Industrial Demand-Side Flexibility: A Benchmark Data Set

Nicole Ludwig
Institute for Automation and Applied Informatics,
Karlsruhe Institute of Technology
Eggenstein-Leopoldshafen, Germany
nicole.ludwig@kit.edu

Lukas Barth
Institute of Theoretical Informatics,
Karlsruhe Institute of Technology
Karlsruhe, Germany
lukas.barth@kit.edu

Dorothea Wagner
Institute of Theoretical Informatics,
Karlsruhe Institute of Technology
Karlsruhe, Germany
dorothea.wagner@kit.edu

Veit Hagenmeyer
Institute for Automation and Applied Informatics,
Karlsruhe Institute of Technology
Eggenstein-Leopoldshafen, Germany
veit.hagenmeyer@kit.edu

ABSTRACT

To cope with the new demands of an electrical grid based on mostly renewable energy, more flexibility on the demand-side is needed. To test new demand-side management strategies, energy consumption data sets which come with some information about the inherent flexibility of the processes, are needed. However, such data sets are often commercially sensitive and thus not published or replaced with entirely artificial data. In the present paper, we introduce a new benchmark data set containing scheduling scenarios of industrial processes with flexibility information. The instances are based on a real-world data set of a small scale industrial facility, from which we extract process characteristics using a novel motif discovery technique. We provide an in-depth analysis of the benchmark data set and show that it is suitable to evaluate smart-grid scheduling techniques.

CCS CONCEPTS

• **Applied computing** → *Industry and manufacturing*; *Computer-aided manufacturing*; • **Mathematics of computing** → Combinatorial optimization; • **Computing methodologies** → *Motif discovery*; • **Information systems** → Data analytics;

KEYWORDS

flexibility, smart grid, project scheduling, demand side management, demand response, motif discovery

ACM Reference Format:
Nicole Ludwig, Lukas Barth, Dorothea Wagner, and Veit Hagenmeyer. 2019. Industrial Demand-Side Flexibility: A Benchmark Data Set. In *Proceedings of the Tenth ACM International Conference on FutureEnergy Systems (e-Energy '19), June 25–28, 2019, Phoenix, AZ, USA*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3307772.3331021

1 INTRODUCTION

Societies around the globe aim for future electricity grids which rely mostly on renewable energy sources (RES). Unfortunately, the intermittent nature of the RES and the fact that they cannot be controlled makes integrating them into todayâĂŹs grid difficult. One difficulty is that there is often a mismatch between the supply from RES and the actual power demand. While increasing transmission and storage capacities is one option to ease the integration, another one is generally referred to as demand-side management (DSM). DSM includes all measures which aim at changing behaviour in the energy usage by demand-side actors. DSM has been discussed extensively in the literature ([10],[5]). An important assumption underlying all DSM approaches is that the consumers have some kind of temporal or operational flexibility. Thus, they can either change *when* they use energy (temporal flexibility), or *how* they use energy (operational flexibility).

Information about the flexibility of individual consumers, especially industrial ones, is not readily available leading to difficulties in testing new strategies and ideas for DSM or making different strategies comparable with each other. To test different algorithms and frameworks, one can use data from or resembling smart meters (e. g. [11]) or grid data (e. g. [16]). However, regardless of the type of data, most authors either do not publish the data their analysis is based on ([1]) or synthesize the whole data set. The synthetic data can range from being entirely made up (e.g., [20]), being modelled with specific appliances in mind (e.g., [15]) or being generated based on data but without using algorithms to extract information from this data (e.g., [23]). Benchmark data sets play an essential role in making research comparable and more accessible. For general project scheduling, for example, there exists the PSBLIB benchmark data set from [14], which the related literature uses heavily (e. g. [22]). However, this benchmark data set is not rooted in real-world data. Specifically for resource-constrained project scheduling, [13] have published a data set. Again, the data set is not derived from real-world data. Moreover, no such benchmark data set exists for demand-side flexibility in industrial processes.

Recently, the HIPE data set, a real-world data set with smart meter measurements from industrial machines, has been published [4]. This data set contains power demand time series from a set of machines in a small-scale electronics factory. However, the data set consists only of a relatively small amount of machines and there is no readily available information about their flexibility. Hence, in the present paper, we extract demand information from the real-world data set of industrial machines, generate more process

This is a preprint version of https://dx.doi.org/10.1145/3307772.3331021.
e-Energy '19, June 25–28, 2019, Phoenix, AZ, USA

instances based on this information and infer flexibility attributes. More specifically, we use a novel algorithm based on motif discovery to find regular process patterns in each machine and extract information on when each process starts throughout the day as well as how many different processes can be identified in each machine. Based on this information we generate instances that model real-world scenarios with available demand-side flexibility. To the best of our knowledge, we are the first to create such artificial instances based on pattern recognition through motif discovery. We evaluate the found patterns and show that the data can be used to evaluate performance-critical scheduling algorithms on workloads that resemble real-world scenarios.

The remainder of the paper is structured as follows. We start with a short introduction to a scheduling problem which we use to evaluate the data set in Section 2. In Section 3 we describe the methodology to extract the process pattern from the power demand time series. We then describe how we generate instances based on this information in Section 4. Section 5.1 goes into detail about the origin of the time series data as well as the exact parameters chosen to generate our benchmark data set. It also explains how to obtain our data set. After that, we describe the respective processes found (Section 6), and evaluate their behaviour in scheduling algorithms (Section 7).

2 PRELIMINARIES

The main contribution of the data set described in Section 5 is a set of real-world-data based benchmark instances for certain scheduling problems. The problems at hand arise in smart grids when flexible electrical demands can be moved in time to optimize various objectives. In this section, we introduce such a scheduling problem, which we also use in Section 7.2 to evaluate the suitability of the benchmark instances. Additionally, we show in Section 2.2 how the assumptions of the problem in Section 2.1 can be relaxed.

2.1 Single-Resource Project Scheduling

In a first step, we define the problem under the assumption that all processes have constant power demand during their execution. In Section 2.2 we describe how the defined problem can be used to optimize scenarios where processes' power demand changes over time without the need for a more elaborate model.

In the scheduling problem, every time-moveable process constitutes one *job*. Let j_i be a job. Then, j_i has a *processing time* p_i, i. e., a time during which it must be executed without interruption. The job also has a *release time* r_i, which is the earliest time during which the job can execute, and a *deadline* d_i, which is the earliest time at which the job must be finished. Finally, every job has a *usage* u_i, which is the (constant) amount of power required by j_i during its execution.

Given a set of n such jobs $J = \{j_1, j_2, \ldots j_n\}$, a problem instance also has an edge-weighted directed acyclic graph $G = (J, D, w)$ on J, with edge set $D \subset J \times J$ and weight function $w : D \rightarrow \mathbb{Z}$. The edge set D defines *dependencies* between two jobs, while the weight function indicates the necessary *lag* between the two jobs. If $(j_a, j_b) \in D$, then j_b can start at the earliest $w((j_a, j_b))$ time steps after j_a has started.

Based on these definitions, we now define the problem used throughout this paper, which models a peak shaving scenario.

Problem 1 (Single-Resource Acquirement Cost Problem). *Given an instance as J and G as defined above, find a start time for each job such that the peak demand of the resulting schedule is minimized.*

Using the notation for project scheduling problems developed by Herroelen et al. [12], the S-RACP problem can be denoted as $1 \mid cpm, \rho_j, \delta_j \mid av$.

2.2 Non-Constant Power Demands

The problem defined in Section 2.1 assumes power demands to be constant over time. This assumption might be an especially unrealistic and simplifying one. Therefore, we describe in this section how the model from Section 2.1 can be used to model jobs with fluctuating power demand.

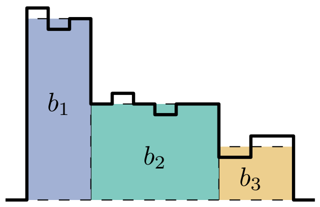


Figure 1: Decomposition of a stepwise power demand function into blocks. The fat black line is the original demand function with 11 steps. The three blocks b_1, b_2 and b_3 approximate this function.

The power demand function of a job is a stepwise function. To model a stepwise power demand function for a job, we perform a *block decomposition*, which is illustrated in Figure 1. For a perfect representation of a stepwise power demand function with k steps, we decompose the respective job into k blocks. From the perspective of the scheduling problem, each of the blocks again is an individual job. However, we will use the term *block* for a job with constant power demand that is part of a decomposition of a job with non-constant power demand. It is easy to see that the k blocks of a job, when executed consecutively, behave like a single job with the appropriate stepwise power demand function.

Thus, we must make sure that the blocks are executed consecutively and without any pauses between them. To this end, we use the dependencies introduced in Section 2.1. Let $b_1, b_2, \ldots b_k$ be the k blocks that we decomposed a job j_a into. Then, for every $i \in \{1, \ldots k-1\}$, we add (b_i, b_{i+1}) to D, with a weight $w((b_i, b_{i+1})) = p_i$. With this, no block can start before its predecessor has finished (but immediately after). Finally, we add (b_k, b_1) to D with $w((b_1, b_k)) = -1 \cdot \sum_{i=1}^{k-1} p_i$. This negative lag forces the last block to start *at the latest* $\sum_{i=1}^{k-1} p_i$ time steps after b_1 started. Combined, these dependencies cause the chain of $b_1, b_2, \ldots b_k$ to be executed concurrently.

When modelling a real-world process (with a stepwise power demand function) in this way, the obvious k to choose is the number of steps in the respective power demand function. As mentioned before, this would result in a perfect representation of the original stepwise power demand function. However, one can also choose a smaller number. If one chooses k smaller, the job's power demand function can not reflect the original process' power demand function perfectly, but only approximate it. In Section 7, we take a closer look at how well this approximation works with a low value for k, as well as how the value k influences the complexity of S-RACP instances.

3 FINDING PROCESS PATTERNS

It is a typical unsupervised learning problem to find patterns in time series. The most popular methods for finding these patterns include different forms of clustering. In the present paper, we use a variation of the motif discovery algorithm [7]. The advantage of motif discovery in contrast to other time series clustering approaches is that it uses a symbolic approximation (SAX) of the time series. This approximation allows for a more fuzzy matching of the compared sequences which is essential for us as we are not looking for exact pattern matches but rather patterns which are similar yet not the same. Additionally, motif discovery uses random projection [6] making it a lot faster than other clustering approaches; however with similar results [19].

Motif discovery relies on several parameters, most importantly are the size of the pieces in each sequence, the word size w which is compared and the size and distribution of the alphabet a for SAX.

In the original motif discovery algorithm, a predefined window of size w is moved over the time series to find the patterns. This moving window works excellent when looking for patterns of equal length and especially patterns which occur at regular intervals. However, in the case of the machine data we have at hand, the patterns are highly irregular both in their size and their occurrence over time. We thus need an initial step of finding the relevant sequences we want to compare. Similar to [18], we find those sequences with an event search. An event starts when the power values are greater than zero and ends when they are equal to zero again. As found in [2] some of the machines seem to have a standby mode or some minor issues with recording zero values and thus have a small offset. In the cases where such an offset is present, an event starts as soon as the power values are above this offset and ends as they are below this offset. Each time series from a machine now consists of several sequences which we assume to describe active states of the corresponding machine. To find the processes running on each machine, we want to compare these sequences with the help of an adapted motif discovery algorithm.

As stated before, following the approach in [7] we want to make a piecewise approximation of the time series and distribute an alphabet. The found sequences from the machines differ significantly in their length, thus to compare them easily later we approximate each sequence in such a way, that the length of each sequence from one machine time series is the same size w. The size of each piece in a sequence thus depends on the length of the sequence n and the predefined word size w. The word size is in our case specified as the median length of all sequence from a time series.

Given the sequences and their piecewise approximation we can now distribute the letters of the alphabet. Contrary to the original motif discovery algorithm [7], our alphabet distribution is based on the empirical cumulative distribution function (ECDF) of the time series, which is defined as follows. For observations $x = (x_1, x_2, \ldots, x_n)$, $F_n(p)$ is the fraction of observations less or equal to p, i. e.,

$$F_n(p) = \frac{1}{n} \sum_{i=1}^{n} \mathrm{I}(x_i \leq p).$$

The letters for each sequence are taken to be the percentiles of the ECDF of the current time series. We are thus limiting the alphabet to size $a = 10$. The ECDF is chosen as most of the machines do not follow a normal distribution even after normalization. The time series sequences are thus better characterized using the ECDF.

With the symbolic approximation of all time series sequences, we can then apply the random projection algorithm. We determine the number of iterations based on the length of the sequences, defining the number of iterations as 10% of the maximum length of sequences from one time series. The parameters which determine when a new motif candidate is found are left at their default values from the TSMining implementation in R [9]. The motif candidates found are then evaluated using the dynamic time warping distance.

The motifs and their occurrences are then real-valued time series sequences of different sizes. Each time series has at least one motif with at least two occurrences attributed to this motif.

As final step in the pattern finding process, we determine block decompositions for each detected occurrence, as necessary for the process to represent jobs with non-constant power demand introduced in Section 2.2. In Section 7, we specify a measure for what we consider a good block decomposition of an occurrence. For each occurrence, and each k, i. e., each number of blocks the occurrence is to be decomposed into, we use a sequential least-squares programming solver to find a decomposition into k blocks that optimizes the measure from Section 7. The block decompositions of all occurrences are part of the data set we release, see Section 5 for how to obtain them.

4 GENERATING S-RACP INSTANCES

After detecting motifs, their occurrences and their block decompositions, we generate jobs that form instances of the Single-Resource Acquirement Cost Problem (cf. Section 2.1). Several parameters influence the generation. First, the *instance size* specifies the number of jobs per instance. Also, a *time horizon* must be given, i.e., the latest deadline of any job, assuming that the earliest release time is 0. For the individual jobs, we first must specify the *block count*, i. e., into how many blocks a job should be decomposed. Also, we must specify how much flexibility we assume to be part of the instance, which is done in terms of a *window growth mean*, a *window growth standard deviation* and a *window base factor.*

Instance generation works by creating a set of *job generators*, one for each motif and each block count k, and then generating jobs from these generators up to the desired instance size. The idea behind the job generators is to fit random distributions to the respective motif's occurrences. When generating a job that should be divided into k blocks, we start with the respective motifs' occurrences that have been decomposed into k blocks as per Section 3. For each block, we obtain its length and total energy consumption, for a total of $2k$ values for each of the motif's occurrences. To these values, we fit a $2k$-variate Gaussian Mixture Model (GMM). We chose Gaussian Mixture Models because they are universal density approximators, as shown by e.g. Plataniotis and Hatzinakos [21], meaning they can approximate any given probability density with arbitrary precision (under the condition that the GMM has enough components). Since we do not want to make any assumptions about the underlying distribution of the duration and energy values, GMMs seem appropriate. Drawing from this distribution results in the shape of a new job: For each of the job's blocks, we get a duration and an energy consumption, from which we determine the power demand.

To determine a release time and a deadline, we first fit a distribution to the start times of the motif's occurrences. However, the start times do empirically not fit a (mixed) normal distribution well. Thus, we instead use a mixture of uniform distributions. To do so, we first cluster the start times using the DBSCAN algorithm [8] to account for the assumption that there might be multiple separate time spans throughout a day during which the respective process is usually started. Then, we determine the 0.1 and 0.9 quantile of each determined cluster to account for outliers. These form the lower and upper limit of one uniform distribution each. We weight each uniform distribution by the number of occurrences assigned to the respective cluster. Randomly selecting one such uniform distribution by their weight and then drawing from that distribution yields a preliminary start time s.

However, we need a release time r and and a deadline d (together forming the *window* of the job). We obtain them by determining a window size w and then setting $r = s - (w/2)$ and $d = s + p + (w/2)$ (with p being the processing time, cf. Section 2.1). We determine w in two components, the *window base* w_b and the *window growth* w_g. The window base is meant to reflect the flexibility we see in the real-world data. However, the factory we retrieved the real-world data from was so far not managed with demand-side management in mind, thus we assume that more flexibility could be created if operations were changed to facilitate DSM, which is why we add the window growth component.

The window base w_b is determined by the span of the uniform distribution we drew the start time from multiplied by the window base factor. We assume that the more flexible a process is, the larger the spans of its uniform start-time distributions will be. The window growth is determined by drawing from a normal distribution with the specified window growth mean and window growth standard distribution.

To generate an instance with n jobs, we n times perform a weighted selection on the set of job generators. We weight the generators by the number of occurrences in the respective motif. Each time, we generate one job using the selected generator. For each job generated in this way, the release times and deadlines produced by the job generator are based on time-of-day. Thus, we finally move each generated job to a random day within the time horizon. Note that although the S-RACP as defined in Section 2.1 allows for dependencies between jobs, we only use these dependencies for the block decomposition as described in Section 2.2. The real-world data we obtained (cf. Section 5.1) does not contain any satisfactory information about dependencies between processes (cf. Section 6), thus we do not incorporate these into the generated instance sets.

4.1 Grouped Generation

To evaluate the effect of k, i. e., the number of blocks into which jobs are decomposed, we generate groups of instances that differ only in the value of k. We do this by first generating an instance with $k = 1$. Then, to generate an instance with $k = 2$, we iterate over all jobs in the $k = 1$ instance. For each such job, we generate a $k = 2$ job from the same motif, scale the job so that the total duration and energy consumption is the same as for the $k = 1$ job, and set the same window. We proceed in the same way for all values for k.

5 THE BENCHMARK DATA SET

Based on the process to generate instances as described in Section 4, we now describe the real-world data in which the instance sets are based, the concrete instance sets we generated and explain how to obtain these instance sets and the accompanying auxiliary data.

5.1 Data Origin

As mentioned above, real-world data forms the basis for our generated instances. This real-world data comes from a data set of smart meter measurements in a small scale electronics factory and is called HIPE [4]. In the present paper, we use a subset in machines and superset in time of the originally published HIPE data set. The instances are generated based on 6 machines: a chip press, a high temperature oven, a screen printer, a soldering oven, a vacuum oven and a washing machine. We only use this subset of machines since for each of the other machines, the data quality for the selected time range (see below) was questionable for various reasons. All of the machines have been equipped with smart meters which record several quantities such as voltages, currents, frequencies etc. at a frequency of 50Hz. Out of a large number of measured quantities we only consider the active power. The first active power value we use is the last day of 2016 10 pm, while the last power value is from 31.12.2018 10:59 pm. We thus use two full years of data. We down-sample the data to one minute resolution, where the one minute values are the mean values from the original 50Hz measurements during that minute. Due to some problems during the recording of the measurements, not all machines have data for all minutes in the considered time period. For the machines with a complete set of power values we consider 1,051,260 minutes. For more information on the origin of the data and the machines we refer the interested reader to [4].

5.2 Data Set Parameters and Publication

The instance generation process from Section 4 requires several parameters to be set. Table 1 list the parameters we chose for the instance set we generated. If a table cell contains three values like (a, b, c), that means we chose all values from a to b (inclusive) in increments of c. Between all parameters where we choose more than one value, we take the cartesian product to obtain the final parameter space. The chosen parameter space results in a total of 1764 instances.

Table 1: Generated instance sets parameters. Three values (a, b, c) in a cell indicate that a range of parameters was chosen: from minimum a to maximum b (inclusive), with steps of size c.

Parameter Name	Chosen Values
Job Count	(200, 500, 15)
Window Growth Mean	(50, 200, 50)
Window Growth Std.Dev.	20
Window Base Factor	(0.05, 0.15, 0.05)
Time Horizon	5 days
Time Resolution	1 minute
Block Count	{1, 2, 3, 4, 5, 7, 10}

Table 2: Characteristics of the individual machines and the patterns found in the sequences. Where $\bar{E}$ is the average energy needed in a sequence per machine and $\bar{n}$ is the average length of a sequence per machine.

Machine	Sequences	Pattern	$\bar{E}(kW)$	$\bar{n}(min)$
High Temperature Oven	226	7	1.13	74.50
Screen Printer	173	1	0.32	285.51
Soldering Oven	206	4	1.89	146.32
Vacuum Oven	572	7	0.41	12.57
Washing Machine	66	4	1.95	188.47
Chip Press	51	2	1.09	433.63

We publish the instance set generated as above together with some auxiliary data as a separate data publication [17], accessible at https://doi.org/10.5445/IR/1000094324.

The data archive itself contains a detailed description of its contents and the file formats. The instance file format is suitable to be used with the TCPSPSuite software package[1], which is what we used for all optimizations performed for the evaluation. The auxiliary data includes a description of the motifs discovered (as described in Section 3) as well as for every instance the best solution we computed during our evaluation. These solutions can be used as a baseline for benchmarks.

6 EVALUATION: CHARACTERISTICS OF THE PATTERNS

In Section 3 we have shown how to extract the patterns from the machine time series before we generate a bigger set of instances. In this section, we want to briefly characterise the patterns we have found using the above described method on the HIPE data set. Table 2 summarises how many sequences and patterns were found for each machine as well as the average energy and time they needed. Dependencies among machines is often an argument against any flexibility in operation. We thus want to gather some information from the real machines that help us to determine whether they are dependent on each other. For this purpose, we examine the correlation between the start and end times of all machines, which could indicate a temporal dependency.

As can be seen in Figure 2, the time dependencies among the processes are all relatively low. Therefore, we assume that either all output can be stored efficiently in between operating dependent machines, or that the machines do not depend on each other much. Either way, we assume we can ignore dependencies for the moment, as already mentioned in Section 4.

[1]https://github.com/kit-algo/TCPSPSuite

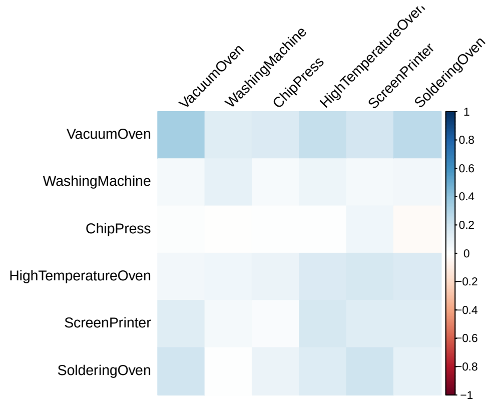


Figure 2: Pearson correlation between the end times (in minutes after midnight) and start times (in minutes after midnight) among all processes.

Although there does seem to be no dependency between the machines, we check for other dependencies such as the time of day a machine is operated. As shown in Table 3, most machines show a correlation between the start time of a process and the length of the process, with shorter processes starting later in the day than longer processes. The main reason for this seems to be the fact that working hours are roughly between 6 am and 6 pm. Thus, any machine or process which needs supervision or needs to have ended before the workers go home is not started late in the day.

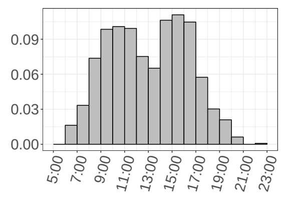
Table 3: Pearson correlation between the length of the processes and the time they are started for all machines.

Machine	ρ
Vacuum oven	-0.01
Washing machine	-0.15
Chip press	-0.42^{***}
High temperature oven	-0.17^{***}
Screen printer	-0.48^{***}
Soldering oven	-0.39^{***}

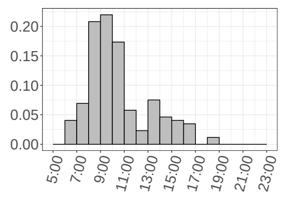
Stars indicating the significance level, with *** for $p < 0.01$.

The dependency on working hours seems to also be relevant during the lunch break. As we can see in Figure 3 there is a significant drop in starting times of processes during noon. In total, most processes get started during the peaks occurring before and after the lunch break. Most machines exhibit the pattern seen in Figure 3a, with the exception of the screen printer which is more often started before noon.

Overall, the starting times are spread over the whole working day and there seems to be only little time restriction on the processes other than when the workers have a break or go home.



(a) Hours during which all machines are started.



(b) Hours during which the screen printer is started.

Figure 3: Density of processes started during each hour of a day for all machines Figure 3a and the most unusual start time distribution found for the screen printer Figure 3b.

7 EVALUATION: BLOCK DECOMPOSITION GRANULARITY

Since the data has one-minute time resolution, the power demand curve for every occurrence detected in Section 3 is a stepwise function, with one step per minute. However, for many algorithmic approaches, for example scheduling algorithms, the run-time complexity increases significantly if the demand functions have too many steps. Many approaches even can not deal with non-constant demand, i.e., require a demand function with exactly one step.

For this reason, in Section 4 we generate jobs with varying, but low numbers of steps in their power demand function. We also call the number of steps in the demand function the number of *blocks* that we decompose a job into. In this section, we analyze the effects that the number of blocks of a job has. In Section 7.1, we evaluate how closely we can approximate the original power demand functions with various numbers of blocks. In Section 7.2, we look at the increase in optimization complexity that comes with a rising number of blocks, using a mixed-integer program for the S-RACP as an example. Finally, in Section 7.3, we look at how well solutions for instances with low k values work for the same instances with high k values.

7.1 Approximation of the Original Power Demand Curve

In Section 2.2 we describe a block decomposition that allows to approximate the (stepwise) power demand curve of some original process with a varying number of steps (resp. blocks).

In this section, we analyze how close a given power demand curve with a low number of steps can be to the original curve it tries to approximate. As original curves, we take the occurrences discovered during motif discovery. First, we need a distance measure between two stepwise functions. Given the stepwise demand function of an occurrence o as P_o, and a stepwise demand function with k steps $\tilde{P}_{o,k}$ that approximates P_o, we use the measure

$$\Delta\left(P_o, \tilde{P}_{o,k}\right) = \frac{1}{N_o} \int_0^\infty \left(P_o(t) - \tilde{P}_{o,k}(t)\right)^2 \mathrm{d}t. \qquad (1)$$

Here, N_o is a normalization factor determined as $N_o = \int_0^\infty P_o(t)\mathrm{d}t$, i.e., the total energy of the original occurrence. Intuitively, the distance between two stepwise functions should correlate with the area between the two curves. However, we assume that — especially for peak shaving applications — a large deviation over a short time is worse than a small deviation over a longer time, which is why we square the difference inside the integral.

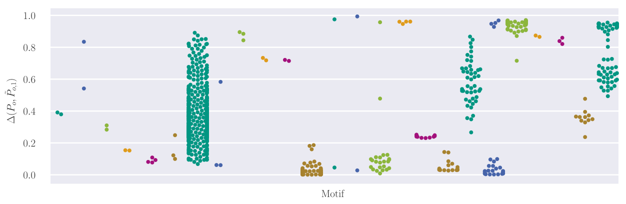


Figure 4: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 1$.

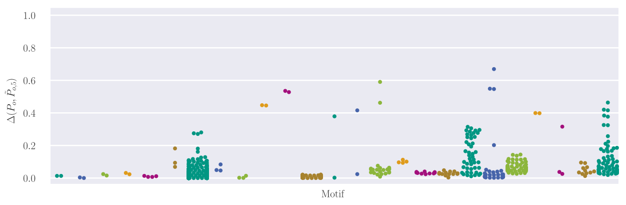


Figure 5: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 5$.

We now investigate how well the detected occurrences can be approximated with a low number of steps (resp. blocks) according to this measure. Note that this is very different from generating jobs as described in Section 4 and then computing (1) for each job. A job is never generated from a single occurrence, and therefore does not approximate any single occurrence's power demand function. Instead, for a given k, we compute for every occurrence of every motif a block decomposition that minimizes (1). If $\Delta(P_o, \tilde{P}_{o,k})$ becomes small for a given occurrence o and its optimal k-block approximation $\tilde{P}_{o,k}$, that means that occurrence o can be approximated well using only k blocks.

Figures 4, 5 and 6 show the Δ values for k in $\{1, 10\}$ and every occurrence. Larger plots, as well as plots for all k in 1 to 10 plus 15 and 20, can be found in the appendix in Section A. In the plots, every dot is one occurrence, which are arranged into columns by their motif. We see that there are some motifs the occurrences of which can be well approximated with only one block. However, for many motifs, one block is not enough for a good approximation. On the other hand, we can also see that the effects of using more than 5 blocks is small.

Figure 7 shows a line plot of the change in Δ for changing values of k (on the x axis). Here, for every occurrence o, and every $k \in \{1, \ldots 10\}$, we set the y value to $\Delta(P_o, \tilde{P}_{o,k}) - \Delta(P_o, \tilde{P}_{o,20})$, giving an insight into how much one can improve the approximation for that occurrence when changing k from its respective value to 20. Here, we again see that until about $k = 5$, there is a sharp drop in Δ values, with the curve being rather flat afterwards. Thus, we can conclude that for the processes we mined from the time series data, a block decomposition into 5 blocks might be a good compromise between accuracy and instance complexity.

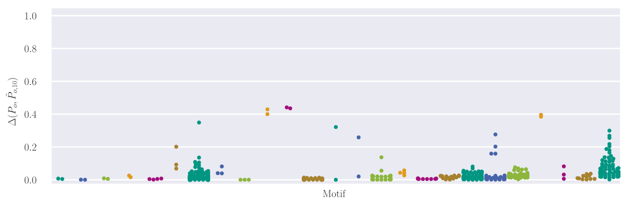


Figure 6: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 10$.

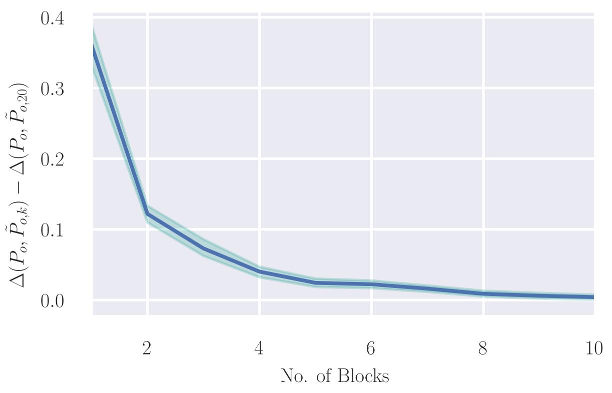


Figure 7: $\Delta(P_o, \tilde{P}_{o,k}) - \Delta(P_o, \tilde{P}_{o,20})$ for all occurrences, with k being on the horizontal axis. The solid blue line indicates the mean, and all lines for all occurrences fall within the green shaded area.

7.2 Scheduling Complexity

When choosing the number of blocks to decompose jobs into, one important consideration is the time complexity of the optimization problem one intends to solve. In this section, we explore how the complexity of optimizing S-RACP using a mixed-integer program changes with changing block decomposition.

To this end, we use the MIP formulation introduced by Barth et al. [3], which supports all necessary features for the block decomposition as described in Section 2.2, such as dependencies which negative time lags. We optimize every instance for 30 minutes using Gurobi 7.0 with 16 threads on a machine with 16 Intel® Xeon® E5-2670 CPUs and 64 GBs of RAM. All resulting models fit into the available RAM. Figure 8 shows how complexity changes with increasing number of blocks in the instance. Every dot is one optimized instance. The y axis indicates the MIP gap achieved after 30 minutes, the x axis reports the number of blocks in the instance. The color of a dot indicates how many blocks every job in the respective instance has been decomposed into.

We see that for the instances with one block per job, the solver usually achieves a MIP gap of at most 5%. With two blocks per job, the achieved MIP gaps increase significantly. For most instances, they go up to around 10%, however there is a sizeable fraction of instances that can only be optimized to around 40% MIP gap. Going to three blocks per job again worsens MIP gaps, however there seems to be no drastic further deterioration for four and five blocks per job. For seven and ten blocks per job, most instances can still be optimized to below 40% MIP gap, however we now have some instances with a MIP gap of 100%, i. e., for which the solver could not find any feasible solution.

These results indicate that if one is to choose at least three blocks per job, one might as well go with a finer decomposition, since it does not increase computational complexity significantly. However, it is already questionable whether expected MIP gaps around 40% are still acceptable. If not, one is restricted to coarser block decompositions.

7.3 Quality of Schedules with Few Blocks

Regarding the decision into how many blocks to decompose jobs, an obvious question is how well a solution for a low-block decomposition translates to a high-block decomposition of the same jobs. If the solutions translate well, one does not gain much by choosing a higher number of blocks, and since the number of blocks increases computational complexity (cf. Section 7.2), it would be advisable to chose a low number of blocks.

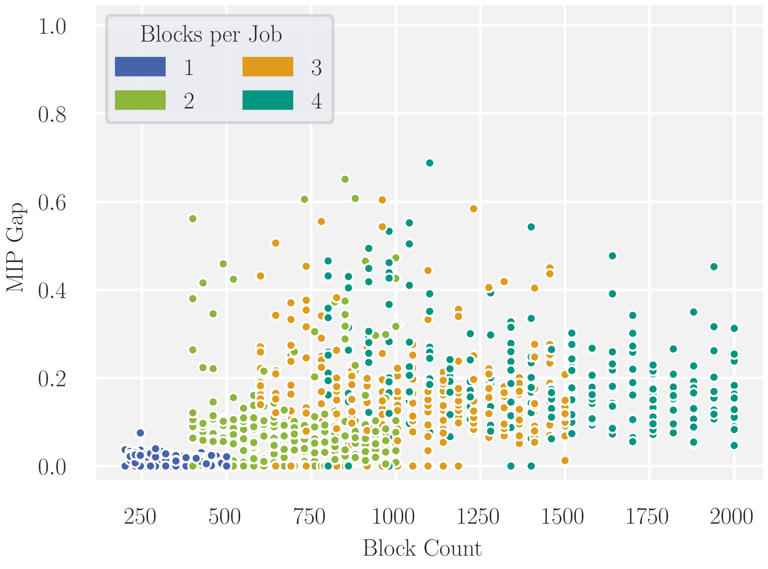
As described in Section 4.1, we generated instances that are suitable to evaluate this question: We generated groups of instances, within which the same jobs are decomposed into varying numbers of jobs. The maximum number of blocks we decomposed each job into is 10. We evaluate the quality of a low-block solution as follows: For each job in the low-block instance, determine its computed start time. Then, set that start time as the start time of the corresponding job in the high-block (with $k = 10$) instance. Doing this for all jobs in an instance leads to a new solution for the $k = 10$ instance. We determine the factor between the quality of the so constructed solution and the best solution computed for the $k = 10$ instance.

Figure 9 shows the results of this evaluation. Every dot is one instance. The x axis depicts the number of blocks that the jobs in the instances in the respective column were decomposed into. The y axis shows the quality of the solution transferred from $k = \{1, 2, 3, 4, 5, 7\}$ divided by the respective $k = 10$ instance. We see a downward trend. While for many $k = 1$ instances, the transfer results in a deterioration by up to 40%, for $k = 7$, the deterioration is mostly limited to 20%. There are some instances where the transferred solution is better than the solution computed on the $k = 10$ instance — this is likely because the $k = 10$ instance was harder to solve and could not be well optimized within the time limit.

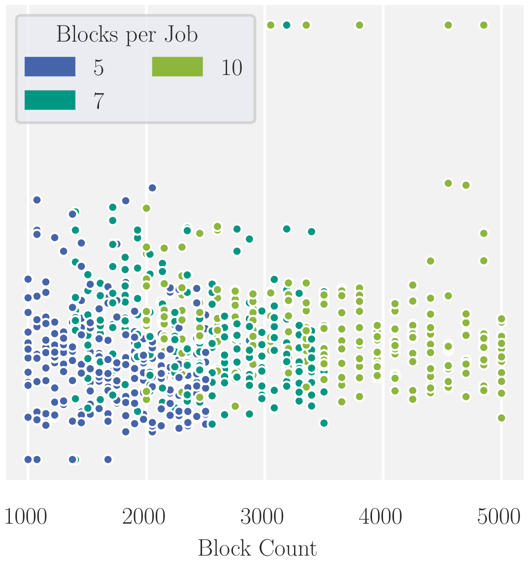
Because of these mixed results, we conclude that the approach of transferring a solution from one block decomposition to another itself has a stronger influence on the result than the number of blocks. Therefore, it must be decided on a case-by-case basis whether this approach is valid in practice.

8 CONCLUSION

In the present paper, we presented a new benchmark data set of industrial demand-side flexibility scheduling scenarios. The data set is based on real-world smart meter information from a small industrial facility and has been up-sampled to address large scale problems. The instances in the data set vary in terms of their size, the assumed amount of flexibility and the complexity of their processes' power demand functions, such that the data set covers a



(a) $k \leq 4$



(b) $k \geq 5$

Figure 8: MIP gaps achieved after 30 minutes of optimization versus number of blocks in the instance. Every dot represents one instance. Color denotes the number of blocks each job is decomposed into. For readability reasons split into block decompositions with less than 5 blocks (a) and at least 5 blocks (b). Where the gap is 1.0, the solver did not find any feasible solution within 30 minutes.

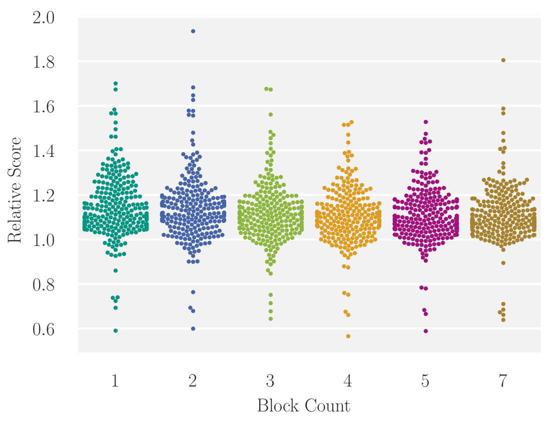


Figure 9: Every dot depicts an instance with k as per the x axis. The value on the y axis is the relation between the solution obtained by transferring the best solution of the respective instance to the corresponding instance with $k = 10$, and the best solution computed for the corresponding $k = 10$ instance.

wide range of conceivable scheduling problems. We evaluated how the precision of the approximation of the found patterns with different blocks influences the scheduling performance and find that the complexity does not increase much as the job is split in more than three blocks. Additionally, there is no straight forward answer to the question how good a schedule with few blocks is compared to a schedule with more blocks per job. One should decide the used block count on a case-by-case basis, based on the need for accuracy weighted against the need for performance. Please note that it is in no way necessary that all jobs are decomposed into the same number of blocks.

Overall, we believe that the benchmark data set can be used to evaluate scheduling techniques dealing with demand-side management. The instances are complex enough to provide a challenge, yet because of the large parameter space diverse enough to point out strengths and weaknesses in the algorithms to be evaluated. Our evaluation has shown that the block decomposition we perform is to a certain extend suitable to reflect the original power demand curves, thus we may assume that our instances resemble real-world scenarios.

In the future, it would be interesting to enrich the data set with additional constraints from real-world scenarios, such as dependencies between processes, storage constraints, etc.

ACKNOWLEDGMENTS

This work was supported by the German Research Foundation (DFG) as part of the Research Training Group GRK 2153: Energy Status Data — Informatics Methods for its Collection, Analysis and Exploitation. Thanks to Dominik Werle for helping us with the original data.

REFERENCES

[1] Sankar Ashok. 2006. Peak-load management in steel plants. *Applied Energy* 83, 5 (2006), 413–424.

[2] Lukas Barth, Veit Hagenmeyer, Nicole Ludwig, and Dorothea Wagner. 2018. How much demand side flexibility do we need? Analyzing where to exploit flexibility in industrial processes. In *Proceedings of the Ninth International Conference on Future Energy Systems - e-Energy '18*, Unknown (Ed.). ACM Press, New York, New York, USA, 43–62. https://doi.org/10.1145/3208903.3208909

[3] Lukas Barth, Nicole Ludwig, Esther Mengelkamp, and Philipp Staudt. 2018. A comprehensive modelling framework for demand side flexibility in smart grids. *Computer Science - Research and Development* 33, 13 (2018), 1865–2042. https://doi.org/10.1007/s00450-017-0343-x

[4] Simon Bischof, Holger Trittenbach, Michael Vollmer, Dominik Werle, Thomas Blank, and Klemens Böhm. 2018. HIPE – an Energy-Status-Data Set from Industrial Production. In *Proceedings of ACM e-Energy (e-Energy 2018)*. ACM, New York, NY, USA. https://doi.org/10.1145/3208903.3210278

[5] Nina Boogen, Souvik Datta, and Massimo Filippini. 2017. Demand-side management by electric utilities in Switzerland: Analyzing its impact on residential electricity demand. *Energy Economics* 64 (2017), 402–414. https://doi.org/10.1016/j.eneco.2017.04.006

[6] Jeremy Buhler and Martin Tompa. 2002. Finding motifs using random projections. *Journal of computational biology : a journal of computational molecular cell biology* 9, 2 (2002), 225–242. https://doi.org/10.1089/10665270252935430

[7] Bill Chiu, Eamonn Keogh, and Stefano Lonardi. 2003. Probabilistic discovery of time series motifs. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, Ted Senator (Ed.). ACM, New York, NY, 493–498. https://doi.org/10.1145/956750.956808

[8] Martin Ester, Hans-Peter Kriegel, JÃűrg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. AAAI Press, 226–231.

[9] Cheng Fan. 2015. *TSMining: Mining Univariate and Multivariate Motifs in Time-Series Data*. R package version 1.0.

[10] Paddy Finn and Colin Fitzpatrick. 2014. Demand side management of industrial electricity consumption: Promoting the use of renewable energy through real-time pricing. *Applied Energy* 113 (2014), 11–21. https://doi.org/10.1016/j.apenergy.2013.07.003

[11] Sebastian Gottwalt, Wolfgang Ketter, Carsten Block, John Collins, and Christof Weinhardt. 2011. Demand side management—A simulation of household behavior under variable prices. *Energy Policy* 39, 12 (2011), 8163–8174. https://doi.org/10.1016/j.enpol.2011.10.016

[12] Willy Herroelen, Erik Demeulemeester, and Bert De Reyck. 1999. *A Classification Scheme for Project Scheduling*. Springer US, Boston, MA, 1–26. https://doi.org/10.1007/978-1-4615-5533-9_1

[13] Rainer Kolisch, Christoph Schwindt, and Arno Sprecher. 1999. Benchmark Instances for Project Scheduling Problems. In *Project Scheduling*, Jan Węglarz (Ed.). Springer, Boston, MA, 197–212. https://doi.org/10.1007/978-1-4615-5533-9{_}9

[14] Rainer Kolisch and Arno Sprecher. 1997. PSPLIB - A project scheduling problem library. *European Journal of Operational Research* 96, 1 (1997), 205–216. https://doi.org/10.1016/S0377-2217(96)00170-1

[15] Ying Li, Boon Loong Ng, Mark Trayer, and Lingjia Liu. 2012. Automated Residential Demand Response: Algorithmic Implications of Pricing Models. *IEEE Transactions on Smart Grid* 3, 4 (2012), 1712–1721. https://doi.org/10.1109/TSG.2012.2218262

[16] Thillainathan Logenthiran, Dipti Srinivasan, and Tan Zong Shun. 2012. Demand Side Management in Smart Grid Using Heuristic Optimization. *IEEE Transactions on Smart Grid* 3, 3 (2012), 1244–1252. https://doi.org/10.1109/TSG.2012.2195686

[17] Nicole Ludwig, Lukas Barth, Dorothea Wagner, and Veit Hagenmeyer. 2019. Benchmark Dataset for "Industrial Demand-Side Flexibility: A Benchmark Data Set". KITOpen Repository. (2019). https://doi.org/10.5445/IR/1000094324

[18] Nicole Ludwig, Simon Waczowicz, Ralf Mikut, and Veit Hagenmeyer. 2017. Mining Flexibility Patterns in Energy Time Series from Industrial Processes. In *Proceedings. 27. Workshop Computational Intelligence, Dortmund, 23. - 24. November 2017*, Frank Hoffmann, E. Hüllermeier, and Ralf Mikut (Eds.). KIT Scientific Publishing, 13–32.

[19] Nicole Ludwig, Simon Waczowicz, Ralf Mikut, and Veit Hagenmeyer. 2018. Assessment of Unsupervised Standard Pattern Recognition Methods for Industrial Energy Time Series. In *Proceedings of the Ninth International Conference on Future Energy Systems - e-Energy '18*, Unknown (Ed.). ACM Press, New York, New York, USA, 434–435. https://doi.org/10.1145/3208903.3212051

[20] Mette K. Petersen, Lars H. Hansen, Jan Bendtsen, Kristian Edlund, and Jakob Stoustrup. 2014. Heuristic Optimization for the Discrete Virtual Power Plant Dispatch Problem. *IEEE Transactions on Smart Grid* 5, 6 (2014), 2910–2918. https://doi.org/10.1109/TSG.2014.2336261

[21] Kostantinos N. Plataniotis and Dimitris Hatzinakos. 2000. Gaussian mixtures and their applications to signal processing. In *Advanced Signal Processing Handbook: Theory and Implementation for Radar, Sonar, and Medical Imaging Real Time Systems*, Stergios Stergiopoulos (Ed.). CRC Press, Boca Raton, Chapter 3.

[22] Pilar Tormos and Antonio Lova. 2001. A Competitive Heuristic Solution Technique for Resource-Constrained Project Scheduling. *Annals of Operations Research* 102, 1/4 (2001), 65–81. https://doi.org/10.1023/A:1010997814183

[23] Sean Yaw, Brendan Mumey, Erin McDonald, and Jennifer Lemke. 2014. Peak demand scheduling in the Smart Grid. In *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 770–775. https://doi.org/10.1109/SmartGridComm.2014.7007741

A FULL FIGURES FOR SECTION 7.1

Here, we supply larger plots for the analysis performed in Section 7.1 and for all values for k in $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20\}$. Note that in rare cases, the $\Delta(P_o, \tilde{P}_{o,k})$ value slightly increases with increasing k for some occurrences. This is likely because the algorithm we used to optimize the block decomposition is not exact. We used 10^5 iterations of sequential least-squares programming.

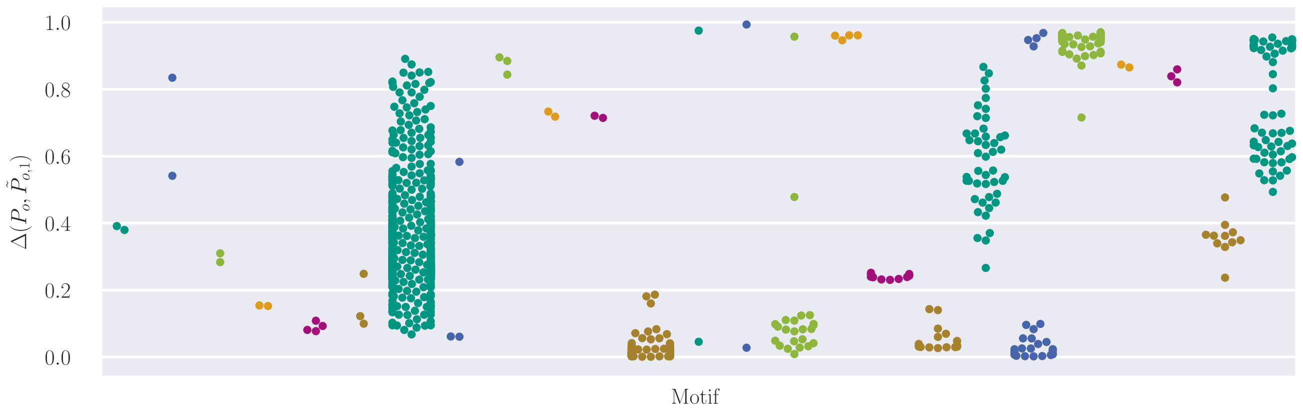


Figure 10: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 1$.

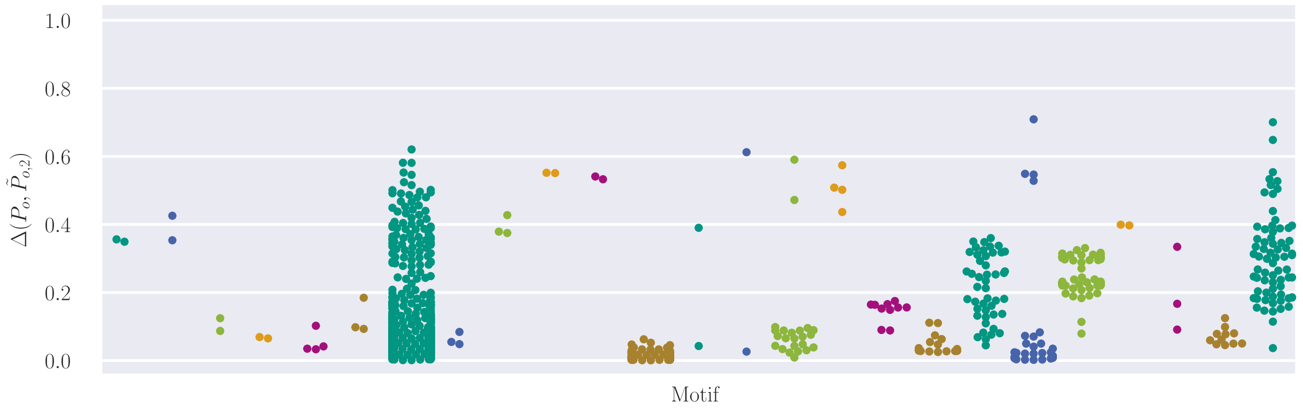


Figure 11: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 2$.

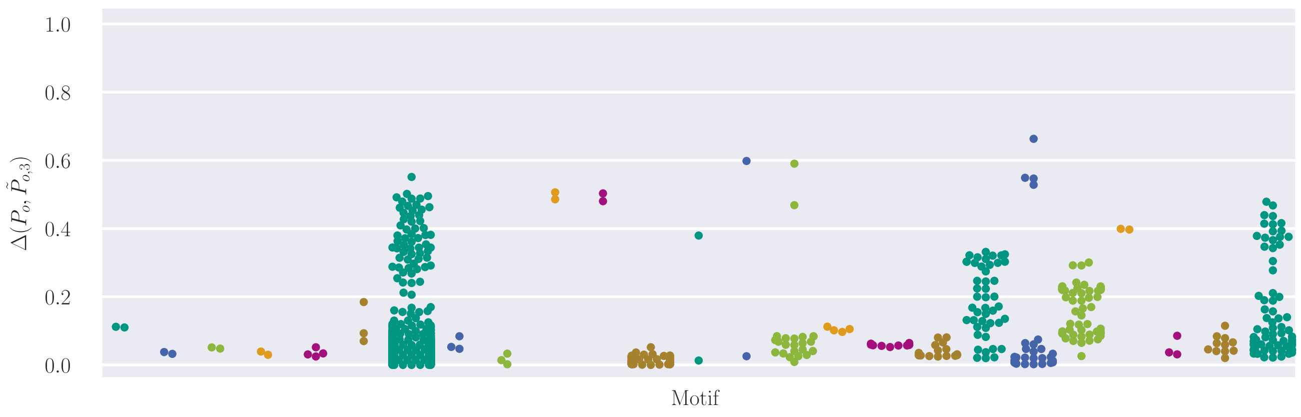


Figure 12: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 3$.

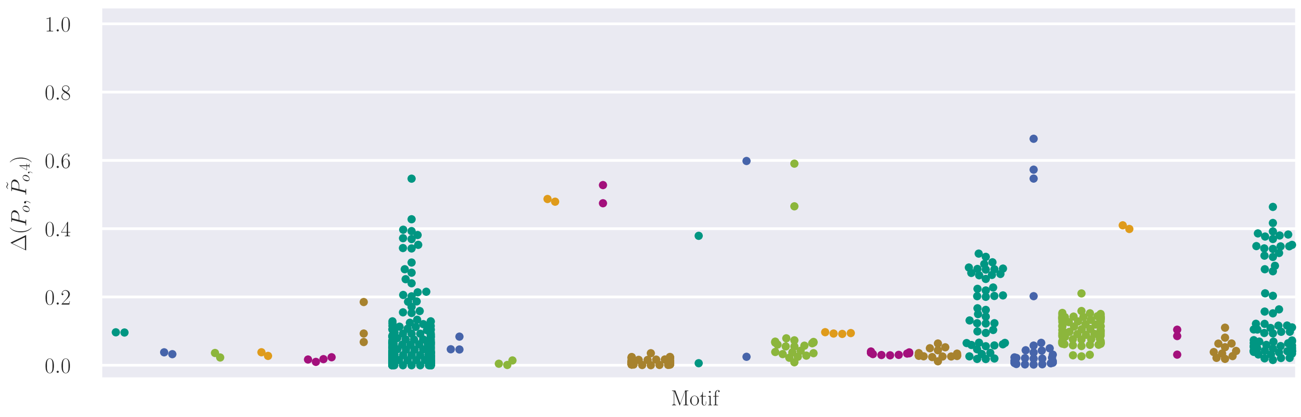


Figure 13: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 4$.

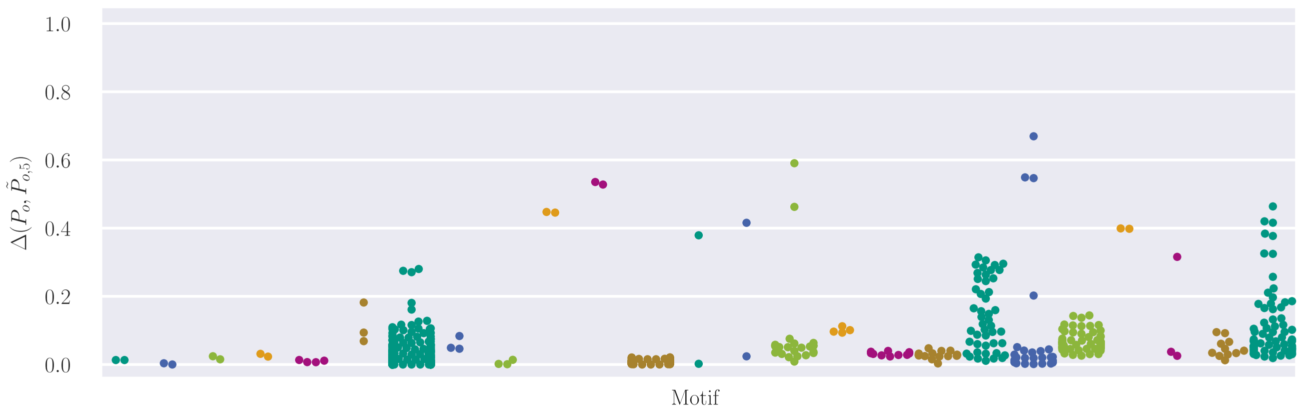


Figure 14: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 5$.

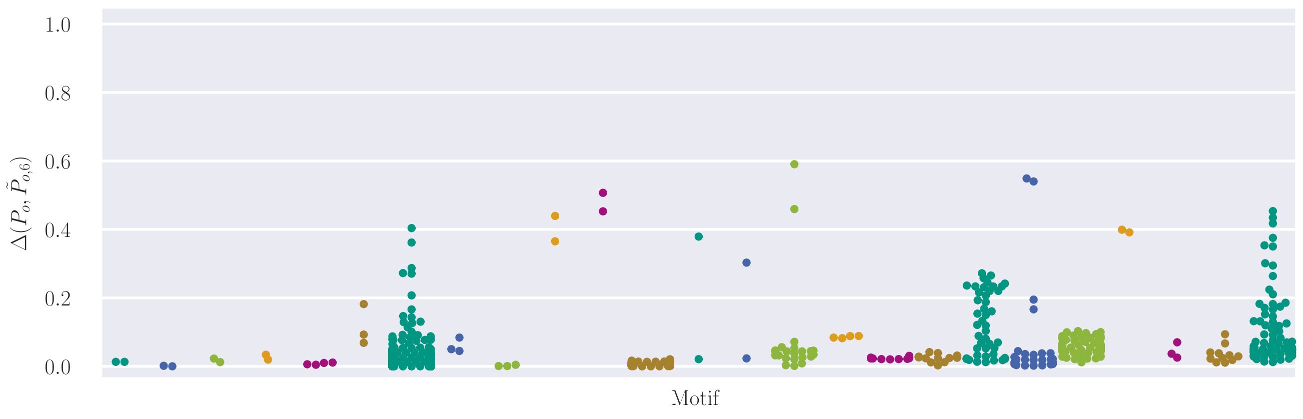


Figure 15: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 6$.

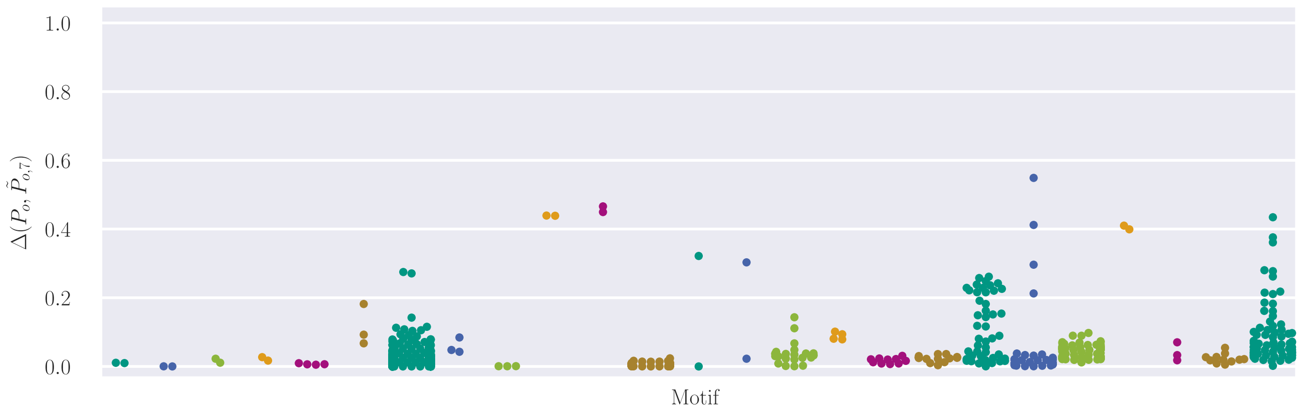


Figure 16: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 7$.

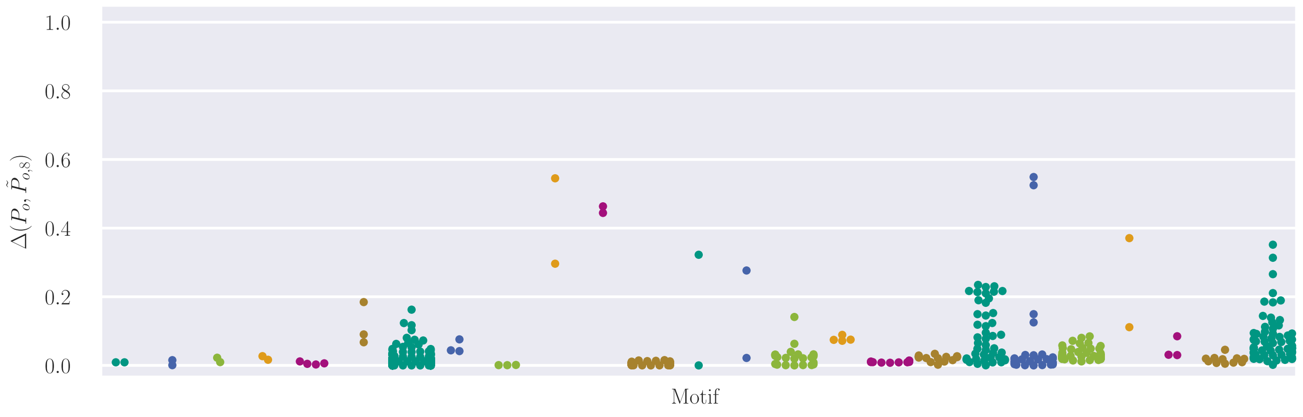


Figure 17: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 8$.

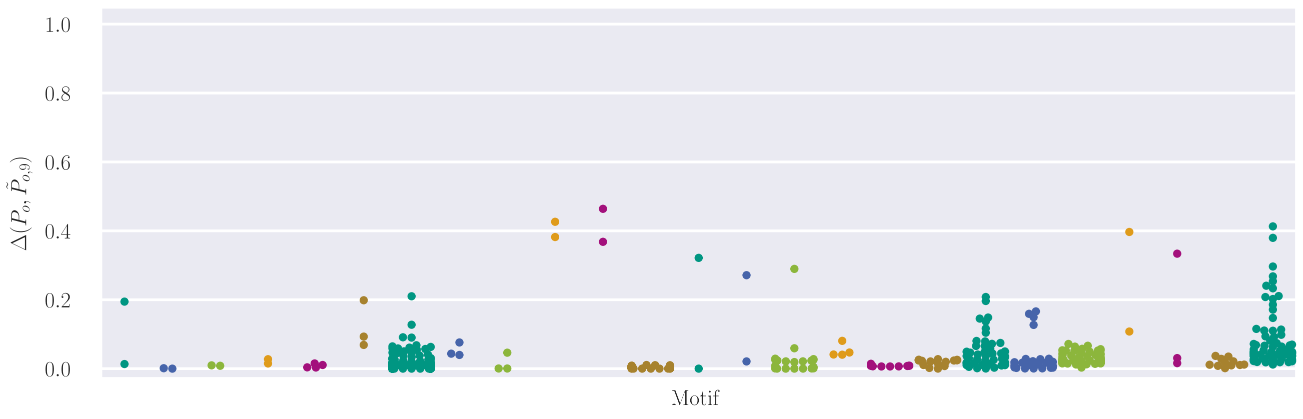


Figure 18: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 9$.

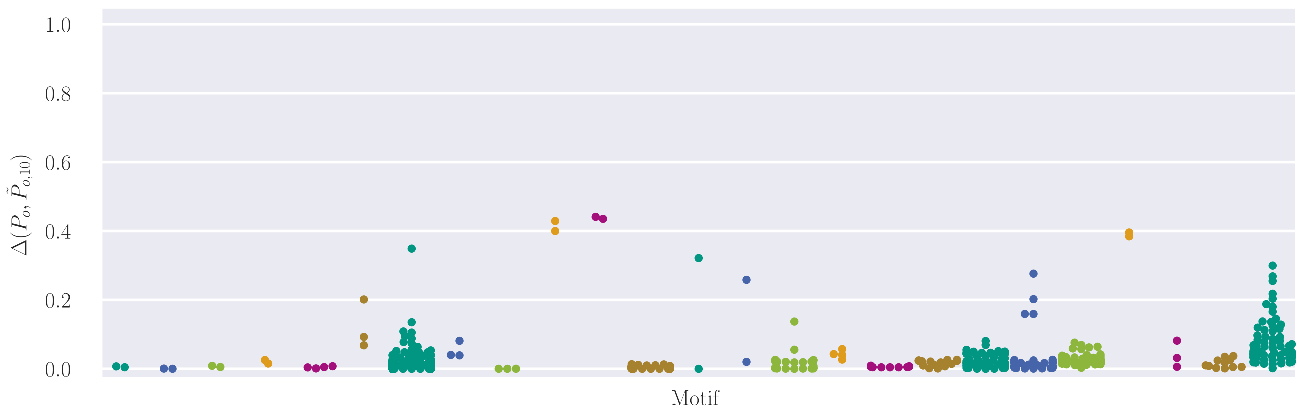


Figure 19: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 10$.

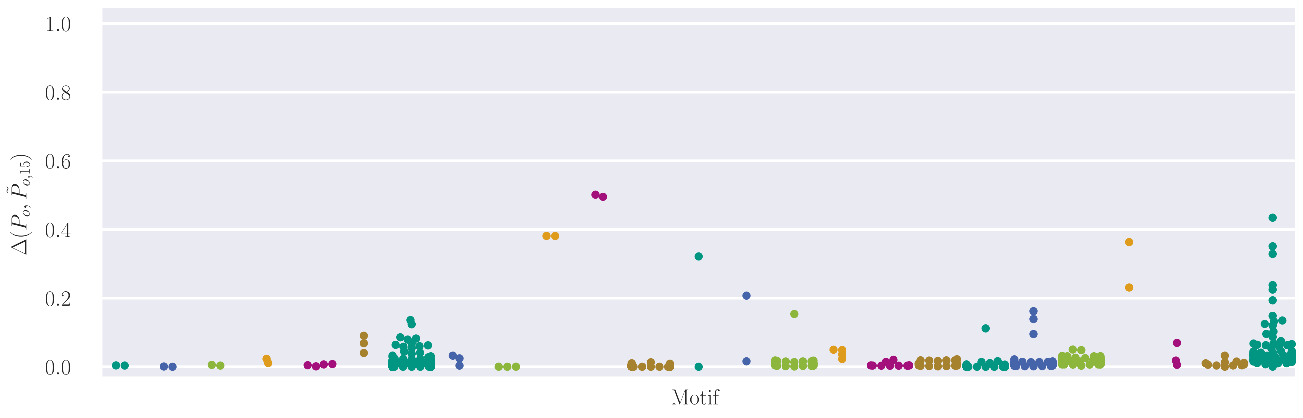


Figure 20: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 15$.

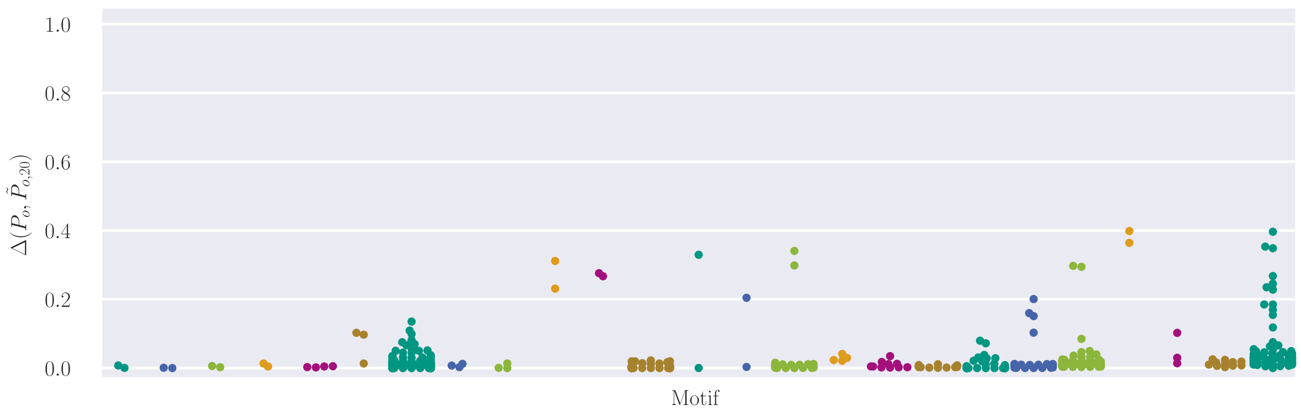


Figure 21: Δ measures for each occurrence, ordered by motif (on the x axis), for $k = 20$.