasiliou and Shmuel S. Oren, *Supplying Renewable Energy to Deferrable Loads: Algorithms and Economic Analysis*, IEEE Power and Energy Society General Meeting, 2010.

[15] Ralph Hermans, Mads Almassalkhi and Ian Hiskens, *Incentive-based Coordinated Charging Control of Plug-in Electric Vehicles at the Distribution-Transformer Level*, 2012 American Control Conference, 2012, pp. 264-269.

[16] A. Subramanianz, M. Garcia, A. Domnguez-Garca, D. Callaway, K. Poollay and P. Varaiyay, *Real-time Scheduling of Deferrable Electric Loads*, 2012 American Control Conference, 2012, pp. 3643-3650.

[17] Jing Huang, Vijay Gupta and Yih-Fang Huang, *Scheduling Algorithms for PHEV Charging in Shared Parking Lots*, 2012 American Control Conference, 2012, pp. 276-281.

[18] Kin Cheong Sou, James Weimer, Henrik Sandberg, and Karl Henrik Johansson, *Scheduling Smart Home Appliances Using Mixed Integer Linear Programming*, 50th IEEE Conference on Decision and Control and European Control Conference, 2011.

[19] Mette Petersen, Lars Henrik Hansen and Tommy Mølbak, *Exploring the Value of Flexibility: A Smart Grid Discussion*, 8th IFAC Conference on Power Plant and Power System Control, 2012.

[20] `www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/`