Comput Sci Res Dev (2018) 33:185–191
https://doi.org/10.1007/s00450-017-0357-4

energieinformatik2017
The Swiss-German-Austrian Research Summit on Energy Informatics

SPECIAL ISSUE PAPER

Exploiting flexibility in smart grids at scale

The resource utilization scheduling heuristic

Lukas Barth[1] · **Dorothea Wagner**[1]

Published online: 11 September 2017

Abstract Large parts of the worldwide energy system are undergoing drastic changes at the moment. Two of these changes are the increasing share of intermittent generation technologies and the advent of the smart grid. A possible application of smart grids is *demand response*, i.e., the ability to influence and control power demand to match it with fluctuating generation. We present a heuristic approach to coordinate large amounts of time-flexible loads in a smart grid with the aim of peak shaving with a focus on algorithmic efficiency. A practical evaluation shows that our approach scales to large instances and produces results that come close to optimality.

Keywords Smart grids · Demand response · Scheduling · Heuristics

1 Introduction

The electrical energy system of the future will be based on a *smart grid*, i.e., the confluence of communications and power transmission technology. An anticipated feature of smart grids is *demand response*, which is defined as "changes in electric usage by end-use customers from their normal consumption patterns […] to induce lower electricity use at times of high wholesale market prices or when system reliability is jeopardized" by the U.S. Department of Energy [17]. Demand response can be facilitated by various means, such as time-of-use pricing, dynamic tariffs with price signals, or centralized demand response. In the latter case, a central authority controls devices connected to the smart grid. These devices allow to control their load, for example by shifting it to a different time, modifying the shape of the load curve, or shedding the load altogether. An extensive survey regarding the possibilities of smart grids and demand response is given by Siano [16].

Throughout this paper, we deal with centralized demand response (also called direct load control) and *load shifting*, i.e., the shifting of the unmodified load curve of devices to a desirable time. While this might technically be possible in the foreseeable future, the owners of the devices might not want to relinquish control over their devices. This can be addressed with (financial) incentives for allowing such control; however, these considerations are beyond the scope of this paper.

Desirable and undesirable times for electrical loads can be the result of a high level of renewable generation in the electrical system. In contrast to conventional generation such as coal, gas or nuclear power, solar and wind power cannot be dispatched, i.e., the generation cannot be controlled to match a fluctuating demand. It can therefore be desirable to shift loads away from times of high demand or low solar and wind availability.

From this arises the objective of *peak shaving*, where the goal is to minimize the maximum power requirement that exceeds the renewable generation. Intuitively, this peak corresponds to the maximum capacity of the conventional generation that needs to be activated to satisfy all demand. Hence, demand response for peak shaving can reduce the necessary installed conventional capacity, as for example Zibelman and Krapels [19] show. Earle et al. [6] also come

Lukas Barth's 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.

✉ Lukas Barth
lukas.barth@kit.edu

[1] Karlsruhe Institute of Technology, Am Fasanengarten 5, 76131 Karlsruhe, Germany

to this result, but also argue that the positive effect is diminished if the demand response is uncertain, e.g. because it is based on price signals and relies on customers acting on these signals. With direct load control, such uncertainties do not exist.

1.1 Related work

Topics revolving around the smart grid are currently very active throughout the energy community. Fang et al. [7] give an overview over the developments in the field of smart grids. Approaches to exploit the flexibility offered by smart devices can be loosely separated into two groups, one considering flexibility in household contexts, one considering industry contexts. In the first group, Allerding et al. [1] present an evolutionary algorithm aimed at scheduling devices within a smart building that is equipped with generation. Li et al. [11] present a mixed-integer linear programming approach to schedule household appliances. Pedrasa et al. [13] use particle swarm optimization to schedule electrical loads in households, where some loads can be shed.

In the context of industry, Ashok [2] looks at steel plants and argues that these have a large potential for saving money by using their flexibility. Mitra et al. [12] consider continuous energy intensive processes and state a Mixed-Integer Program (MIP) to optimize these under fluctuating energy prices.

Some work has also gone into abstracting and unifying both household and industry contexts: Petersen et al. [14] develop a taxonomy of flexible electrical loads. Gottwalt et al. [8] describe how to select a portfolio of flexible electric loads to achieve maximum utility.

Besides the energy community, the task of scheduling flexible electric demands touches two fields of research: in operations research, the "time-constrained project scheduling problem" (TCPSP) is well known, which contains our problem as a special case. Guldemond et al. [9] give an overview over work related to the TCPSP and propose a solution technique in which jobs may miss their deadline (or need to be completed in "overtime"). The first ILP formulation for the problem is given by Deckro and Hebert [5].

The second field touched is the computer science field of machine scheduling. Here, the problem of minimizing the number of machines to schedule a set of jobs is similar to our problem. Cieliebak et al. [4] first introduce this idea, show its hardness and present efficient algorithms for special cases. Chuzhoy et al. [3] present an approximation algorithm to this problem. In machine scheduling problems, jobs usually only take one machine simultaneously. When applying machine scheduling approaches to smart grid scheduling, machines correspond to the power requirement of a job; thus, these approaches only directly apply to smart grid scheduling scenarios where all electrical loads have the same power requirement.

1.2 Contribution and outline

We present an iterative heuristic that minimizes electricity usage peaks using load shifting of directly controllable loads in smart grids—the *Resource Utilization Scheduling Heuristic* (RUSH).

Benefits from using flexibility in smart grids increase with the number of controlled loads. Thus, being able to optimize flexible devices at large scales is crucial. Most of prior research focuses on modeling sophisticated constraints or optimization criteria, but neglects algorithmic efficiency. Also, many approaches are based on meta heuristics which are not fine-tuned to the problem at hand. Our presented approach is focused on speed of computation even for very large instances. According to the principle of *algorithm engineering*, we intend to increase the model complexity in future research while trying to maintain algorithmic efficiency.

Section 2 formalizes the problem under study. In Sect. 3, we describe the details of the RUSH heuristic in detail. Section 4 presents an experimental evaluation of RUSH, analyzing its performance and comparing result quality to optimal results obtained via a mixed-integer linear program. In Sect. 5, we outline what further steps we are going to take with this research.

2 Problem formulation

We define our problem as a set of (electrical) loads with release times, deadlines, execution times and power requirements. These loads can represent for example individual cycles of devices such as refrigerators or A/C units, runs of one-off devices such as ovens, or batches in industrial processes.

Formally, let n be the number of loads. Then, each load $j \in \{1, \dots n\}$ is described by a tuple

$$(r_j, d_j, e_j, p_j) \in \mathbb{N}^3 \times \mathbb{R}$$

where r_j (the *release* time) is the first time step in which j may be executed, d_j (the *deadline*) is the first time step in which j must be finished, e_j (the execution time or *duration*) is the number of time steps j must be active consecutively and p_j is the amount of power required by j during its execution. Note that we model time steps as discrete, while power is continuous. Given all loads, we define a global release time $R = \min_i\{r_i\}$ and a global deadline $D = \max_i\{d_i\}$.

The objective is to find an assignment $s \in \mathbb{N}^n$ of start times that minimizes the maximum total power requirement over all time steps. For such an assignment to be *feasible*, it

must hold for every load j that $s_j \geq r_j$ and $s_j + e_i \leq d_i$. We will call a feasible assignment a *schedule*.

Note that this model aims to minimize the peak of all consumption. However, as stated in the introduction, one is usually interested in minimizing the (maximum) consumption that exceeds renewable generation, the so-called *residual load*. In our model, this can easily be achieved by introducing a set of immovable (by virtue of deadlines and release times) jobs that represent the difference between the amount of renewable generation available during the respective time interval and the maximum renewable generation.

3 Resource utilization scheduling heuristic

In this section, we describe how RUSH works, starting with a high level overview.

Given a problem as defined in Sect. 2 and a feasible schedule to the problem (i.e., a start-time assignment $s \in \mathbb{N}^n$), we define its *profile* as a sequence of intervals, each of which marks a time span in which the schedule requires roughly the same amount of power.

Formally, let $p_s(t)$ be the power requirement at timestep t induced by schedule s. To group timesteps with roughly the same power requirement together, we define a set of power *levels*. Let λ be a parameter specifying the size of each power level. We then say that during time step t, the schedule s executes in power level l if and only if $l\lambda \leq p_s(t) < (l+1)\lambda$.

For a given schedule, profile $\mathcal{P} = (\mathcal{I}, \mathcal{L})$ consists of a sequence of intervals $\mathcal{I}$ and a function $\mathcal{L} : \mathcal{I} \to \mathbb{R}^+$. Here, $\mathcal{I}$ is a sequence of consecutive, disjunct, right-open intervals spanning the whole scheduling horizon, i.e., $\cup_i \mathcal{I}_i = [R, D]$, $\cap_i \mathcal{I}_i = \emptyset$ and $\limsup(\mathcal{I}_i) = \min(\mathcal{I}_{i+1})$. Correspondingly, for $\mathcal{I}_i \in \mathcal{I}$, $\mathcal{L}(\mathcal{I}_i)$ states the power level that the schedule is in during $\mathcal{I}_i$.

The working principle of RUSH is to first generate a feasible schedule by scheduling every load at its release time, and then repeatedly pick a load j, determine the highest power level during the scheduled execution of j (let this be $\hat{l}$) and then move j so that the total time the schedule executes in power level $\hat{l}$ is minimized while not increasing the time executed in any level above $\hat{l}$.

For such an iterative approach it is desirable to have the individual iterations execute as efficiently as possible, so that the algorithm arrives at a satisfactory solution as quickly as possible. Because of this, most computations performed by RUSH can be implemented as a set of simple operations on sets of intervals. Abusing notation a bit, we treat $\mathcal{I}$ as a function which is the inverse of $\mathcal{L}$: Let $\mathcal{I}(l) = \{\mathcal{I}_i : \mathcal{L}(\mathcal{I}_i) = l\}$ be the set of intervals in the profile where the schedule is in level l. Also, for a fixed load j, let $\mathcal{P}^j = (\mathcal{I}^j, \mathcal{L}^j)$ be the profile derived from $\mathcal{P}$ by removing load j.

For a fixed load j, define the *base level* l^* of load j as the maximum level in $\mathcal{P}^j$ during the scheduled execution in j. Consider Fig. 1. Here, the maximum power demand *below* the highlighted load j is in level 3. Thus, $l^* = 3$ in this example.

Intuitively, executing j during any of $\mathcal{I}^j(l^*)$, i.e., the times in which the schedule without j executes in the base level, results in power consumption roughly equal to the old maximum power consumption during the execution of j. Thus, we try to minimize the amount of j we schedule during $\mathcal{I}^j(l^*)$. We also forbid scheduling any of j during any $\mathcal{I}^j(l^* + k)$ for any $k > 0$, i.e., no part of j may be scheduled on top of a level that is higher than the base level. This leaves $\mathcal{T}_j = \cup_{i \in \{0,\ldots l^*-1\}} \mathcal{I}^j(i)$ as the desirable area to schedule j in, called the *target*. In the example of Fig. 1, the target is everything in level 2 and below (after j has been removed), i.e., $\mathcal{T}_j = \{[0, 1), [6, 17)\}$. We further define a set of *forbidden* intervals, in which j may not be started. We do so by taking all intervals of levels above l^* in $\mathcal{I}^j$ and extending these intervals by $e_j - 1$ to the left:

$$\mathcal{F}_j = \left\{ [\max(0, a - e_j - 1), b) : [a, b) \in \bigcup_{i > l^*} \mathcal{I}^j(i) \right\}$$

In the example of Fig. 1, the only time the profile (after removal of j) is in a level above $l^* = 3$ is in $[17, 19)$. Since j has length 12, that results in $\mathcal{F}_j = \{[6, 19)\}$. It is easy to see that starting j during any of these intervals would cause a part of j to be scheduled during a higher power usage than before.

Observing that $d_j - e_j + 1$ is the latest time step at which we can start j without missing its deadline, we now combine everything. We want to find a start point in $[r_j, d_j - e_j + 1) \backslash \mathcal{F}_j$ that maximizes j's overlap with $\mathcal{T}$. We can show that to find this optimum position, it is sufficient to check the borders of the intervals in $[r_j, d_j) \backslash \mathcal{F}_j$ plus all time steps t for which j would be right-aligned in an interval in $\mathcal{T}_j \cap [r_j, d_j)$, i.e., the set of candidates for start positions is

$$\begin{aligned} \mathcal{C}_j = &\{x : [x, \cdot) \in ([r_j, d_j - e_j + 1) \backslash \mathcal{F}_j)\} \\ &\cup \{x : [\cdot, x) \in ([r_j, d_j - e_j + 1) \backslash \mathcal{F}_j)\} \\ &\cup \{\max(x - e_j, 0) : [\cdot, x) \in \mathcal{T} \cap [r_j, d_j)\} \end{aligned}$$

For each $\hat{s} \in \mathcal{C}_j$, we check the size of $[\hat{s}, \hat{s} + e_j) \cap \mathcal{T}_j$ and select the start point with the largest overlap.

Putting everything together, one iteration of RUSH works in these five steps:

1. Randomly select a load $j \in \{1, \ldots n\}$
2. Remove j from the current schedule
3. Compute $\mathcal{I}^j$, $\mathcal{L}^j$, $\mathcal{T}_j$, $\mathcal{F}_j$ and $\mathcal{C}_j$ as above
4. In $\mathcal{C}_j$, find the start candidate that results in the largest overlap with $\mathcal{T}_j$
5. Re-insert j into the schedule at this point

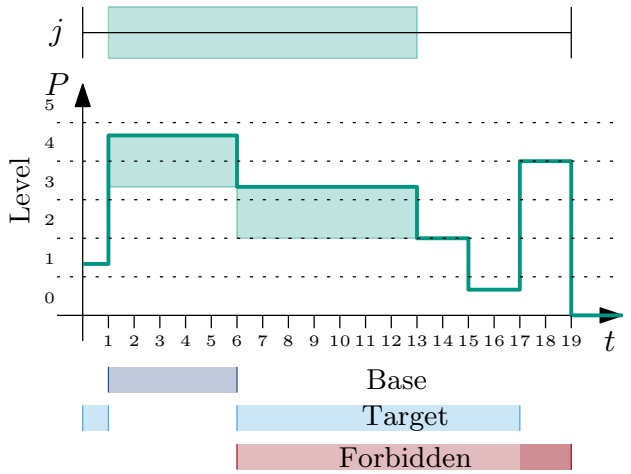

Fig. 1 Example setting before moving load j. The *green rectangle* corresponds to the length and power requirement of j. On the top, the release and deadline of j are shown together with j. Below is a graphical depiction of a possible profile, its discretization into levels, and j's contribution to it. Base level, target and forbidden sets are indicated below (color figure online)

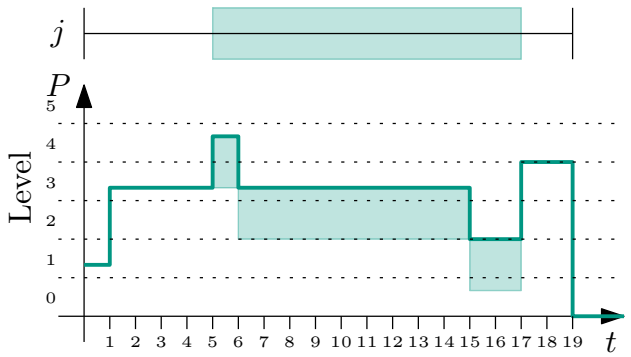

Fig. 2 Example setting after moving load j

In the example in Fig. 1, the set of candidates is $\{0\}\cup\{5\}\cup\{5\}$ as per the above formula. Indeed, setting $s_j = 5$ results in the largest overlap of j with the target of $\{[0, 1), [6, 17)\}$. Moving j to start at time step 5 results in j being executed for only one time step inside level 4, as seen in Fig. 2 instead of five time steps in Fig. 1.

Efficient implementation The sets computed in Step 3 above can all be computed via efficient set operations from $\mathcal{L}$ and $\mathcal{I}$. More precisely, for $\mathcal{T}_j$ (resp. $\mathcal{F}_j$), we need unions in the style of $\cup_{i\geq k}\mathcal{I}(i)$ (resp. $\cup_{i\leq k}\mathcal{I}(i)$) for some value k. To facilitate efficient access to these unions, we store each of them directly. However, since in our case $\cup_{i\geq k}\mathcal{I}(i) = [R, D)\setminus \cup_{i\leq k-1}\mathcal{I}(i)$, it suffices to store $\cup_{i\leq k-1}\mathcal{I}(i)$ for all possible levels k. We do so using Boost's ICL datastructures.[1]

[1] http://www.boost.org/doc/libs/1_64_0/libs/icl/doc/html/index.html.

4 Experimental evaluation

We evaluate RUSH experimentally by using it to schedule a set of instances of the problem introduced in Sect. 2. Runs of RUSH were conducted on a machine with four Intel Xeon E5-1630 cores at 3.7 GHz (of which only one was used) and 128 GB of RAM, the baseline Mixed-Integer Program (MIP) was run on a machine with 16 Intel Xeon E5-2670 cores at 2.6 GHz and 64GB of RAM, using Gurobi 6.5 as a solver.

We generated two groups of instances: One set of medium-sized instances and a set of very large instances. For the medium instances, the MIP described in Sect. 4.1 yields acceptable lower bounds within reasonable time, allowing us to compare the solution quality obtained by RUSH to a lower bound. On the set of large instances, we demonstrate the scalability of RUSH.

For the set of medium-sized instances, the number of loads per instance was drawn uniformly at random from the range [100, 400]. For the large instances, we generated 10000 loads per instance.

The duration of all loads was also randomly drawn, from a normal distribution with a mean of 30, a standard deviation of 20, and a minimum value of 1. For each load, we assigned the release time uniformly at random between 0 and 200 (0 and 2000 for the large instances), and the *slack*, that is the difference between deadline minus release time and execution time, i.e., the amount of flexibility of a load, uniformly between 0 and 200 (0 and 2000 for large instances).

Finding randomly generated instances which adequately represent real smart grid scheduling problems is not easy, and has been done in various ways throughout literature. Petersen et al. [14] randomly chose all loads lengths from $\{2, 3, 4, 5\}$, all power requirements from $\{1, 2, 3, 4\}$ and the deadline for each load from $\{1, \ldots 100\}$ while all loads are released at $t = 1$. Li et al. [11] explicitly model four household appliances: kettles, toasters, ovens and refrigerators. Each has a unique power consumption and pattern of release, deadline and duration. Yaw et al. [18] also model individual household appliances, however they base their models on consumption profiles obtained from the REDD dataset [10]. We intentionally chose to generate our instances randomly and not by explicit modeling, since we think that a small number of household appliances alone are not sufficient to represent the various demands in the electrical system. By picking values from a continuous distribution instead of a fixed set we tried to have as much diversity among the loads as possible.

On each of these instances, we ran the MIP described in Sect. 4.1 for a maximum of 1200 s. For most instances, an optimal solution could not be found in this time. However, from the MIP runs we obtain lower bounds on the optimal solution quality as well as non-optimal feasible solutions.

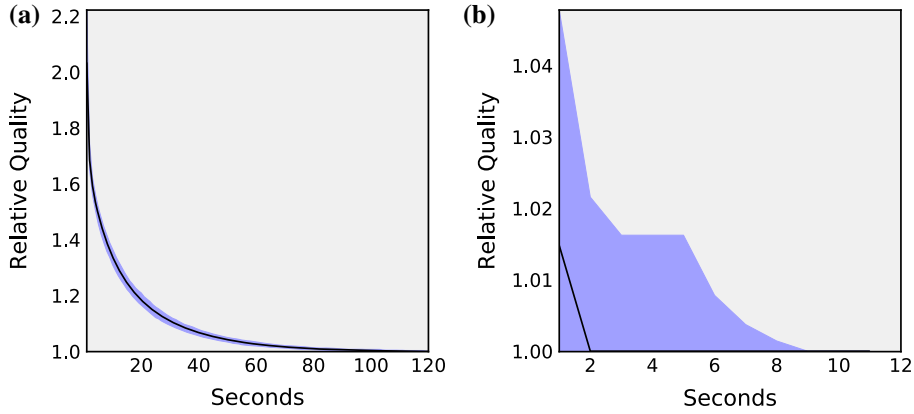

Fig. 3 Convergence speed of RUSH⟨4000⟩. Measurements were taken every second. All measurements fall within the *blue area*. The *black line* indicates the median value at every point in time. All results are normalized by the optimum value RUSH⟨4000⟩ computes on the respective instance. **a** large instances, **b** medium instances (color figure online)

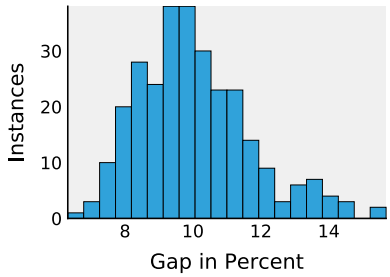

Fig. 4 Histogram of the gaps achieved by the MIP solver

Table 1 Median, upper quartile and maximum values for the quality gap achieved by the RUSH variants

Algorithm	Median	0.75-quantile	Max
RUSH⟨20⟩	13.5	15.3	23.7
RUSH⟨40⟩	10.5	12.1	17.5
RUSH⟨60⟩	9.1	10.4	16.6
RUSH⟨200⟩	7.4	9.4	17.2
RUSH⟨1000⟩	7.4	9.4	17.5
RUSH⟨4000⟩	7.6	9.4	18.5

4.1 Baseline mixed-integer linear program (MIP)

The MIP used to compute a baseline is based on the formulation by Pritsker et al. [15]. The basic idea is to introduce a matrix of $n \times (D - R)$ binary variables x_{jt}, where $x_{jt} = 1$ if and only if load j starts in time step t. The original MIP formulation solves the Resource-Constrained Project Scheduling problem, but rewriting it for the Time-Constrained Project Scheduling problem, of which our problem is a special case, is straightforward.

4.2 Results

It is easy to see that the quality achieved by RUSH depends on the number of levels that RUSH discretizes the power requirement into. We ran RUSH on all instances with 20, 40, 60, 200, 1000 and 4000 levels each. We write RUSH with k power levels as RUSH⟨k⟩.

We first examine the speed with which RUSH converges against a solution. The individual iterations become more expensive the finer the power consumption is discretized, i.e., the more power levels RUSH uses. Thus, we mainly look at the speed of RUSH⟨4000⟩, the largest number of levels we evaluated.

Figure 3 shows for every of the runs of RUSH⟨4000⟩ on each instance of medium resp. large size how close each computation got to the final result after what time. Note that 1.0 in this case does not indicate the global optimum of the respective instance, but rather the optimal value that RUSH achieved. Measurements were taken every second. While on medium instances, the near-optimum value is achieved after 10 s for all runs, it takes about 100 s on the large instances. On medium instances, the median solution is near-optimal after 2 s already.

We now take a look at the quality of the computed solutions. The MIP described in Sect. 4.1 is not suited to solve a significant number of medium-sized instances to optimality within reasonable time. Therefore, we take the average between lower bound and quality of the best feasible solution found by the MIP as baseline. Since the MIP does not compute anything useful on the large instances, the quality comparison is only done on the medium instances. Figure 4 shows a histogram of the gaps that the MIP achieved on the medium instances after 1200 s, i.e., the ratio between best feasible solution found and best lower bound.

Figure 5 show histograms of the gap between the baseline and the solutions achieved by RUSH with the various levels. Table 1 reports median, upper quartile and maximum values. It can be seen that going from 20 to 200 levels, the average solution quality improves significantly, even though the worst case is not improved by much after 40 levels. From 200 levels upwards, result quality does not improve much anymore. In fact, solution quality does even deteriorate marginally when going from 1000 to 4000 levels. This could be explained by the fact that in RUSH⟨4000⟩, individual iterations are

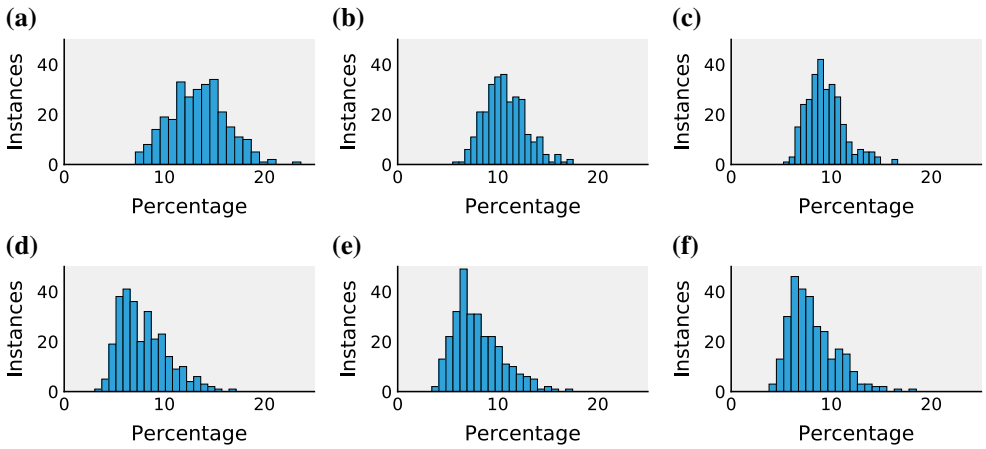

Fig. 5 Relative quality of variations of RUSH compared to the baseline derived from the MIP. **a** RUSH⟨20⟩, **b** RUSH⟨40⟩, **c** RUSH⟨60⟩, **d** RUSH⟨200⟩, **e** RUSH⟨1000⟩ and **f** RUSH⟨4000⟩

more expensive and therefore less iterations can be computed within the given time limit.

5 Conclusion and future work

We have presented RUSH, an iterative heuristic to exploit flexibility in smart grids on a large scale. We have shown that RUSH yields results with good quality in short time. Still, this approach is research in progress.

There are several directions that seem promising. First, RUSH can be improved to yield even better results. Second, RUSH can be extended to be applicable to more complex models. Finally, a more thorough evaluation of RUSH, and a direct comparison against prior algorithmic approaches should be done.

To improve RUSH, we intend to find a smarter way of (randomly) selecting the load to be moved in each iteration.

Also, we did not yet pay any attention to the solution RUSH starts with. Using a simple list scheduling heuristic instead of initially scheduling every load at its release time might lead to faster convergence.

In terms of extending RUSH, the first thing that comes to mind is adding dependencies between loads, i.e., allowing loads to only start after some predecessors finished. What needs to be done is to limit the feasible range of start candidates in Step 4 of the algorithm presented in Sect. 3. Regarding the evaluation, while we show that RUSH arrives at good solutions in very short time, solution quality and performance should be directly compared to competing algorithmic solutions such as the GRASP-based metaheuristic by Petersen et al. [14] or the PDM heuristic by Yaw et al. [18].

Finally, since our heuristic works iteratively, a technique like simulated annealing to climb out of local optima seems applicable: After every iteration, one can decide to accept the modified solution or go back to the previous solution. A scheme in which this decision is driven by some cool down factor as in simulated annealing might be beneficial.

References

1. Allerding F, Mauser I, Schmeck H (2014) Customizable energy management in smart buildings using evolutionary algorithms. Springer, Berlin, pp 153–164. doi:10.1007/978-3-662-45523-4_13
2. Ashok S (2006) Peak-load management in steel plants. Appl Energy 83(5):413–424. doi:10.1016/j.apenergy.2005.05.002
3. Chuzhoy J, Guha S, Khanna S, Naor JS (2004) Machine minimization for scheduling jobs with interval constraints. In: 45th annual IEEE symposium on foundations of computer science, pp 81–90. doi:10.1109/FOCS.2004.38
4. Cieliebak M, Erlebach T, Hennecke F, Weber B, Widmayer P (2004) Scheduling with release times and deadlines on a minimum number of machines. Springer, Boston, pp 209–222. doi:10.1007/1-4020-8141-3_18
5. Deckro RF, Hebert JE (1989) Resource constrained project crashing. Omega 17(1):69–79. doi:10.1016/0305-0483(89)90022-4
6. Earle R, Kahn EP, Macan E (2009) Measuring the capacity impacts of demand response. Electr J 22(6):47–58. doi:10.1016/j.tej.2009.05.014
7. Fang X, Misra S, Xue G, Yang D (2012) Smart grid the new and improved power grid: a survey. IEEE Commun Surv Tutor 14(4):944–980. doi:10.1109/SURV.2011.101911.00087
8. Gottwalt S, Grttner J, Schmeck H, Weinhardt C (2016) Modeling and valuation of residential demand flexibility for renewable energy integration. IEEE Trans Smart Grid PP(99):1–10. doi:10.1109/TSG.2016.2529424
9. Guldemond TA, Hurink JL, Paulus JJ, Schutten JMJ (2008) Time-constrained project scheduling. J Sched 11(2):137–148. doi:10.1007/s10951-008-0059-7
10. Kolter JZ, Johnson, MJ (2011) Redd: a public data set for energy disaggregation research. In: SustKDD
11. Li Y, Ng BL, Trayer M, Liu L (2012) Automated residential demand response: algorithmic implications of pricing models. IEEE Trans Smart Grid 3(4):1712–1721. doi:10.1109/TSG.2012.2218262
12. Mitra S, Grossmann IE, Pinto JM, Arora N (2012) Optimal production planning under time-sensitive electricity prices for continuous power-intensive processes. Comput Chem Eng 38:171–184. doi:10.1016/j.compchemeng.2011.09.019

13. Pedrasa MAA, Spooner TD, MacGill IF (2010) Coordinated scheduling of residential distributed energy resources to optimize smart home energy services. IEEE Trans Smart Grid 1(2):134–143. doi:10.1109/TSG.2010.2053053
14. Petersen MK, Hansen LH, Bendtsen J, Edlund K, Stoustrup J (2014) Heuristic optimization for the discrete virtual power plant dispatch problem. IEEE Trans Smart Grid 5(6):2910–2918. doi:10.1109/TSG.2014.2336261
15. Pritsker AAB, Waiters LJ, Wolfe PM (1969) Multiproject scheduling with limited resources: a zero-one programming approach. Manag Sci 16(1):93–108
16. Siano P (2014) Demand response and smart grids a survey. Renew Sustain Energy Rev 30:461–478. doi:10.1016/j.rser.2013.10.022
17. US Department of Energy (2006) Benefits of demand response in electricity markets and recommendations for achieving them
18. Yaw S, Mumey B, McDonald E, Lemke J (2014) Peak demand scheduling in the smart grid. In: 2014 IEEE international conference on smart grid communications (SmartGridComm), pp 770–775. doi:10.1109/SmartGridComm.2014.7007741
19. Zibelman A, Krapels EN (2008) Deployment of demand response as a real-time resource in organized markets. Electr J 21(5):51–56. doi:10.1016/j.tej.2008.05.011