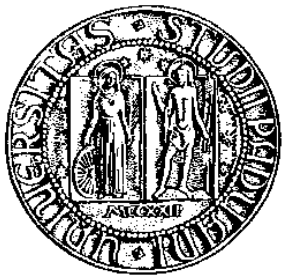
UNIVERSITÀ DEGLI STUDI DI PADOVA
FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA DELL'AUTOMAZIONE



TESI DI LAUREA

OPTIMAL SCHEDULING OF SMART HOME APPLIANCES USING MIXED-INTEGER LINEAR PROGRAMMING

Relatore: Ch.mo Prof. Fischetti Matteo

Laureando: Sartor Giorgio

Anno Accademico 2012–2013
Padova, 10 dicembre 2012

Ai miei genitori e ad Elena

Acknowledgments

I would like to thank Prof. Matteo Fischetti, the best supervisor that a student could wish for.

A special acknowledgment to Ing. Arrigo Zanette, who oversaw this work with outstanding kindness.

Special thanks also to M31 Italia S.r.l, who supported me during this project.

Contents

Sommario

Negli ultimi anni abbiamo assistito ad un crescente interesse per gli edifici intelligenti (smart buildings), in particolare per quanto riguarda la gestione efficiente dell'energia, le fonti di energia rinnovabile e gli elettrodomestici intelligenti (smart appliances). In questa tesi poniamo la nostra attenzione sul problema dell'allocazione (scheduling) degli elettrodomestici intelligenti lungo un certo periodo della giornata, tenendo anche conto di fonti di energia alternativa e batterie di accumulo. Il profilo energetico di una smart appliance è modellato attraverso una sequenza di fasi, ciascuna delle quali ha il proprio consumo energetico e le proprio caratteristiche. Lo scopo è quello di allocare gli elettrodomestici in modo da ridurre il conto totale in bolletta, rispettando le caratteristiche dei profili energetici e le preferenze dell'utente. Viene progettato un modello a programmazione lineare intera (MIP) e un algoritmo euristico greedy, con l'idea di combinarli assieme. Mostreremo come una generica procedura di raffinamento sfruttata dal modello MIP può essere usata per migliorare, in poco tempo, la qualità della soluzione fornita dall'algoritmo euristico. I risultati ottenuti confermano la validità di questo approccio, sia in termini di qualità della soluzione sia di velocità.

Parole chiave Mateuristico, Programmazione Lineare Intera, euristiche di raffinamento, gestione dell'energia, case intelligenti

Abstract

In the last years we have witnessed an increasing interest in smart buildings, in particular for what concerns optimal energy management, renewable energy sources, and smart appliances. In this paper we investigate the problem of scheduling smart appliance operation in a given time horizon with a set of energy sources and accumulators. Appliance operation is modeled in terms of uninterruptible sequential phases with a given power demand, with the goal of minimizing the energy bill fulfilling duration, energy, and user preference constraints. A Mixed Integer Linear Programming (MIP) model and a greedy heuristic algorithm are given, intended to be used in a synergic way. We show how a general purpose (off-the-shelf) MIP refining procedure can effectively be used for improving, in short computing time, the quality of the solutions provided by the initial greedy heuristic. Computational results confirm the viability of the overall approach, in terms of both solution quality and speed.

Keywords Matheuristics, Mixed-Integer Programming, Refinement heuristics, Energy Management, Smart Houses

1 Introduction

Energy optimization is attracting an increasing interest amongst researchers as long as new "smarter" infrastructures and devices are going to replace the traditional ones.

The most popular scenario involves a new concept of electrical grid, the *Smart Grid* (Fig. 1) that allows to convey a two-way flow of electricity and information between central generators and customers [10]. Although there are many ways of addressing the demand side energy management problem like using neural network [6] or PID neural networks [11], multi-agent systems [25], fuzzy control [17] and even ant colony optimization [8], the most common way is by solving a scheduling problem. This involves multiple appliances with different operational constraints, user preferences, renewable energy sources and batteries. Other authors have investigated variants of the appliance scheduling problem, Hatami and Pedram [14] by taking the interaction among different users into account, Zhang et al. [24] by considering a so-called microgrid, and Agnetis et al. [2] by addressing additional thermal comfort constraints.

Mixed Integer Programming (MIP) models from the literature allow for an effective mathematical formulation of the appliance scheduling problem. Barbato et al. [3] also take the photovoltaic energy into account, and a linearized description of battery charge states is given. Sou et al. [18] provide a detailed MIP formulation of appliance power profiles and operations, and model appliance operations as a set of sequential uninterruptible phases with variable inter-phase delays.

As far as the solution of appliance scheduling problem is concerned, Carpentieri et al. [7] propose an LP-rounding heuristic for solving the appliance scheduling problem with the goal of minimizing the maximum peak energy of multiple houses. Barbato et al. [4] use different heuristics to address the problem of online recovering an offline schedule taking into account the real parameters.

With this paper we want to propose a solution to the energy management

problem using a combination between greedy heuristic algorithm and the recent technique of *polishing* in a smart house scenario (Fig. 2).

Many successful matheuristic schemes use a black-box MIP solver to generate high-quality heuristic solutions for difficult optimization problems. The hallmark of this approach is the availability of a (possibly incomplete) MIP model of the problem at hand, and of an external metascheme that iteratively solves sub-MIPs obtained by introducing invalid constraints (e.g., variable fixings) defining "interesting" neighborhoods of certain solutions. The goal of the approach is to iteratively refine the incumbent solution, producing a sequence of better and better feasible solutions in short (or, at least, acceptable) computing times.

The above solution-refinement approach is completely general, i.e., it can in principle be applied to the original MIP without the need of ad-hoc adaptations. An example of a general MIP refinement procedure is the evolutionary *polishing method* of Rotbergh [15] that automatically defines sub-MIPs to combine a population of feasible solutions. (Interesting enough for practitioners, an implementation of the polishing heuristic is available in some commercial MIP solvers, hence it can be used off-the-shelf.) A more recent approach is the *proximity search* by Fischetti and Monaci [12], where the objective function of the original MIP is modified with the aim of attracting the search in the neighborhood of the incumbent, without the need of adding any additional constraint.

In very difficult cases, however, the approach based on general MIP refinement is not successful, and one tends to design ad-hoc matheuristics that exploit the structure of the problem. As a matter of fact, an issue with the general approach is the lack of good (or even feasible) solutions to refine. In this context, one can argue that ad-hoc heuristics and general MIP refinement procedures are complementary one to each other, the former being typically able to find feasible solutions very quickly, while the latter can exploit the underlying MIP model to improve them by reaching a quality degree that is difficult to attain otherwise.

In the present paper we argue that the application of above scheme can lead to a very effective overall heuristic even in case a very simple greedy is used to feed the general MIP refinement module with feasible solutions. The resulting *MIP-and-refine* approach is exemplified and tested in the context of smart grid energy management, whose underlying MIP models turn out to be very difficult to solve without the hints provided by an external heuristic.

Below we present a description of the Smart Grid scenario as well as the MIP modeling, the heuristic algorithm and the *polishing technique*. Moreover we introduce the real scenario where the greedy algorithm will be applied 1.5. In section 2 we propose a MIP model of the scheduling problem whereas in section 3 a greedy heuristic to solve it. In section 5 we show some results.

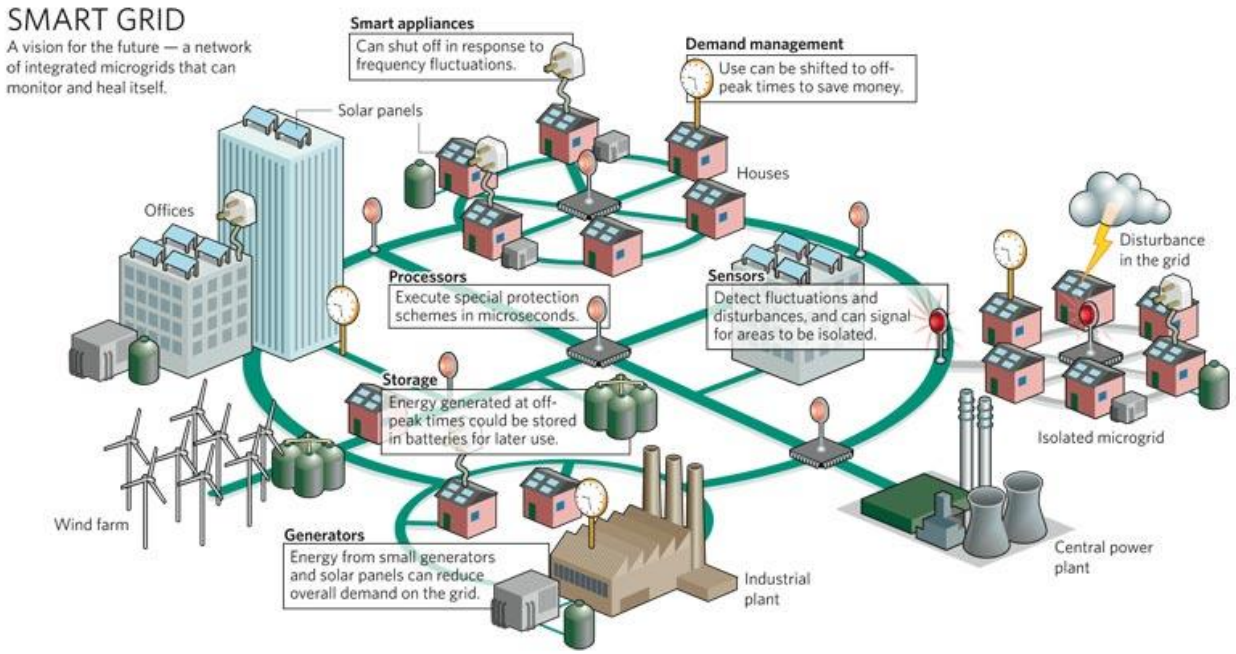


Figure 1: Example of a smartgrid

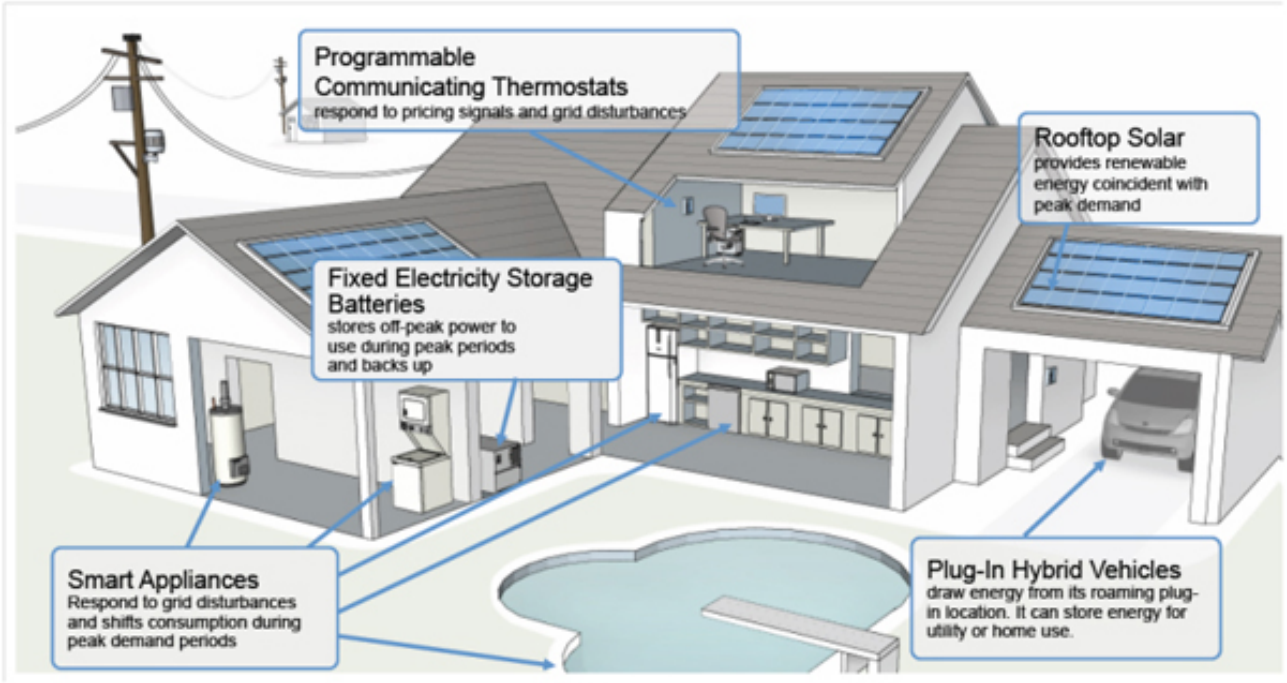


Figure 2: Example of a smart house

1.1 Smart Grid

As described in [1], the term "smart grid" refers to an electricity transmission and distribution system that incorporates elements of traditional and cutting-edge power engineering, sophisticated sensing and monitoring technology, information technology, and communications to provide better grid performance and to support a wide array of additional services to consumers. A smart grid is not defined by what technologies it incorporates, but rather by what it can do. The key attributes of the 21st century grid include the following:

- *The grid will be "self-healing."* Sophisticated grid monitors and controls will anticipate and instantly respond to system problems in order to avoid or mitigate power outages and power quality problems.

- *The grid will be more secure from physical and cyber threats.* Deployment of new technology will allow better identification and response to manmade or natural disruptions.

- *The grid will support widespread use of distributed generation.* Standardized power and communications interfaces will allow customers to interconnect fuel cells, renewable generation, and other distributed generation on a simple plug and play basis.

- *The grid will enable consumers to better control the appliances and equipment in their homes and businesses.* The grid will interconnect with energy management systems in smart buildings to enable customers to manage their energy use and reduce their energy costs.

- *The grid will achieve greater throughput, thus lowering power costs.* Grid upgrades that increase the throughput of the transmission grid and optimize power flows will reduce waste and maximize use of the lowest-cost generation resources. Better harmonization of the distribution and local load servicing functions with interregional energy flows

and transmission traffic will also improve utilization of the existing system assets.

Smart Grid benefits are fully exploited only if the grid endpoints, home appliances for examples, are smart as well. Smart appliances are able to exchange data with the grid, such as dynamic energy prices and grid status. Along with user preferences, this information can be used to optimally manage the energy demand in order to reduce the customer energy bill and to prevent major blackouts.
Each appliance, to be called "smart", must have some particular features:

- Computational capability
- Sensors
- Interconnection capability

The smart appliance must be able to measure something, analyze or process the measure and transmit it to the other appliances or to a central station. Hence, a smart appliance have to be interconnected and able to communicate with each other. This could be possible by a "classic" wired connection or, in a more fashion way, using wireless sensor network (WSN). Tung et al. [20], Yiming et al. [22], Tsang et al. [19] and Dae et al. [13] show this application using ZigBee, that is a specification for a suite of high level communication protocols using small, low-power digital radios based on an IEEE 802 standard for personal area networks [26].

1.2 MIP

Linear programming (LP, or linear optimization) is a mathematical method for determining a way to achieve the best outcome (such as maximum profit or lowest cost) in a given mathematical model for some list of requirements represented as linear relationships [16]. Linear programming is a specific case

of mathematical programming (mathematical optimization). More formally, linear programming is a technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints. Its feasible region is a convex polyhedron, which is a set defined as the intersection of finitely many half spaces, each of which is defined by a linear inequality. Its objective function is a real-valued affine function defined on this polyhedron. A linear programming algorithm finds a point in the polyhedron where this function has the smallest (or largest) value if such a point exists. Linear programs are problems that can be expressed in canonical form:

$$\begin{aligned} \textit{min} \quad & \mathbf{c^T x} \\ \textit{subject to} \quad & \mathbf{Ax} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

where $\mathbf{x}$ represents the vector of variables (to be determined), $\mathbf{c}$ and $\mathbf{b}$ are vectors of (known) coefficients, $\mathbf{A}$ is a (known) matrix of coefficients, and $(\cdot)^T$ is the matrix transpose. The expression to be maximized or minimized is called the objective function ($\mathbf{c^T x}$ in this case). The inequalities $\mathbf{Ax} \geq \mathbf{b}$ are the constraints which specify a convex polytope over which the objective function is to be optimized. Linear programming can be applied to various fields of study. It is used in business and economics, but can also be utilized for some engineering problems. Industries that use linear programming models include transportation, energy, telecommunications, and manufacturing. It has proved useful in modeling diverse types of problems in planning, routing, scheduling, assignment, and design.

Integer programming (IP) adds additional constraints to linear programming. In particular it adds the requirement that some or all of the variables take on integer values. This seemingly innocuous change greatly increases the number of problems that can be modeled, but also makes the models more difficult to solve. In fact, two seemingly similar formulations for the same problem (one integer and the other one linear) can lead to radically different computational experience. Integer programming is NP-hard.

Logical condition	Binary condition
At most N of $a, b, c, \ldots$	$a + b + c + \ldots \leq N$
At least N of $a, b, c, \ldots$	$a + b + c + \ldots \geq N$
Exactly N of $a, b, c, \ldots$	$a + b + c + \ldots = N$
If a then b	$b \geq a$
Not b	$\overline{b} = 1 - b$
If a then not b	$a + b \leq 1$
If not a then b	$a + b \geq 1$
If a then b, and if b then a	$a = b$
If a then b and c; a only if b and c	$b \geq a$ and $c \geq a$
If a then b or c	$b + c \geq a$
If b or c then a	$a \geq b$ and $a \geq c$
If b and c then a	$a \geq b + c - 1$

Table 1: Logical conditions to Binary condition. All variables $\in \{0, 1\}$

MIP (Mixed Integer Programming) is a generalization of LP in which the variables of the linear model are integer. In some cases the variables could be also binary.

The binary modeling can be very tricky sometimes because our thinking is not used to. In Table 1 are presented some useful transformations of logical conditions to binary conditions.

1.3 Greedy heuristic algorithm

Cormen et al. [9] define the greedy algorithm as an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum. In many problems, a greedy strategy does not in general produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a global optimal solution in a reasonable time.

Most of the problems that can be solved using greedy approach have two parts [5]:

- *Greedy choice property*: Globally optimal solution can be obtained by making locally optimal choice and the choice at present cannot reflect possible choices at future.
- *Optimal substructure*: Optimal substructure is exhibited by a problem if an optimal solution to the problem contains optimal solutions to the subproblems within it.

To prove that a greedy algorithm is optimal we must show the above two parts are exhibited. For this purpose first take globally optimal solution; then show that the greedy choice at the first step generates the same but the smaller problem, here greedy choice must be made at first and it should be the part of an optimal solution; at last we should be able to use induction to prove that the greedy choice at each step is best at each step, this is optimal substructure.

In general, greedy algorithms have five components [21]:

- A candidate set, from which a solution is created
- A selection function, which chooses the best candidate to be added to the solution
- A feasibility function, that is used to determine if a candidate can be used to contribute to a solution
- An objective function, which assigns a value to a solution, or a partial solution, and
- A solution function, which will indicate when we have discovered a complete solution

1.4 Polishing technique

In many situations can be very difficult to find quickly a feasible solution of a complex MIP problem but it can be quite easy to find a partial feasible solution in a short time. In this case it can be simple to create a set of partial (or eventually complete) solutions that are differentiated among themselves and not necessarily are close to the optimal solution. This is the common situation in which the polishing technique is applied.

As described by Rotbergh et al. [15], polishing is an algorithm that uses mutation and combination of solutions within a solution pool to generate improved solutions. The polishing algorithm first randomly selects one or more seed solutions from a solution pool for mutation. The selected seed solutions are mutated by fixing a subset of integer variables in the models to the value they take in the seed solution. The remaining variables are then formulated into a sub-MIP problem that is solved by the MIP solver. The solutions generated from this mutation process may then be added to the solution pool. After the one or more iterations of the mutation processes have taken place, the polishing algorithm then selects one or more pluralities of parent solutions from the solution pool to use in generating offspring solutions. The integer variables that agree between one plurality of parent solutions are fixed in the offspring solution. The remaining variables are then formulated into a sub-MIP problem that is solved by the MIP solver. The offspring solutions generated by the combination process may then also be added to the solution pool.

Summarizing, in each phase of the polish algorithm, a new generation of solutions is created from the previous generation and is formed in a series of three steps:

- **Selection:** Pairs of candidate solutions are chosen, typically based on a fitness metric. The intent is that the fittest candidates produce the most descendants;

- **Combination:** Chosen pairs of solutions are combined (in a way that

is meaningful within the problem domain) to produce offspring. The fitness of the offspring may be better or worse than that of its parents;

- **Mutation:** Random changes are introduced into some subset of the offspring (again in a way that is meaningful within the problem domain). The intent of this step is to increase diversity in the population

In most of the cases this approach produces solutions that alternative approaches are unable to find.

1.5 Real scenario

The greedy heuristic algorithm created within this paper will be used in a real energy management system to reduce the overall electricity bill. The system is composed of four principal smart appliance:

- Washing machine
- Cooktop
- Oven
- Refrigerator

In addition, every other appliance that is not "born" smart is equipped with an object called *smart meter* that can measure the consumption of that appliance and transmit it to the central station. Those appliances, unlike the smart one, cannot be scheduled along the day but contribute to the energy consumption.

The central station, that works also as a gateway for all the devices, receives all the informations from smart appliances and smart meters and, using this algorithm, decide the allocation of the four appliance in order to reduce the total electricity cost. Moreover, the GUI installed on the central station lets the user browse all devices connected to the system like power

sources and appliances as well as all the things connected with the smart meter. Each appliance connected can be controlled in detail and has a different GUI. In the case of a washing machine, for example, the user can decide the period of the day in which the washing machine has to be allocated, the type of cycle and all the other parameters that the washing machine lets change, like water temperature, etc.. The user can decide to start the cycle immediately or let that the algorithm decides for him. All these functionalities can be moved to a tablet. Figure 3 shows the results of the algorithm, hence the allocation of the appliances during the day.

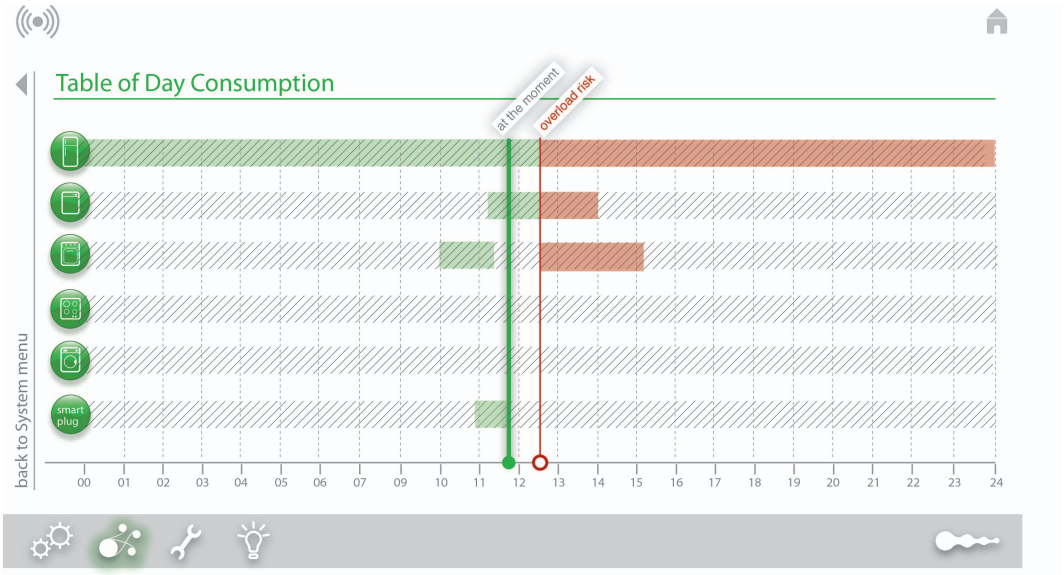


Figure 3: Example of GUI showing the scheduling

2 Our MIP model

Following Sou et al. [18], we model appliance operations as a set of sequential uninterruptible energy phases, each of which uses a given total amount of electric energy. For example, typical washing machine phases are pre-wash,

wash, rinse and spinning. The set of phases is also called the *power profile* of the appliance.

Depending on the appliance, phase duration may vary, as long as the inter-phase delay (e.g., the spinning of the washing machine must start within ten minutes of the rinsing being finished). The total energy given for a phase can be evenly distributed over time, or it may vary. We model the latter case with per-phase bounds on the instantaneous power consumption.

Besides the intrinsic operational constraints, we allow the user to specify preferences for the time interval in which an appliance should run (e.g., start the washing machine between 4pm and 6pm).

Following Barbato et al. [3], we have modeled three classes of energy sources: power grid, domestic renewable energy and accumulators (batteries). The power grid advertises the maximum amount of available energy (*peak power*) for each time instant. Note that this peak power can be different from the actual user contract maximum power. In fact, a common feature in the Smart Grid paradigm is to dynamically advertise (i.e., broadcast) the peak power depending on the grid state, in order to let users adjust their demands for preventing more dangerous power outages. Along with the peak power, also the cost of energy changes with time. For example, in the Italian market it can vary between two values depending on the day time and on the day of the week. More dynamic power grids allow for a finer grain price adjustment (hourly or less).

Domestic renewable energy sources provide free energy but with a limited availability. For example, the performance of a photovoltaic (PV) plant depends on geographical position, weather conditions, and time. Accumulators allow to store energy (from grid or from other sources) when energy price is low, and to use it later when energy price is higher. This feature represents an important degree of freedom as far as optimization is concerned. Our model only deals with batteries, viewed as direct electric energy accumulators; however, it can trivially be extended to other types of energy accumulators (e.g., boilers for thermic energy).

Finally, the optimization goal is to minimize the total energy cost by finding a proper allocation of all appliance phases.

Given two integers a and b, let $[a, b]$ denote the discrete set $\{a, a + 1, \cdots, b\}$. We discretize the scheduling time horizon into m uniform time slots, indexed by $k \in [1, m]$. The phases for each appliance $i \in [1, N]$ are denoted by $j \in [1, n_i]$. To simplify notation, in what follows we write $\forall i, j$ instead of $\forall i \in [1, N], j \in [1, n_i]$, and $\forall k$ instead of $\forall k \in [1, m]$.

In our model, nonnegative continuous variables p_{ij}^k represent the energy assigned to phase j of appliance i during time slot k. The typical unit for p_{ij}^k is Watt (W) per timeslot (energy). With binary variables x_{ij}^k, s_{ij}^k and t_{ij}^k we model the allocation of a time slot k for phase j of appliance i. In particular, $x_{ij}^k = 1$ iff phase j of appliance i is allocated in time slot k. Variable s_{ij}^k jumps from 0 to 1 right after the last time slot of where the phase j of appliance i is allocated, and is defined by the equations:

$$x_{ij}^{k-1} - x_{ij}^k \leq s_{ij}^k \qquad \forall\ i, j, \forall k \in [2, m] \tag{2a}$$

$$s_{ij}^{k-1} \leq s_{ij}^k \qquad \forall\ i, j, \forall k \in [2, m] \tag{2b}$$

Instead, t_{ij}^k is 1 iff there is a inter-phase delay between phase $j - 1$ and j in time slot k, and is defined as:

$$t_{ij}^k = s_{i(j-1)}^k - (x_{ij}^k + s_{ij}^k) \qquad \forall i, k, \forall j \in [2, n_i]$$

Figure 4 illustrates the meaning of the above variables in a simple case of an appliance with two phases: the first phase is allocated between hours 2 and 6, and the second between 14 and 16 (the day being divided into 12 time slots).

Our model also uses nonnegative continuous variables z^k and y^k to represent the amount of total energy sold and bought in each time slot k, respectively. Then, if c^k and g^k denote the input cost of bought and sold electricity during time slot k, respectively, our MIP model calls for:

$$z = \min \sum_{k=1}^{m} \left(c^k y^k - g^k z^k \right) \tag{3}$$

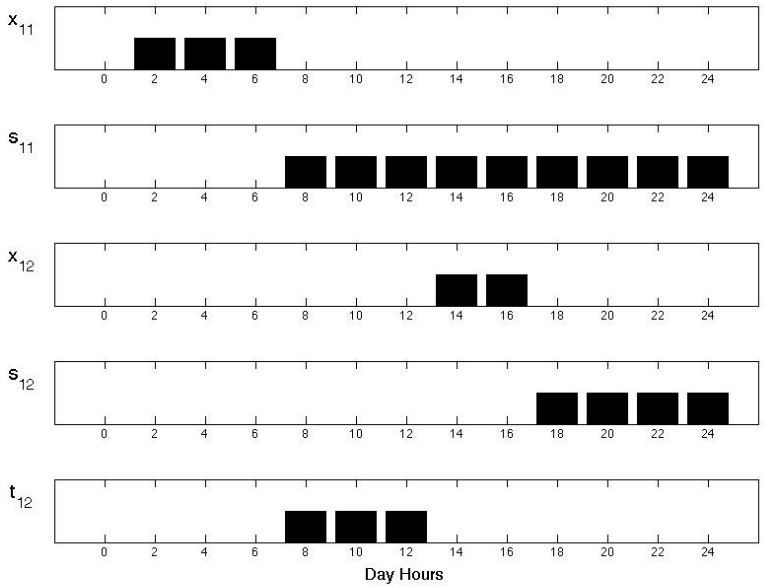


Figure 4: Example of binary variables x_{ij}^k, s_{ij}^k and t_{ij}^k

subjct to the following constraints.
Phase energy: ensures the energy allocated during phase j of appliance i meets the given phase total energy E_{ij}

$$\sum_{k=1}^{m} p_{ij}^{k} = E_{ij} \qquad \forall\ i, j \tag{4}$$

Energy bounds: ensures the energy allocated in phase j for appliance i in any time slot k belongs to the allowed range $[\underline{P}_{ij}, \overline{P}_{ij}]$

$$\underline{P}_{ij}\ x_{ij}^{k} \leq p_{ij}^{k} \leq \overline{P}_{ij}\ x_{ij}^{k} \qquad \forall\ i, j, \forall k \tag{5}$$

Power safety: guarantees that the total energy assigned in time slot k does not exceed the peak power limit

$$y^{k} \leq P_{peak}^{k} \qquad \forall\ k \tag{6}$$

where P_{peak}^{k} is the peak limit of slot k; this constraint can also be used to model additional unscheduled power demands that reduce the available grid energy in time slot k.

Energy phase duration:

$$\underline{T}_{ij} \leq \sum_{k=1}^{m} x_{ij}^{k} \leq \overline{T}_{ij} \qquad \forall\ i, j \tag{7}$$

where $\underline{T}_{ij}$ and $\overline{T}_{ij}$ represent, respectively, the lower and upper bound on the number of time slots to allocate for phase j of appliance i.

Uninterruptible phase: these constraints ensure that all time slots of phase

j are allocated contiguosly (i.e., when an energy phase starts, it must finish without interruptions).

$$x_{ij}^k + s_{ij}^k \leq 1 \qquad \forall\ i, j, k \tag{8}$$

Recall that s_{ij}^k is 0 before the last time slot allocated for phase j and appliance i, and becomes 1 afterwards (2a) until the end (2b). Thus, constraint (8) prevents the variable x_{ij}^k to be 1 after the last-phase time slot.

Sequential processing: previous energy phase must be finished, before a new one starts

$$x_{ij}^k \leq s_{i(j-1)}^k \qquad \forall\ i, k,\ \forall\ j \in [2, n_i] \tag{9}$$

Inter-phase delay duration:

$$\underline{D}_{ij} \leq \sum_{k=1}^{m} t_{ij}^k \leq \overline{D}_{ij} \qquad \forall\ i,\ \forall\ j \in [2, n_i] \tag{10}$$

where $\underline{D}_{ij}$ and $\overline{D}_{ij}$ are the minimum and the maximum number of time slots between phase $j-1$ and j of the appliance i.

User time preferences: disable phase allocation of appliance i in the given time slots

$$x_{ij}^k \leq TP_i^k \qquad \forall\ i, j, k \tag{11}$$

where TP_i^k is equal to zero iff phase j of appliance i cannot be allocated in time slot k.

In order to model batteries behavior we need two extra binary variables w_c^k and w_d^k, where w_c^k is equal to 1 if the battery is charging in time slot k and 0 otherwise, while w_d^k is equal to 1 if the battery is discharging in time slot k and 0 otherwise. Moreover, with the nonnegative continuous variables v_c^k and v_d^k we describe the charge and discharge rates, respectively, that is the amount of energy that is charged/discharged in time slot k. The total

accumulated energy in time slot k is described by the nonnegative continuous variable e^k.
Battery usage constraint: the battery cannot charge and discharge at the same time.

$$w_c^k + w_d^k \leq 1 \qquad \forall\ k \tag{12}$$

Battery charge/discharge rate bounds:

$$v_c^k \ \leq \ v_c^{max} \cdot w_c^k \qquad \forall\ k \tag{13a}$$
$$v_d^k \ \leq \ v_d^{max} \cdot w_d^k \qquad \forall\ k \tag{13b}$$

where v_c^{max}, v_d^{max} denote the max charge and discharge rates respectively.

Battery energy function: this is a linearization of the actual charge/discharge curves

$$e^k = e^{k-1} + \eta_c \cdot v_c^k - \eta_d \cdot v_d^k \qquad \forall\ k \tag{14}$$

where η_c and η_d are, respectively, the charging and discharging efficiency

Battery capacity bounds: used to limit the energy stored in the battery

$$\gamma^{min} \leq \ e^k \ \leq \ \gamma^{max} \qquad \forall\ k \tag{15}$$

where γ^{max} and γ^{min} represent the maximum capacity and the minimum energy safety value (required, for example, by lithium batteries)

Balancing constraint: balance between produced and consumed energy

$$y^k + \pi^k + v_d^k = z^k + \sum_{i=1}^{N} \sum_{j=1}^{n_i} p_{ij}^k + v_c^k \qquad \forall\ k \tag{16}$$

where π^k is the the sum of the newable domestic power sources contribution in time slot k.

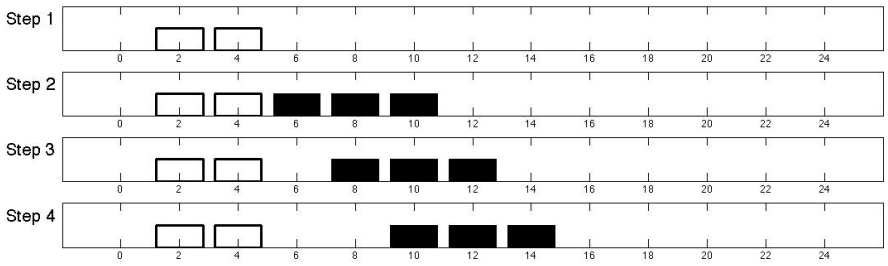
3 A Greedy Algorithm

In this section we describe a heuristic greedy algorithm for finding good feasible solutions of the described problem, that we apply in a multi-start fashion. The algorithm schedules appliances in order of decreasing priority, according to a greedy policy—once an appliance has been scheduled, it is not changed anymore, and all other appliances are allocated on top of the current partial solution. In the first application of the greedy, we use energy requirements as appliance priorities. In the subsequent runs, the priority vector is shifted to generate different solutions. For each appliance, we look for a feasible allocation of its phases according to the following rules.

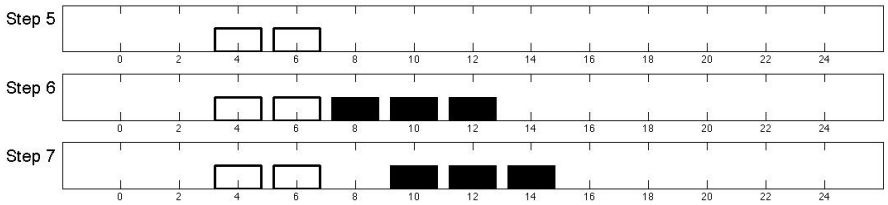
We consider a simplification of problem, where the duration d_{ij} of each phase is the minimum between $\overline{T}_{ij}$ and $\lceil E_{ij}/\underline{P}_{ij} \rceil$, and bounds (5) become $p_{ij}^k = x_{ij}^k E_{ij}/d_{ij}$ for all k.

Accordingly, every phase has a constant duration and a constant energy consumption, and can be scheduled in the time slots interval $[1, m - d_i^{min}]$ where $d_i^{min} = \sum_{j=1}^{n_i} d_{ij}$ represents the minimum duration of a complete appliance power profile (i.e., without phase delays). To be more specific, once a phase has been allocated we look for all possible allocations of the next phase in the range given by $[\underline{D}_{ij}, \overline{D}_{ij}]$, see (10), and we select the most profitable one. Our allocation procedure enforces the user preferences on time slots (11) and three other constraints: power safety (6), uninterruptible phase (8), and sequential processing (9).
Fig. 5 shows an example of scheduling procedure of a single appliance composed of two phases and *User time preferences* $\in [2, 14]$

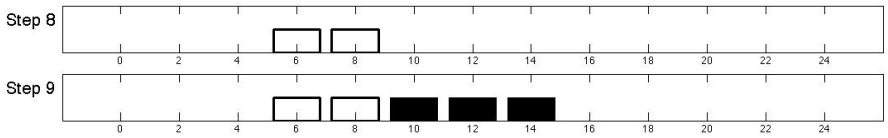
The algorithm consists of four main loops. The first loop iterates over all the N appliances, the second one over the set of the m time slots, the third one over the set of the $n_i = O(m)$ appliance phases, the fourth over the set of m_j delay time slots $[\underline{D}_{ij}, \overline{D}_{ij}]$. Moreover, our allocation procedure requires $O(\overline{T}_j)$ iterations, where T_j is the maximum duration of phase j of appliance i, while the fourth loop requires $O(m_j T_j) = O(m^2)$. However, the latter complexity can be amortized with that of the third loop, because the



(a) Starting slot $k = 2$



(b) Starting slot $k = 3$



(c) Starting slot $k = 4$

Figure 5: Example of the greedy scheduling procedure of a single appliance composed of two phases and *User time preferences* $\in [2, 14]$

following bounds hold:

$$m^2 \geq (\sum_{j=1}^{n_i}(m_j + \overline{T}_j))^2 \geq \sum_{j=1}^{n_i}(m_j + \overline{T}_j)^2 \geq \sum_{j=1}^{n_i}(m_j\overline{T}_j).$$

So the overall time complexity of the proposed greedy algorithm is $O(Nm^3)$, showing the importance of choosing the number m of time slots so as to guarantee a good compromise between computing time and accuracy.

4 Battery optimization

The battery algorithm, like the main one, is a greedy algorithm. The task of the algorithm is to decide in which time slot charge the battery and in which time slot discharge it. Moreover it can decide the quantity of energy that has to be charged or discharged in each time slot.
Let S be the set of energy sources. It comprises both the grid power source and the renewable power source. We will denote by:

- χ_s^k: the cost of energy of power source $s \in S$ in time slot k (that is 0 for renewable power sources and c^k for grid power source)
- ε_s^k: the energy available from power source $s \in S$ in time slot k (that is $\pi^k - z^{k\star}$ and $P_{peak}^k - y^{k\star}$ for renewable and grid power sources respectively)

For simplicity sake we will consider just one battery, being the generalization for multiple batteries straightforward. We will denote by:

- e^k: energy stored in the battery in time slot k
- e_c^k: energy charged in the battery in time slot k
- e_d^k: energy discharged from the battery in time slot k
- w_c^k: charging indicator variable that is 1 if the battery is charging in time slot k, 0 otherwise

Algorithm 1 Greedy algorithm

$\mathbf{p} \leftarrow$ vector of sorted appliances by decreasing $\sum_j E_{ij}$
for all appliances $i \in \mathbf{p}$ **do**
  $z_i^\star \leftarrow +\infty$ ▷ Initializes the best appliance solution
  for all time slots $k \in [1, m]$ **do**
    $z_i \leftarrow +\infty$ ▷ Initializes the incumbent appliance solution
    for all phases $j \in [1, n_i]$ **do**
      $z_{ij}^\star \leftarrow +\infty$ ▷ Initializes the best phase solution
      for all delay slots $k' \in argmax(x_{ij-1}^k = 1) + [\underline{D}_{ij}, \overline{D}_{ij}]$ **do**
        $z_{ij},\ x_{ij} \leftarrow$ feasible allocation of phase j
        if $z_{ij}^\star > z_{ij}$ **then**
          $z_{ij}^\star \leftarrow z_{ij},\ x_{ij}^\star \leftarrow x_{ij}$ ▷ Updates the best phase solution
        end if
      end for
      $z_i \leftarrow z_i + z_{ij}^\star,\ x_i \leftarrow x_i \cup x_{ij}^\star$ ▷ Constructs the incumbent appliance solution
    end for
    if $z_i^\star > z_i$ **then**
      $z_i^\star \leftarrow z_i,\ x_i^\star \leftarrow x_i$ ▷ Updates the best solution
    end if
  end for
  if $z_i^\star = +\infty$ **then**
    return infeasible
  end if
end for
return $[z_i^\star, x_i^\star]$ for all $i \in [1, N]$

- w_d^k: discharging indicator variable that is 1 if the battery is discharging in time slot k, 0 otherwise

We recall that battery charging and discharging operations are mutually exclusive, so if the battery is being charged in a given time slot in the same time slot it cannot be discharged. Moreover the energy stored or drained in a given time cannot exceed the values of v_c^{max} and v_d^{max} respectively.
Now the idea is to find a period along the day where we are buying expensive energy and replace it with the one delivered by the battery which has been charged previously in a period where the energy cost is lower.
The algorithm starts searching the time slot in which the energy cost is max, to determine if there is a chance of removing some expensive energy used in that time slot and replacing it with the energy of the battery. In order to do that we sort time slots by decreasing energy cost c^k and by increasing time slot index for equal cost. Let denote this vector $\boldsymbol{p}$. Accordingly, we sort time time slots by increasing cost χ_s^k and decreasing time slot index k, for equal χ_s^k. Let $\boldsymbol{\pi}$ be this sorted vector, whose cardinality is $|S| * m$.
The algorithm iterates on vector $\boldsymbol{p}$ computing energy bought from the power grid $\rho^k = \sum_{j=1}^{n_i} p_{ij}^k$, with unit cost c^k. If the battery is not already charging in time slot k ($w_c^k = 0$) we can discharge the energy accumulated in previous time slots to partially of completely fullfill the energy demand ρ^k. To do this the algorithm scans the sorted vector $\boldsymbol{\pi}$ looking for a power source s whose unit cost χ_s^h in time slot $h < k$ is less than c^k. If it can find a cheaper power source s in time slot h, and discharging flag w_d^h is zero, battery is charged by the minimum value between demand ρ^k, battery charging rate $v_c^{max} - e_c^h$, battery discharging rate $v_d^{max} - e_d^k$, available energy ε_s^h and available battery capacity $\gamma^{max} - e^l$ for all $l \in [j, k]$. It then updates values of ρ^k, ε_s^k and e^l and continues iterating over $\boldsymbol{\pi}$ until either $\rho^k = 0$ or $e^k = \gamma^{max}$.

The algorithm consists of three main loops. The first loop iterates over all the m time slots and the second over all the m time slots of each power source. The two inner consecutive loops iterate along an interval of max

length m, hence the computational complexity results:

$$O(2m \sum_{i=0}^{m} i) = O(m \cdot \frac{m(m+1)}{2}) = O(m^3)$$

5 Computational results

In our tests we considered a time horizon of 24 hours, subdivided in 96 time slots of 15 minutes each. Experiments were grouped into four sets according to two main parameters. The first parameter is the "flexibility" of the *user time preference* constraint. We considered two level of flexibility, namely: *high flexibility* (HF), meaning that the appliance can be scheduled at any time during the day, and *medium flexibility* (MF), meaning that appliances can run inside a 12-hour randomly-generated time window within the day. The second parameter is electricity cost: it can vary either every two hours (BC), or every time slot (TC). For each of the four resulting sets, namely HFBC, HFTC, MFBC, and MFTC, we considered 10, 20, or 30 appliances, respectively, and solved 5 random instances for each of the 12 combinations—60 instances in total.

A constant price of the sold photovoltaic energy was considered, equal to half of the minimum cost of the bought energy. All the other model parameters are taken from uniform random distributions: $c_k \in [2, 4]$, $j \in [2, 5]$, $E_{ij} \in [400, 800]$, $\underline{P}_{ij} \in [50, 80]$, $\overline{P}_{ij} \in [400, 800]$, $\underline{T}_{ij} \in [1, 2]$, $\overline{T}_{ij} \in [3, 5]$, $\underline{D}_{ij} = 1$, $\overline{D}_{ij} \in [4, 6]$, and $P^k_{peak} \in [2400, 2600]$.

We considered a single renewable photovoltaic power source, whose provided energy is sampled from a Gaussian distribution $\mathcal{N}(\mu, \sigma)$ with mean $\mu = 52$ (1pm, the period of maximum production at the latitude of Italy), standard deviation $\sigma = 10$, and maximum value of 1250W per time slot. The considered battery has a capacity $\gamma^{max} = 500$ Watt per time slot and charge/discharge rates $v_c^{max} = v_d^{max} = 50$ Watt per time slot, with efficiencies $\eta_c = \eta_d = 1$.

Algorithm 2 Battery algorithm

$\boldsymbol{p} \leftarrow$ vector of time slots indices sorted by decreasing bought energy cost (c^k)

$\boldsymbol{\pi} \leftarrow$ set of time slots and power indices sorted by increasing cost χ_s^k and decreasing time slot index k for all $s \in S$

for all time slots $k \in \mathbf{p}$ **do** ▷ Discharging time slot

  $\rho^k \leftarrow \sum_{i=1}^{N} \sum_{j=1}^{n_i} p_{ij}^k$ ▷ Initializes the value of energy to realloc

  for all pairs of timeslot and source indices $\langle h, s \rangle \in \boldsymbol{\pi}$ **do**

    $\mu \leftarrow min(\rho^k, \varepsilon^h)$

    if $h < k$ and $\chi_s^h < c^k$ and $w_c^k = 0$ and $w_d^h = 0$ **then**

      $\mu \leftarrow min(\mu, v_c^{max} - e_c^h)$ ▷ Check the charging bound

      $\mu \leftarrow min(\mu, v_d^{max} - e_d^k)$ ▷ Check the discharging bound

      for all time slots $l \in [h, k]$ **do**

        $\mu \leftarrow min(\mu, \gamma^{max} - e^l)$ ▷ Check the capacity bound

      end for

      for all time slots $l \in [h, k]$ **do** ▷ Updating battery status

        $e^l \leftarrow e^l + \mu$

      end for

      $e_d^k \leftarrow e_d^k + \mu$

      $w_d^k \leftarrow 1$

      $e_c^h \leftarrow e_c^h + \mu$

      $w_c^h \leftarrow 1$

      $\varepsilon_s^h \leftarrow \varepsilon_s^h - \mu$ ▷ Updating available energy for source s

      $\varepsilon_s^k \leftarrow \varepsilon_s^k - \mu$

      $\rho^k \leftarrow \rho^k - \mu$

      break if $\rho^k = 0$

    end if

  end for

end for

All the simulations ran un computing cluster *Blade* of the University of Padova (more explanation as well as a simple guide in section 6), in single-thread mode.

We compared seven different solution approaches:

- `Greedy-alone`: our stand alone greedy algorithm without multistart enhancement;
- `Greedy`: our greedy algorithm applied N times by starting from the N possible shifts of the initial priority vector, taking the best solution found and storing the others;
- `Battery`: our battery algorithm applied after `Greedy-alone`;
- `Greedy+Battery`: our battery algorithm applied after `Greedy`;
- `Cplex`: the state of the art IBM ILOG CPLEX MIP solver used as a black-box, with its default setting, stopped as soon as the first feasible solution is found;
- `Cplex+Polish`: CPLEX's polishing refining heuristic [15] applied right after the root node and for a total of 10 nodes, when starting from the feasible solution found by the previous `Cplex` algorithm;
- `Greedy+Polish`: our proposed MIP-and-refine scheme, i.e., the previous `Cplex+Polish` algorithm but starting from the list of all feasible solutions found by `Greedy`.

Table 2 entries represent percentage cost increase with respect to the `Greedy-alone` algorithm per CPU second, so it is the ratio between the percentage cost improvement of a method (respect to the `Greedy-alone`) and the difference between the execution time of that method and the execution time of `Greedy-alone`. In some sense that value describes the efficiency of those methods.

	`Greedy` efficiency	`Battery` efficiency
Set	% Impr/sec.	% Impr/sec.
HFBC 10	0.00	0.00
HFBC 20	0.24	231.22
HFBC 30	0.10	74.51
HFTC 10	0.00	0.00
HFTC 20	0.76	326.17
HFTC 30	0.09	68.84
MFBC 10	0.00	0.00
MFBC 20	2.18	326.94
MFBC 30	0.23	72.45
MFTC 10	0.00	0.00
MFTC 20	0.83	234.87
MFTC 30	0.28	59.70

Table 2: Efficiency of methods `Greedy` and `Battery` compared to `Greedy-alone`. Percentage cost improve (% Impr) per CPU second.

It is evident that in terms of efficiency `Battery` is fairly better and can significantly improve the solution within a short computing time.

Moreover, Table 2 shows an apparently strange behaviour: the efficiency of the sets with 20 appliances is always greater than the one with 30 appliances. The reason of this fact must be sought on the characteristics of the photovoltaic power source and, in particular, on the amount of energy that it can deliver; in our case $\sum_{k=1}^{m} \pi^k = 31253$. We can notice in Tables 17 18 that this value exceeds the total amount consumption of 10 appliances whereas is a little bit lower than the case of 20 apps and quite lower than the case of 30. It becomes obvious that the positioning of the apps is much more important in the case where the total energies involved are similar. In fact, in first case all the apps are allocated within the photovoltaic production area whereas in the third case the photovoltaic energy is almost completely used by a bunch of apps and the remaining ones are allocated somewhere else where the cost saving becomes difficult, reducing the efficiency (Fig.6, Fig.7, Fig.8).

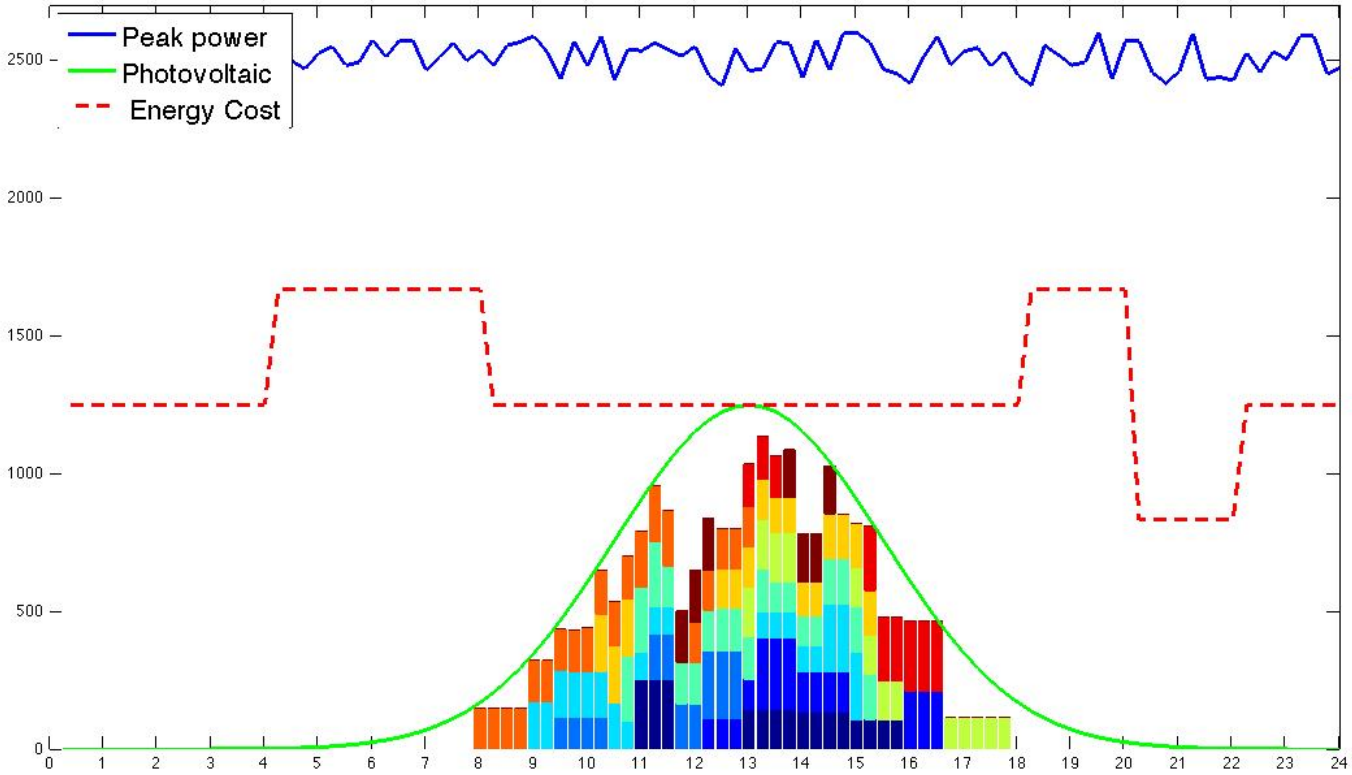


Figure 6: Example of 10 appliances scheduling on HFBC using `Greedy-alone`. Each color represents a different appliance.

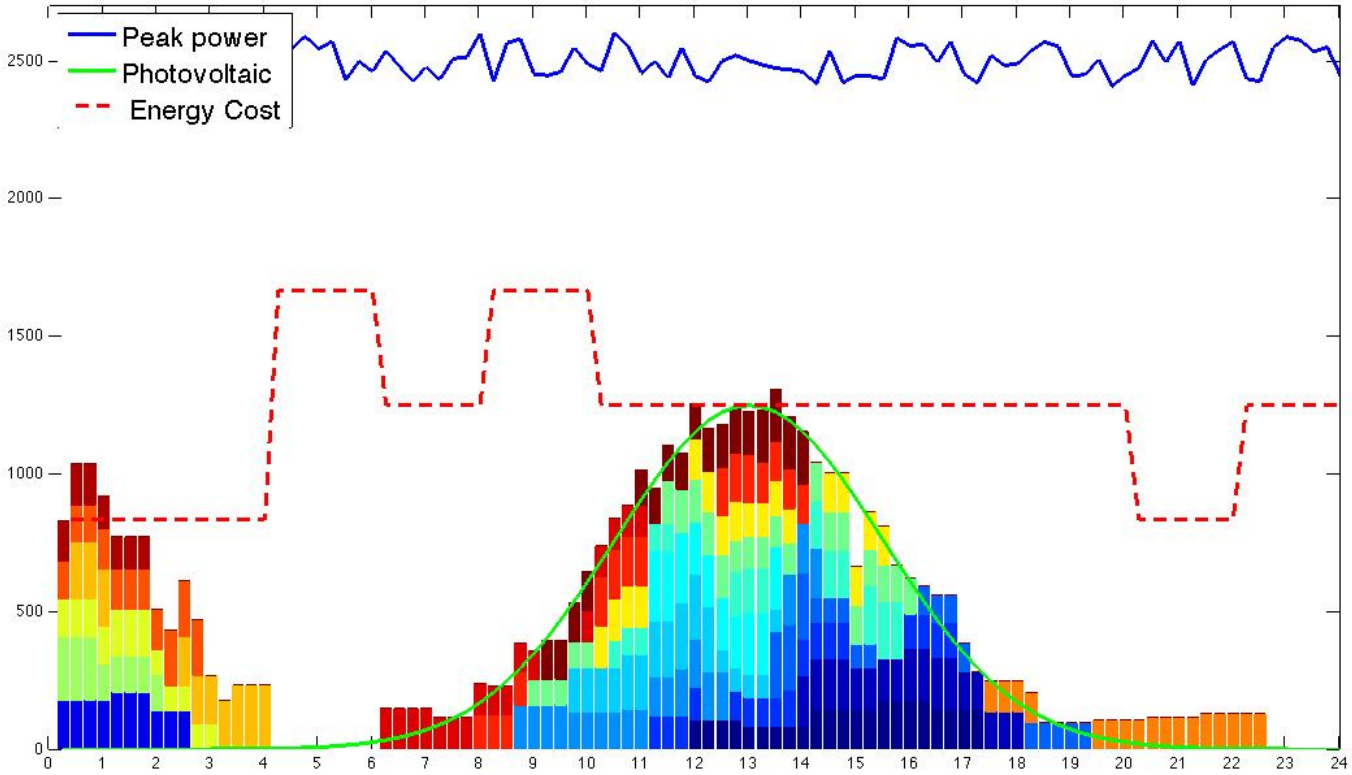


Figure 7: Example of 20 appliances scheduling on HFBC using `Greedy-alone`. Each color represents a different appliance.

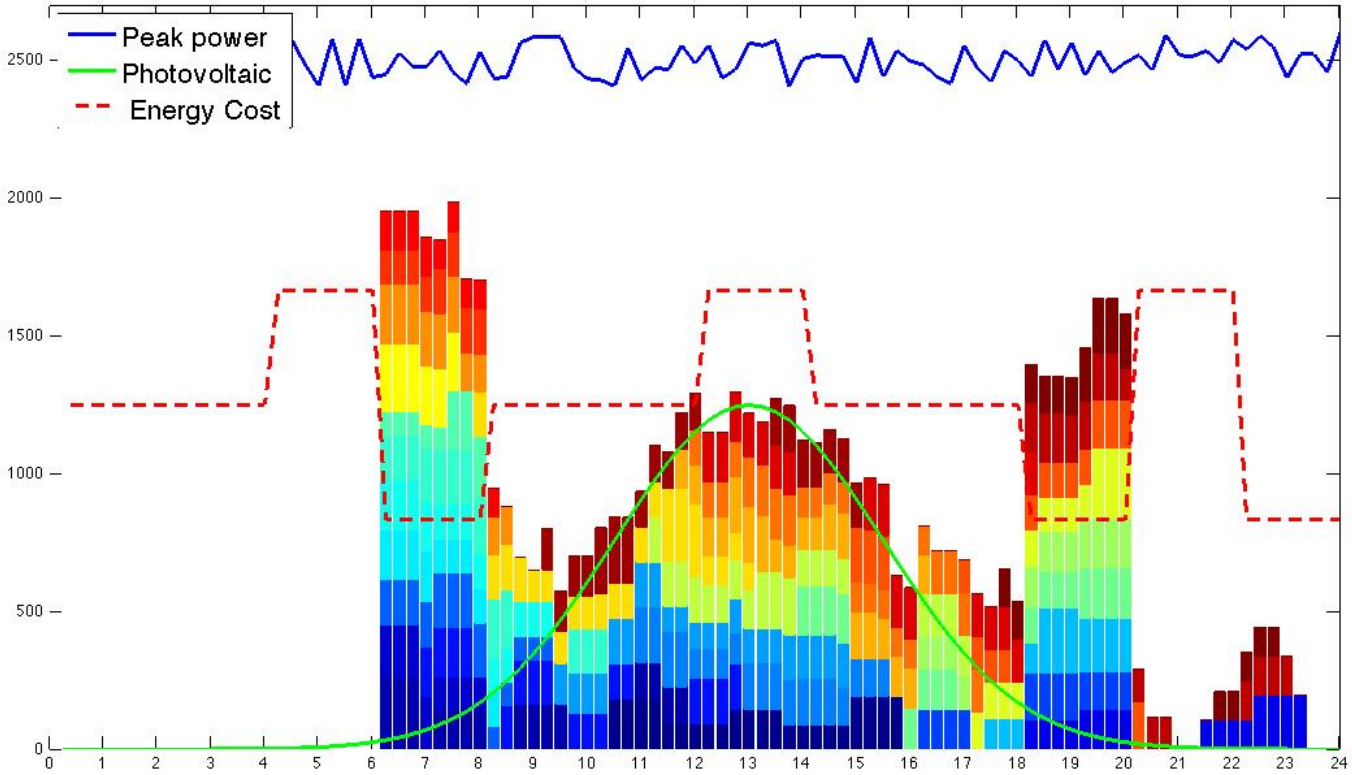


Figure 8: Example of 30 appliances scheduling on HFBC using `Greedy-alone`. Each color represents a different appliance.

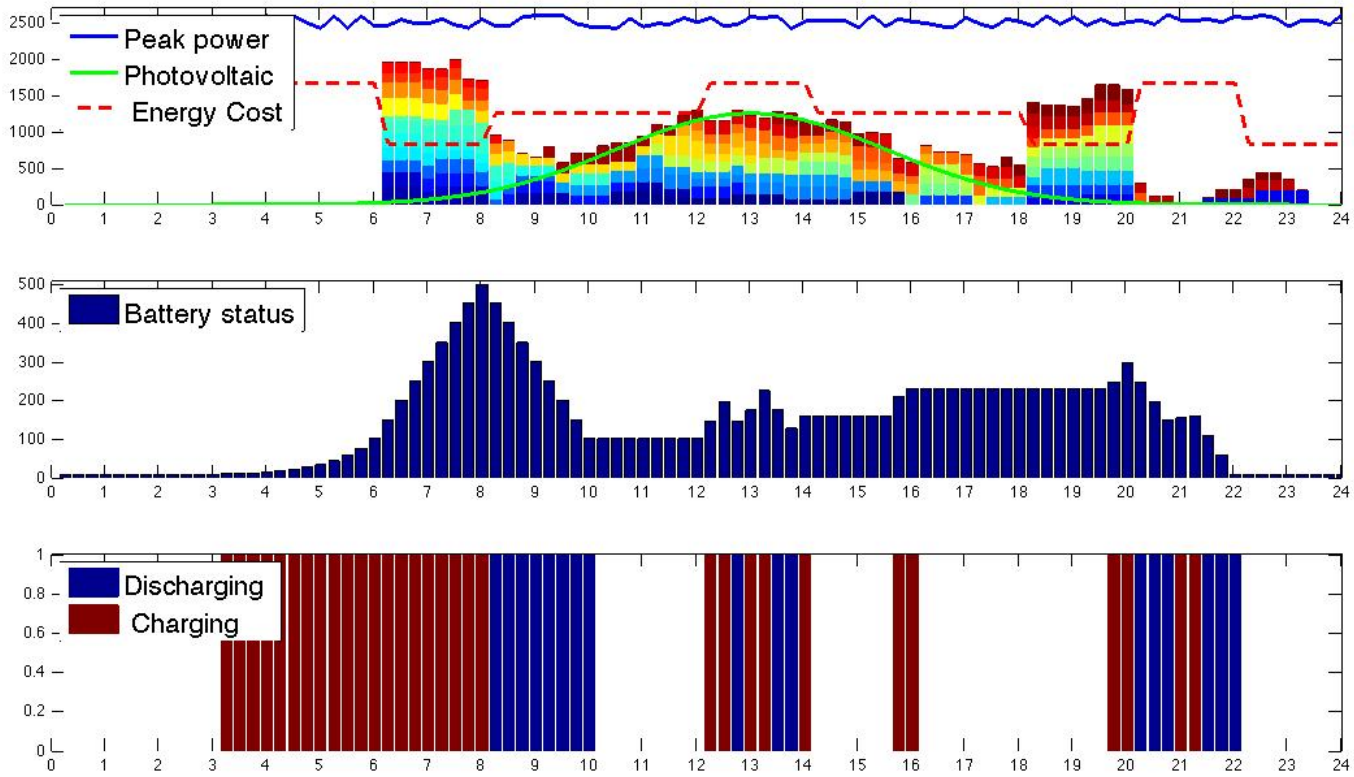


Figure 9: Example of 30 appliances scheduling on HFBC using `Greedy+Battery`. Each color represents a different appliance.

	Greedy			Battery			Greedy+Battery		
Set	% Incr	% STD	Time	% Incr	% STD	Time	% Incr	% STD	Time
HFBC 10	0.0 ^{5}	0.0	0.5	0.0 ^{5}	0.0	0.1	0.0 ^{5}	0.0	0.6
HFBC 20	-0.5	0.5	2.4	-4.6	1.6	0.1	-5.1	1.3	2.8
HFBC 30	-0.6	0.5	6.4	-2.2	0.4	0.2	-2.7	0.5	7.3
HFTC 10	0.0 ^{5}	0.0	0.4	0.0 ^{5}	0.0	0.1	0.0 ^{5}	0.0	0.5
HFTC 20	-1.7	1.2	2.3	-6.5	1.7	0.1	-8.1	1.1	2.7
HFTC 30	-0.6	0.3	6.6	-2.3	0.3	0.3	-2.8	0.5	7.6
MFBC 10	0.0 ^{5}	0.0	0.3	0.0 ^{5}	0.0	0.0	0.0 ^{5}	0.0	0.4
MFBC 20	-3.0	2.2	1.4	-5.9	3.1	0.1	-8.8	5.5	1.8
MFBC 30	-0.8	0.5	3.7	-1.3	0.3	0.1	-2.1	0.4	4.2
MFTC 10	0.0 ^{5}	0.0	0.3	0.0 ^{5}	0.0	0.0	0.0 ^{5}	0.0	0.4
MFTC 20	-1.2	1.2	1.5	-4.7	0.7	0.1	-5.9	1.6	1.9
MFTC 30	-1.0	0.3	3.7	-2.1	0.3	0.2	-3.1	0.5	4.8

Table 3: Percentage cost increase (% Incr) with respect to the `Greedy-alone` algorithm alone, and computing time (in CPU sec.s); a negative increase corresponds to a better solution w.r.t. `Greedy-alone`. The reported values are arithmetic means over 5 random instances. Column % STD gives the percentage standard deviation of cost increase. Computing times for `Greedy-alone` are just negligible. Superscript ^{k} means proven optimality for k out of 5 instances.

	Cplex			Cplex+Polish			Greedy+Polish		
Set	% Incr	% STD	Time	% Incr	% STD	Time	% Incr	% STD	Time
HFBC 10	252.9	193.4	56.8	72.7 ^{2}	95.6	94.9	0.0 ^{5}	0.0	0.6
HFBC 20	50.2	80.4	3,054.4	13.4	41.2	3,223.2	-9.4	4.8	262.0
HFBC 30	1.8	4.7	9,585.8	-5.1	3.4	9,966.5	-6.1 ^{1}	3.2	310.2
HFTC 10	94.0	129.2	49.0	47.0 ^{4}	94.1	83.4	0.0 ^{5}	0.0	0.6
HFTC 20	58.0	49.9	2,213.0	-1.4	21.3	2,387.4	-22.2 ^{1}	1.8	154.7
HFTC 30	-12.3	1.4	46,439.1	-19.0	2.6	46,904.7	-21.2 ^{1}	2.5	617.7
MFBC 10	157.3 ^{1}	146.7	5.2	38.8 ^{2}	51.4	15.4	0.0 ^{5}	0.0	1.3
MFBC 20	160.1	45.9	35.2	24.4	24.2	90.0	-13.8	7.8	100.8
MFBC 30	44.4	11.5	65.1	0.7	8.3	164.5	-11.7 ^{1}	4.5	67.4
MFTC 10	159.9	89.5	5.6	33.5 ^{1}	24.8	18.8	0.0 ^{5}	0.0	1.5
MFTC 20	54.8	21.8	52.5	-8.4	13.6	114.2	-23.5 ^{1}	2.6	54.6
MFTC 30	16.2	24.8	117.8	-14.0	6.1	232.7	-22.0	1.6	112.0

Table 4: Percentage cost increase (% Incr) with respect to the `Greedy-alone` algorithm alone, and computing time (in CPU sec.s); a negative increase corresponds to a better solution w.r.t. `Greedy-alone`. The reported values are arithmetic means over 5 random instances. Column % STD gives the percentage standard deviation of cost increase. Computing times for `Greedy-alone` are just negligible. Superscript ^{k} means proven optimality for k out of 5 instances.

According to Table 3 and Table 4, `Greedy+Polish` outperforms its competitors by a large amount.

As expected, `Greedy-alone` is always able to provide feasible solutions in very short computing times. In spite of its greedy nature, the solution quality is fair in many cases, in particular in the easiest scenarios where the greedy solution often turns out to be optimal. Nevertheless, for more difficult scenarios there is room for important improvements—also because of the contribution of the batteries that is exploited by the MIP model but not by the `Greedy-alone`.

As to `Cplex`, it has a great difficulty even in finding its first feasible solution—a task that takes a huge amount of time in the difficult cases. Significantly improved solutions are found by `Cplex+Polish`, thus confirming the effectiveness of this heuristic. However, the full power of MIP refinement is only exploited when `Greedy+Polish` comes into play. This is due to two main factors: the speed of the greedy, and the fact that several diversified solutions are passed to the polishing method.

Of course, we cannot claim that `Greedy+Polish` would outperform more sophisticated heuristic approaches from the literature on similar problems—for that, much more extensive computational comparisons would be needed. However, we believe our computational results support the message of the present paper—sound matheuristics can be built around a simple greedy and an off-the-shelf MIP refinement procedure.

A more sophisticated heuristic has been created adding the battery optimization. It guarantees a significant improving of the solution within a negligible period of time but can not achieve the results of `Greedy+Polish` and it costs us creating it nearly the same time of creating the `Greedy-alone`. This fact proves again that to produce a very complicated heuristic can be a real waste of time if `Greedy+Polish` can be used.

In Fig. 10 is represented an example of the battery status behavior as well as the appliances scheduling using `Greedy+Battery`, while in Fig. 11 is represented the same instance solved by `Greedy+Polish`. It can be noticed

that `Greedy+Polish` can benefit of the much more complex mathematical model of the MIP adapting the power profile of the appliances to fit perfectly inside the area of where there is photovoltaic energy or where the power-grid-energy cost is lower. In section 6 are reported other examples of different instances.

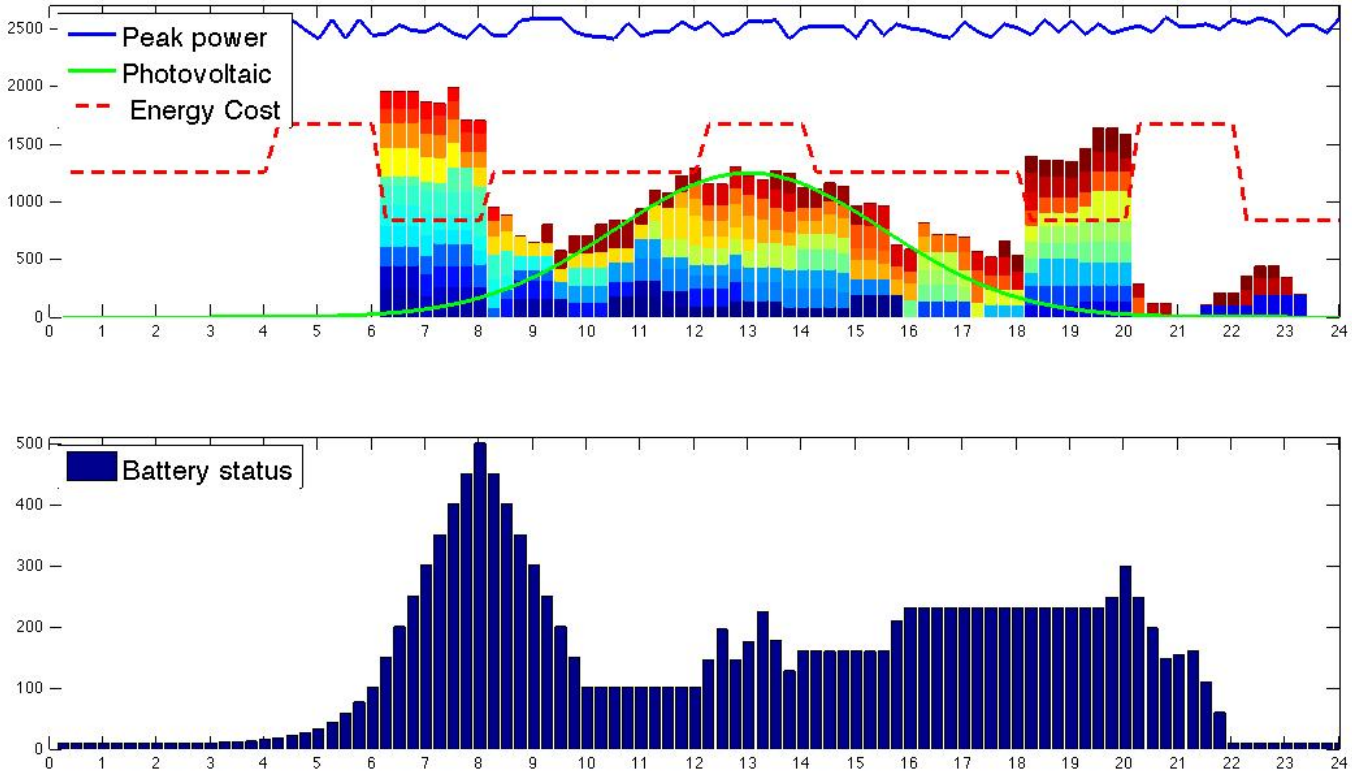


Figure 10: Example of 30 appliances scheduling and battery behavior on HFBC using `Greedy+Battery`. Each color represents a different appliance.

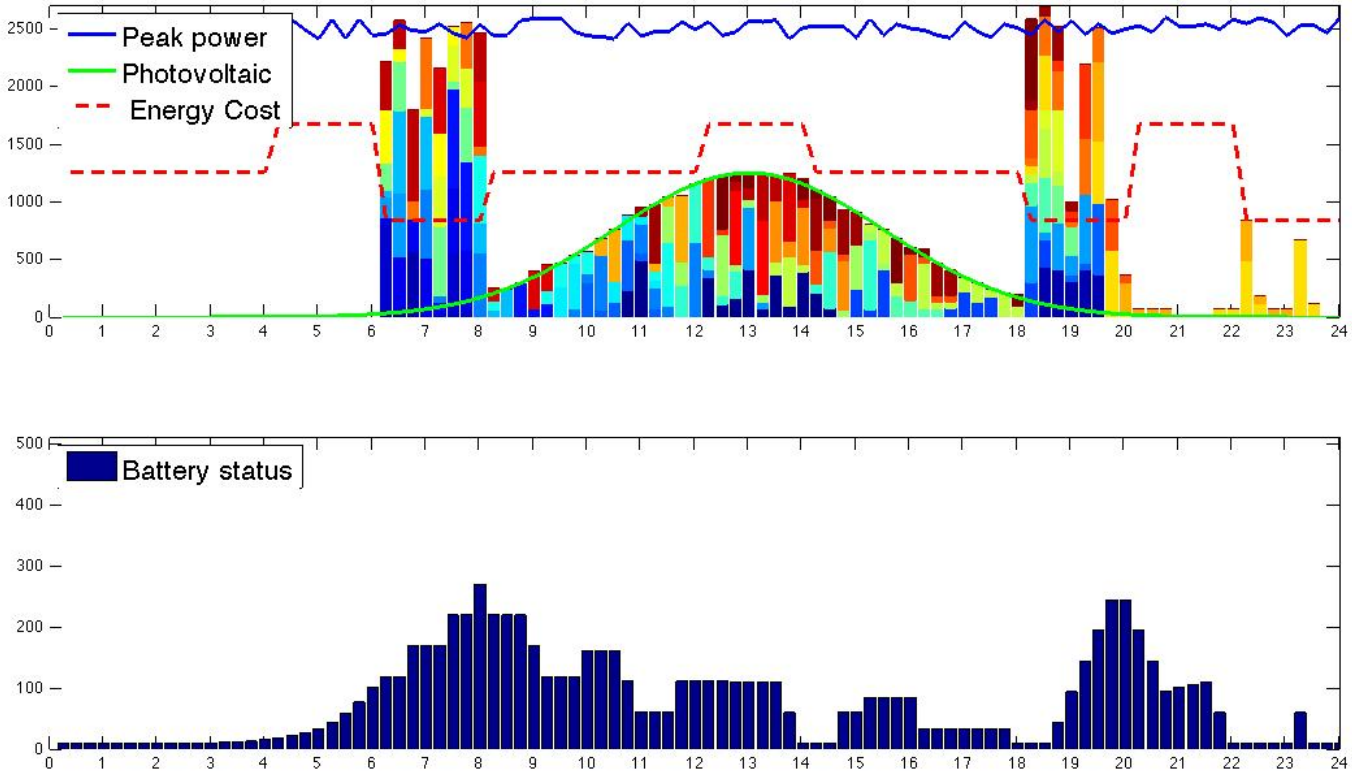


Figure 11: Example of 30 appliances scheduling and battery behavior on HFBC using `Greedy+Polish`. Each color represents a different appliance.

6 Conclusions

A simple MIP-and-refine matheuristic framework has been addressed, where a greedy heuristic is used to trigger a general purpose MIP refinement procedure. Moreover a more sophisticated heuristic is presented. Computational results on a smart-grid energy management problem have been presented, showing that the method and the heuristic produces sound results.

The approach is based on two ingredients: an initial heuristic, and a MIP model. The heuristic needs not to be very effective, as its role is just to initialize a pool of feasible solutions–the more diversified the better. The MIP model itself needs not to be very sophisticated, as it is automatically resized by the general purpose MIP refinement procedure. Nevertheless, the combination of the two can be much more effective than the sum of its parts, in the sense that the two modules work in a highly synergic way and can produce outcomes whose solution quality can only be matched by sophisticated ad-hoc heuristics. A little demonstration of this fact is that the battery-pumped heuristic, even obtaining good results, can not achieve the effectiveness of our approach but is much more complicated than the stand alone heuristic.

Future research on the topic will hopefully confirm the viability of the approach on other classes of very difficult problems.

References

[1] Smart Grid Working Group (2003-06). Challenge and opportunity: Charting a new energy future, appendix a: Working group reports. *Energy Future Coalition*, 2008.

[2] A. Agnetis, G. Dellino, P. Detti, G. Innocenti, G. De Pascale, and A. Vicino. Appliance operation scheduling for electricity consumption optimization. In *50th IEEE Conference on Decision and Control and Eu-*

ropean Control Conference (CDC-ECC), Orlando, FL, USA, December 12-15, pages 5899–5904, 2011.

[3] A. Barbato, A. Capone, M. Carello, Delfanti, M. Merlo, and A. Zaminga. House energy demand optimization in single and multi-user scenarios. In *2nd IEEE International Conference on Smart Grid Communications*, 2011.

[4] A Barbato and G. Carpentieri. Model and algorithms for the real time management of residential electricity demand. In *IEEE International Conference and Exhibition, ENERGYCON '12*, 2012.

[5] Samujjwal Bhandari. Design and analysis of algorithms. *Tribhuvan University, Kirtipur, Kathmandu, Nepal.*, 2010.

[6] E. Borioli, E. Ciapessoni, D. Cirio, and E. Gaglioti. Applications of neural networks and decision trees to energy management system functions. *IEEE, Power System Development Dept. ERSE Milano, Italy*, 2009.

[7] G. Carpentieri, G. Carello, and E. Amaldi. Optimization models for residential energy load management. In *Atti delle Giornate AIRO, Vietri*, 2012.

[8] C.M. Colson, M.H. Nehrir, and C. Wang. Ant colony optimization for microgrid multi-objective power management. *IEEE, Electrical and Computer Engineering Department, Montana State University, Bozeman, MT, USA*, 2009.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[10] X. Fang, S. Misra, G. Xue, and D. Yang. Smart grid – the new and improved power grid: A survey. *IEEE Communications Surveys and Tutorials (COMST)*, 14:944–980, 2012.

[11] M. Fardadi, A. Selk Ghafari, and S.K. Hannani. Pid neural network control of sut building energy management system. *IEEE, International Conference on Advanced Intelligent Mechatronics Monterey, California, USA*, 2005.

[12] M. Fischetti and M. Monaci. Proximity search for 0-1 mixed-integer convex programming. *Technical Report, DEI, University of Padova (in preparation)*, 2012.

[13] Dae Man Han and Jae Hyun Lim. Design and implementation of smart home energy management systems based on zigbee. *IEEE, Kongju National University, Republic of Korea*, 2010.

[14] S. Hatami and M. Pedram. Minimizing the electricity bill of cooperative users under a quasi-dynamic pricing model. *SmartGridComm10*, pages 421–426, 2010.

[15] E. Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.

[16] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & sons, 1998.

[17] Enrique Sierra, Alejandro Hossian, Paola Britos, Dario Rodriguez, and Ramn Garca-Martnez. Fuzzy control for improving energy management within indoor building environments. *Electrotecnic Dept. School of Engineering. National University of Comahue*.

[18] K. C. Sou, J. Weimer, H. Sandberg, and K. H. Johansson. Scheduling smart home appliances using mixed integer linear programming. In *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC), Orlando, FL, USA, December 12-15*, pages 5144–5149. IEEE, 2011.

[19] K.F. Tsang, W.C. Lee, K.L. Lam, H.Y. Tung, and Kai Xuan. An integrated zigbee automation system: An energy saving solution. *IEEE, Department of Electronic Engineering, City University of Hong Kong*, 2007.

[20] H.Y. Tung, K. F. Tsang, and K. L. Lam. Energy management with zigbee sensor network. *IEEE, Department of Electronic Engineering, City University of Hong Kong*, 2008.

[21] Wikipedia. Wikipedia, the free encyclopedia, 2012.

[22] Z. Yiming, Y. Xianglong, G. Xishan, Z. Mingang, and W. Liren. A design of greenhouse monitoring and control system based on zigbee wireless sensor network. *IEEE, Department of Bio-System Engineering, Zhejiang University, China*, 2007.

[23] A. Zanette. Cut-and-pivot: a class of exact solution methods for integer programming. *Master's Thesis, DEI, University of Padova*, 2005.

[24] D. Zhang, L.G. Papageorgiou, N.J. Samsatli, and N. Shah. Optimal scheduling of smart homes energy consumption with microgrid. In *ENERGY 2011, The First International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, pages 70–75, 2011.

[25] Peng Zhao, Siddharth Suryanarayanan, and M. Godoy Simes. An energy management system for building structures using a multi-agent decision-making control methodology. *IEEE, Colorado School of Mines Golden, Colorado, USA*, 2010.

[26] ZigBee. *http://www.zigbee.org/*.

Appendix A: ILOG CPLEX

Cplex is the core solver engine of a family of optimization products of ILOG. It includes [23]:

- **simplex optimizers:** to solve linear and quadratic programs using primal and dual simplex algorithms, a presolver and a network optimizer;

- **barrier optimizer:** an alternative to the simplex method to solve linear and quadratic programs;

- **mixed integer optimizer:** to solve problems with mixed-integer variables (general or binary) and linear or quadratic objective function, using state-of-the art algorithms and techniques, including cuts, heuristics, polishing, and a variety of branching and node selection strategies

ILOG CPLEX offers C, C++, Java, and .NET libraries that solve linear programming (LP) and related problems. Specifically, it solves linearly or quadratically constrained optimization problems where the objective to be optimized can be expressed as a linear function or a convex quadratic function. The variables in the model may be declared as continuous or further constrained to take only integer values.

ILOG CPLEX comes in three forms:

- The **ILOG CPLEX Interactive Optimizer** is an executable program that can read a problem interactively or from files in certain standard formats, solve the problem, and deliver the solution interactively or into text files.

- **ILOG Concert Technology** is a set of libraries offering an API that includes modeling facilities to allow a programmer to embed ILOG CPLEX optimizers in C++, Java, or .NET applications.

- **The ILOG CPLEX Callable Library** is a C library that allows the programmer to embed ILOG CPLEX optimizers in applications written in C, Visual Basic, Fortran or any other language that can call C functions.

Since the mathematical model presented in this thesis is a MIP problem, we only use the *mixed integer optimizer* of CPLEX including all the new features like the polishing technique. More explanations on how we used CPLEX are described in 6.

Appendix B: Blade

The computing cluster *Blade* of the Department of Information Engineering (DEI) of the University of Padova is equipped with 28 Intel Quad Core® E5450 at 3 GHz and 6 MB of cache, for a total of 112 core. The overall RAM memory is 224 GByte. The machine uses a batch system that allow to run the simulations, which are automatically assigned to the free resources.

The management program is *Oracle Grid Engine* (previously known as *Sun Grid Engine*), that is an open source batch-queuing system, developed and supported by Sun Microsystems.

The batch has to be submitted from a computer connected at the DEI network or can be accessed via ssh from a any computer all over the world (with Username e Password of the DEI account).

According with *Oracle Grid Engine* guidelines, each batch has to be submitted using the command *qsub* followed by the proper options. Below an example of the batch to be submitted.

Algorithm 3 Blade submitting batch

```
#!/bin/bash
#$ -P 40_Studenti
#$ -cwd
#$ -e Blade/Errors/log.err
#$ -o Blade/Outputs/log.out
#$ -m ea
./myScript
```

where $-P$ 40_*Studenti* indicates the group membership of the submit-

ter, $-cwd$ states that it will be executed in the Current Working Directory, $-e\ Blade/Errors/log.err$ and $-o\ Blade/Outputs/log.out$ where the logs are saved. Moreover, the line $-m\ ea$ specifies that the system will send a mail to the submitter account if the batch is ended (e) or aborted (a). In the successive lines can be used every batch command, in particular the launch of a script.

In our case, the script has to contain the commands to lunch CPLEX as an *Interactive Optimizer*. It is very simple and may be written as in 4.

Algorithm 4 Cplex executable script

```
#!/bin/bash
cplex < myCplexCommands
```

Where *cplex* lunch the CPLEX application and "<" passes to CPLEX the file *myCplexCommands* that contains the instructions for the cplex execution.

An example of file to be passed to CPLEX and used within this thesis in certain circumstances is presented in 5.

Algorithm 5 Cplex commands file

```
read LPModel.txt lp
read CplexStart.txt mst
set mip cuts all -1
set mip poli node 1
set mip limits nodes 10
set thread 1
optimize
write CplexSolution.xml sol
y
quit
```

For more explanations look at CPLEX *User's Manual* available online at
`http://www.ibm.com/software/integration/optimization/cplex-optimizer/`

Appendix C: Detailed tables

	Greedy-alone		Greedy		Battery		Greedy+Battery	
Instance	Obj	Time	Obj	Time	Obj	Time	Obj	Time
HFBC_10_1	-11733	0.04	-11733	0.42	-11733	0.05	-11733	0.52
HFBC_10_2	-9602	0.05	-9602	0.48	-9602	0.06	-9602	0.58
HFBC_10_3	-12442	0.04	-12442	0.39	-12442	0.05	-12442	0.49
HFBC_10_4	-8327	0.05	-8327	0.49	-8327	0.06	-8327	0.59
HFBC_10_5	-10369	0.05	-10369	0.47	-10369	0.06	-10369	0.57
HFBC_20_1	26970	0.12	26970	2.39	25265	0.14	25265	2.79
HFBC_20_2	27689	0.12	27471	2.37	26228	0.14	26127	2.77
HFBC_20_3	26862	0.12	26498	2.48	26012	0.14	25624	2.88
HFBC_20_4	23963	0.11	23963	2.25	22515	0.13	22515	2.65
HFBC_20_5	25685	0.12	25529	2.42	25090	0.14	24944	2.82
HFBC_30_1	56128	0.20	55660	5.99	55054	0.23	54599	6.89
HFBC_30_2	78740	0.22	78740	6.62	77067	0.25	77067	7.52
HFBC_30_3	64039	0.21	63679	6.29	62862	0.24	62687	7.19
HFBC_30_4	64717	0.22	63826	6.49	63268	0.25	62599	7.39
HFBC_30_5	70356	0.22	70175	6.61	68202	0.25	68153	7.51

Table 5: Values of the objective function and time elapsed of the set 1 with *High flexibility* and *Bihour cost*

	Cplex		Cplex+Polish			Greedy+Polish		
Instance	Obj	Time	Obj	Time	Tot time	Obj	Time	Tot time
HFBC_10_1	7152	44.59	-11733	19.01	63.60	-11733	0.18	0.60
HFBC_10_2	45277	48.72	15106	48.16	96.88	-9602	0.18	0.66
HFBC_10_3	-11122	36.37	-12443	16.51	52.88	-12442	0.13	0.52
HFBC_10_4	5434	69.64	-4903	68.31	137.95	-8327	0.15	0.64
HFBC_10_5	26547	84.51	-3647	38.44	122.95	-10369	0.14	0.61
HFBC_20_1	27851	7410.00	24029	242.88	7652.88	23641	263.29	265.68
HFBC_20_2	86075	3099.29	54168	128.00	3227.29	23064	167.62	169.99
HFBC_20_3	29957	325.76	25554	183.00	508.76	25472	297.85	300.33
HFBC_20_4	27229	2915.86	21708	137.29	3053.15	21676	205.16	207.41
HFBC_20_5	28745	1521.00	24805	153.05	1674.05	24800	363.95	366.37
HFBC_30_1	54892	5826.20	53292	266.84	6093.04	52139	279.48	285.47
HFBC_30_2	74735	9814.37	69860	528.59	10342.96	69452	471.53	478.15
HFBC_30_3	65938	5118.16	63202	284.90	5403.06	62599	121.17	127.46
HFBC_30_4	69139	8259.00	62591	392.03	8651.03	61798	336.95	343.44
HFBC_30_5	74862	18911.10	67270	431.34	19342.44	67025	310.04	316.65

Table 6: Values of the objective function and time elapsed of the set 1 with *High flexibility* and *Bihour cost*

	Greedy-alone		Greedy		Battery		Greedy+Battery	
Instance	Obj	Time	Obj	Time	Obj	Time	Obj	Time
HFTC_10_1	-15701	0.04	-15701	0.37	-15701	0.05	-15701	0.47
HFTC_10_2	-10008	0.05	-10008	0.47	-10008	0.06	-10008	0.57
HFTC_10_3	-11560	0.04	-11560	0.43	-11560	0.05	-11560	0.53
HFTC_10_4	-9594	0.05	-9594	0.48	-9594	0.07	-9594	0.68
HFTC_10_5	-10672	0.04	-10672	0.44	-10672	0.04	-10672	0.44
HFTC_20_1	31394	0.12	30708	2.39	29698	0.13	28984	2.59
HFTC_20_2	27202	0.12	27202	2.31	25118	0.15	25118	2.91
HFTC_20_3	29574	0.11	28626	2.29	27272	0.14	26658	2.89
HFTC_20_4	24319	0.11	24201	2.25	22358	0.12	22246	2.45
HFTC_20_5	39074	0.12	38063	2.45	37626	0.14	36453	2.85
HFTC_30_1	90611	0.22	89731	6.54	88721	0.23	87820	6.84
HFTC_30_2	93761	0.23	93246	6.80	91815	0.28	91377	8.30
HFTC_30_3	80938	0.22	80786	6.45	78968	0.28	79065	8.25
HFTC_30_4	74854	0.21	74224	6.31	72635	0.24	72191	7.21
HFTC_30_5	96988	0.23	96758	6.93	94909	0.25	94737	7.53

Table 7: Values of the objective function and time elapsed of the set 1 with *High flexibility* and *Bihour cost*

	Cplex		Cplex+Polish			Greedy+Polish		
Instance	Obj	Time	Obj	Time	Tot time	Obj	Time	Tot time
HFTC_10_1	-11702	27.98	-15701	17.94	45.92	-15701	0.08	0.55
HFTC_10_1	25012	56.80	13527	55.34	112.14	-10008	0.18	0.75
HFTC_10_1	-9012	58.86	-11560	29.81	88.67	-11560	0.16	0.69
HFTC_10_1	-3553	69.69	-9594	42.11	111.80	-9594	0.14	0.82
HFTC_10_1	-9668	31.54	-10672	26.75	58.29	-10672	0.18	0.62
HFTC_20_1	62508	4931.38	33622	203.91	5135.29	25231	56.42	59.01
HFTC_20_1	31635	963.25	21512	177.54	1140.79	21137	159.86	162.77
HFTC_20_1	67604	1488.00	39941	153.06	1641.06	23321	98.02	100.91
HFTC_20_1	23113	1236.95	18759	136.55	1373.50	18751	204.44	206.89
HFTC_20_1	58944	2445.45	36922	201.06	2646.51	29269	242.93	245.78
HFTC_30_1	77307	46896.76	69965	462.66	47359.42	68168	639.94	646.78
HFTC_30_1	82663	26165.10	76253	455.37	26620.47	73189	623.60	631.90
HFTC_30_1	72430	25283.68	69187	414.00	25697.68	66960	649.77	658.02
HFTC_30_1	65173	40103.43	60221	359.73	40463.16	59901	350.58	357.79
HFTC_30_1	85910	93746.58	78110	635.97	94382.55	75778	791.40	798.93

Table 8: Values of the objective function and time elapsed of the set 1 with *High flexibility* and *Bihour cost*

	Greedy-alone		Greedy		Battery		Greedy+Battery	
Instance	Obj	Time	Obj	Time	Obj	Time	Obj	Time
MFBC_10_1	-10261	0.04	-10261	0.35	-10261	0.04	-10261	0.35
MFBC_10_1	-7414	0.04	-7414	0.37	-7414	0.05	-7414	0.47
MFBC_10_1	-9274	0.03	-9274	0.34	-9274	0.04	-9274	0.44
MFBC_10_1	-8740	0.03	-8740	0.34	-8740	0.03	-8740	0.34
MFBC_10_1	-11142	0.03	-11142	0.31	-11142	0.04	-11142	0.41
MFBC_20_1	28164	0.07	27684	1.48	27159	0.08	26797	1.68
MFBC_20_1	22479	0.07	22254	1.45	21290	0.08	21110	1.65
MFBC_20_1	30219	0.08	29864	1.52	28678	0.09	28727	1.72
MFBC_20_1	23938	0.07	22859	1.41	23085	0.11	21821	2.21
MFBC_20_1	19447	0.07	18164	1.37	17132	0.09	15657	1.77
MFBC_30_1	80922	0.12	80183	3.67	80017	0.14	79383	4.27
MFBC_30_1	72953	0.12	72677	3.59	71531	0.14	71137	4.19
MFBC_30_1	77858	0.12	76905	3.62	77137	0.13	76121	3.92
MFBC_30_1	94638	0.13	93272	3.76	93515	0.15	92147	4.36
MFBC_30_1	93856	0.13	93676	3.75	92598	0.15	92486	4.35

Table 9: Values of the objective function and time elapsed of the set 1 with *High flexibility* and *Bihour cost*

	Cplex		Cplex+Polish			Greedy+Polish		
Instance	Obj	Time	Obj	Time	Tot time	Obj	Time	Tot time
MFBC_10_1	-10261	3.89	-10261	0.05	3.94	-10261	0.06	0.41
MFBC_10_1	16919	10.39	-2590	18.71	29.10	-7414	0.07	0.54
MFBC_10_1	-2452	5.88	-9240	17.24	23.12	-9274	0.06	0.50
MFBC_10_1	21070	5.21	2475	12.11	17.32	-8740	0.07	0.41
MFBC_10_1	-6279	0.75	-11142	2.96	3.71	-11142	4.55	4.96
MFBC_20_1	61711	37.55	35099	60.21	97.76	26056	84.57	86.25
MFBC_20_1	72637	46.65	37874	52.76	99.41	20574	88.18	89.83
MFBC_20_1	63536	25.04	28743	64.99	90.03	27934	68.27	69.99
MFBC_20_1	58022	32.77	28550	44.61	77.38	19503	120.31	122.52
MFBC_20_1	59412	34.01	22287	51.26	85.27	14178	135.68	137.45
MFBC_30_1	103146	58.98	71900	108.33	167.31	64389	53.05	57.32
MFBC_30_1	101211	56.65	68009	121.37	178.02	64628	93.39	97.58
MFBC_30_1	126472	69.81	81808	100.49	170.30	71235	89.90	93.82
MFBC_30_1	137003	62.07	99336	99.57	161.64	85782	34.37	38.73
MFBC_30_1	139392	78.03	104491	67.41	145.44	85647	47.77	52.12

Table 10: Values of the objective function and time elapsed of the set 1 with *High flexibility* and *Bihour cost*

	Greedy-alone		Greedy		Battery		Greedy+Battery	
Instance	Obj	Time	Obj	Time	Obj	Time	Obj	Time
MFTC_10_1	-11528	0.03	-11528	0.34	-11528	0.04	-11528	0.44
MFTC_10_1	-9000	0.04	-9000	0.36	-9000	0.04	-9000	0.36
MFTC_10_1	-11502	0.03	-11502	0.33	-11502	0.05	-11502	0.53
MFTC_10_1	-9586	0.03	-9586	0.34	-9586	0.03	-9586	0.34
MFTC_10_1	-12619	0.03	-12619	0.30	-12619	0.04	-12619	0.40
MFTC_20_1	38289	0.08	37976	1.52	36510	0.09	35878	1.72
MFTC_20_1	43472	0.08	43472	1.56	41991	0.12	41991	2.36
MFTC_20_1	37067	0.08	35868	1.52	35246	0.09	34279	1.72
MFTC_20_1	32570	0.08	32570	1.53	30955	0.10	30955	1.93
MFTC_20_1	34767	0.07	34099	1.47	32833	0.09	32184	1.87
MFTC_30_1	78126	0.12	77081	3.52	76278	0.14	75237	4.12
MFTC_30_1	93060	0.13	92566	3.86	91496	0.15	90726	4.46
MFTC_30_1	94133	0.13	92967	3.82	92423	0.18	91467	5.32
MFTC_30_1	81474	0.12	80910	3.71	79535	0.17	79163	5.21
MFTC_30_1	80729	0.13	79660	3.75	78708	0.17	77714	4.95

Table 11: Values of the objective function and time elapsed of the set 1 with *High flexibility* and *Bihour cost*

	Cplex		Cplex+Polish			Greedy+Polish		
Instance	Obj	Time	Obj	Time	Tot time	Obj	Time	Tot time
MFTC_10_1	12355	5.37	-3657	15.38	20.75	-11528	0.05	0.49
MFTC_10_1	14012	8.17	-4205	18.02	26.19	-9000	0.05	0.41
MFTC_10_1	3258	3.48	-7951	14.92	18.40	-11502	0.05	0.58
MFTC_10_1	10341	6.40	-8133	15.82	22.22	-9586	5.60	5.94
MFTC_10_1	-12541	4.57	-12619	2.00	6.57	-12619	0.05	0.45
MFTC_20_1	71024	42.42	42651	61.71	104.13	28703	71.79	73.51
MFTC_20_1	58386	50.74	34516	60.78	111.52	32798	23.35	25.71
MFTC_20_1	57712	47.65	31030	65.62	113.27	27237	52.65	54.37
MFTC_20_1	55780	64.34	34000	61.59	125.93	26398	67.77	69.70
MFTC_20_1	44292	57.13	27543	58.98	116.11	27027	49.94	51.81
MFTC_30_1	114998	143.27	66026	114.45	257.72	59240	117.97	122.09
MFTC_30_1	126294	123.48	86320	136.05	259.53	73391	148.90	153.36
MFTC_30_1	116391	146.15	88079	138.94	285.09	72038	130.81	136.13
MFTC_30_1	70028	88.77	64555	96.67	185.44	64570	73.82	79.03
MFTC_30_1	71626	87.09	64583	88.54	175.63	64363	69.93	74.88

Table 12: Values of the objective function and time elapsed of the set 1 with *High flexibility* and *Bihour cost*

	Greedy-alone		Greedy		Battery		Greedy+Battery	
Set	Obj	Time	Obj	Time	Obj	Time	Obj	Time
HFBC 10	-10495	0.05	-10495	0.45	-10495	0.15	-10495	0.55
HFBC 20	26234	0.12	26086	2.38	25022	0.52	24895	2.78
HFBC 30	66796	0.21	66416	6.40	65291	1.11	65021	7.30
HFTC 10	-11507	0.04	-11507	0.44	-11507	0.14	-11507	0.54
HFTC 20	30313	0.12	29760	2.34	28414	0.52	27892	2.74
HFTC 30	87430	0.22	86949	6.61	85410	1.24	85038	7.63
MFBC 10	-9366	0.03	-9366	0.34	-9366	0.09	-9366	0.40
MFBC 20	24849	0.07	24165	1.45	23469	0.43	22822	1.81
MFBC 30	84045	0.12	83343	3.68	82960	0.66	82255	4.22
MFTC 10	-10847	0.03	-10847	0.33	-10847	0.11	-10847	0.41
MFTC 20	37233	0.08	36797	1.52	35507	0.48	35057	1.92
MFTC 30	85504	0.12	84637	3.73	83688	1.20	82861	4.81

Table 13: Values of the objective function and time elapsed of the set 1 with *High flexibility* and *Bihour cost*

	Cplex		Cplex+Polish			Greedy+Polish		
Set	Obj	Time	Obj	Time	Tot time	Obj	Time	Tot time
HFBC 10	14658	56.77	-3524	38.09	94.85	-10495	0.16	0.71
HFBC 20	39971	3054.38	30053	168.84	3223.23	23731	259.57	262.36
HFBC 30	67913	9585.77	63243	380.74	9966.51	62603	303.83	311.13
HFTC 10	-1785	48.97	-6800	34.39	83.36	-11507	0.15	0.69
HFTC 20	48761	2213.01	30151	174.42	2387.43	23542	152.33	155.07
HFTC 30	76697	46439.11	70747	465.55	46904.66	68799	611.06	618.68
MFBC 10	3799	5.22	-6152	10.21	15.44	-9366	0.96	1.36
MFBC 20	63064	35.20	30511	54.77	89.97	21649	99.40	101.21
MFBC 30	121445	65.11	85109	99.43	164.54	74336	63.70	67.91
MFTC 10	5485	5.60	-7313	13.23	18.83	-10847	1.16	1.57
MFTC 20	57439	52.46	33948	61.74	114.19	28433	53.10	55.02
MFTC 30	99867	117.75	73913	114.93	232.68	66720	108.29	113.10

Table 14: Values of the objective function and time elapsed of the set 1 with *High flexibility* and *Bihour cost*

	Greedy-alone		Greedy		Battery		Greedy+Battery	
Set	Obj	Time	Obj	Time	Obj	Time	Obj	Time
HFBC 10	1472	0.00	1472	0.04	1472.21	0.00	1472.21	0.04
HFBC 20	1305	0.00	1240	0.08	1325.39	0.00	1253.23	0.08
HFBC 30	7497	0.01	7693	0.24	7237.91	0.01	7414.14	0.24
HFTC 10	2200	0.00	2200	0.04	2199.83	0.01	2199.83	0.08
HFTC 20	4981	0.00	4660	0.07	5202.93	0.01	4807.88	0.18
HFTC 30	8269	0.01	8289	0.23	8329.68	0.02	8279.07	0.57
MFBC 10	1278	0.00	1278	0.02	1277.95	0.01	1277.95	0.05
MFBC 20	3885	0.00	4153	0.05	4142.53	0.01	4603.04	0.21
MFBC 30	8712	0.00	8609	0.07	8688.95	0.01	8625.65	0.16
MFTC 10	1344	0.00	1344	0.02	1344.19	0.01	1344.19	0.07
MFTC 20	3685	0.00	3794	0.03	3767.95	0.01	3858.50	0.24
MFTC 30	6709	0.00	6753	0.12	6844.28	0.02	6844.17	0.46

Table 15: Values of the objective function and time elapsed of the set 1 with *High flexibility* and *Bihour cost*

	Cplex		Cplex+Polish			Greedy+Polish		
Set	Obj	Time	Obj	Time	Tot time	Obj	Time	Tot time
HFBC 10	19416	17.69	9959	19.21	32.83	1472	0.02	0.05
HFBC 20	23070	2399.55	12126	41.46	2425.66	1330	68.98	69.02
HFBC 30	7345	4957.18	5647	96.65	5012.17	5943	112.44	112.57
HFTC 10	13667	16.33	10372	13.03	27.19	2200	0.04	0.09
HFTC 20	17882	1448.02	8463	26.38	1466.77	3590	68.01	68.02
HFTC 30	7374	25047.69	6299	92.75	25127.81	5494	143.43	143.57
MFBC 10	12718	3.13	5262	7.50	10.19	1278	1.79	1.80
MFBC 20	5147	7.03	5475	7.12	8.13	4909	24.77	24.88
MFBC 30	16332	7.84	14532	17.82	10.92	9610	23.64	23.51
MFTC 10	9730	1.60	3233	5.71	6.63	1344	2.22	2.18
MFTC 20	8498	7.61	5012	2.17	7.08	2310	17.09	16.89
MFTC 30	24035	25.58	10876	20.26	43.77	5271	31.34	31.16

Table 16: Values of the objective function and time elapsed of the set 1 with *High flexibility* and *Bihour cost*

Instance	Energy consumption
HFBC_10_1	19520
HFBC_10_1	21651
HFBC_10_1	18811
HFBC_10_1	22926
HFBC_10_1	10884
HFBC_20_1	43072
HFBC_20_1	42720
HFBC_20_1	43946
HFBC_20_1	42750
HFBC_20_1	43635
HFBC_30_1	57289
HFBC_30_1	65905
HFBC_30_1	62551
HFBC_30_1	62127
HFBC_30_1	64763
HFTC_10_1	15552
HFTC_10_1	21245
HFTC_10_1	19693
HFTC_10_1	21659
HFTC_10_1	20581
HFTC_20_1	43868
HFTC_20_1	41759
HFTC_20_1	42913
HFTC_20_1	40620
HFTC_20_1	45886
HFTC_30_1	65018
HFTC_30_1	67396
HFTC_30_1	64724
HFTC_30_1	61203
HFTC_30_1	68684

Table 17: Energy total consumption of all the appliances in Watt per time slot.

Instance	Energy consumption
MFBC_10_1	20992
MFBC_10_1	23839
MFBC_10_1	21979
MFBC_10_1	22513
MFBC_10_1	20111
MFBC_20_1	44232
MFBC_20_1	41528
MFBC_20_1	45215
MFBC_20_1	41003
MFBC_20_1	38309
MFBC_30_1	63445
MFBC_30_1	63565
MFBC_30_1	61651
MFBC_30_1	66516
MFBC_30_1	68274
MFTC_10_1	19725
MFTC_10_1	22253
MFTC_10_1	19751
MFTC_10_1	21667
MFTC_10_1	18634
MFTC_20_1	45603
MFTC_20_1	47652
MFTC_20_1	44868
MFTC_20_1	44446
MFTC_20_1	44758
MFTC_30_1	60869
MFTC_30_1	67853
MFTC_30_1	67093
MFTC_30_1	63521
MFTC_30_1	63432

Table 18: Energy total consumption of all the appliances in Watt per time slot.

Appendix D: Some plots

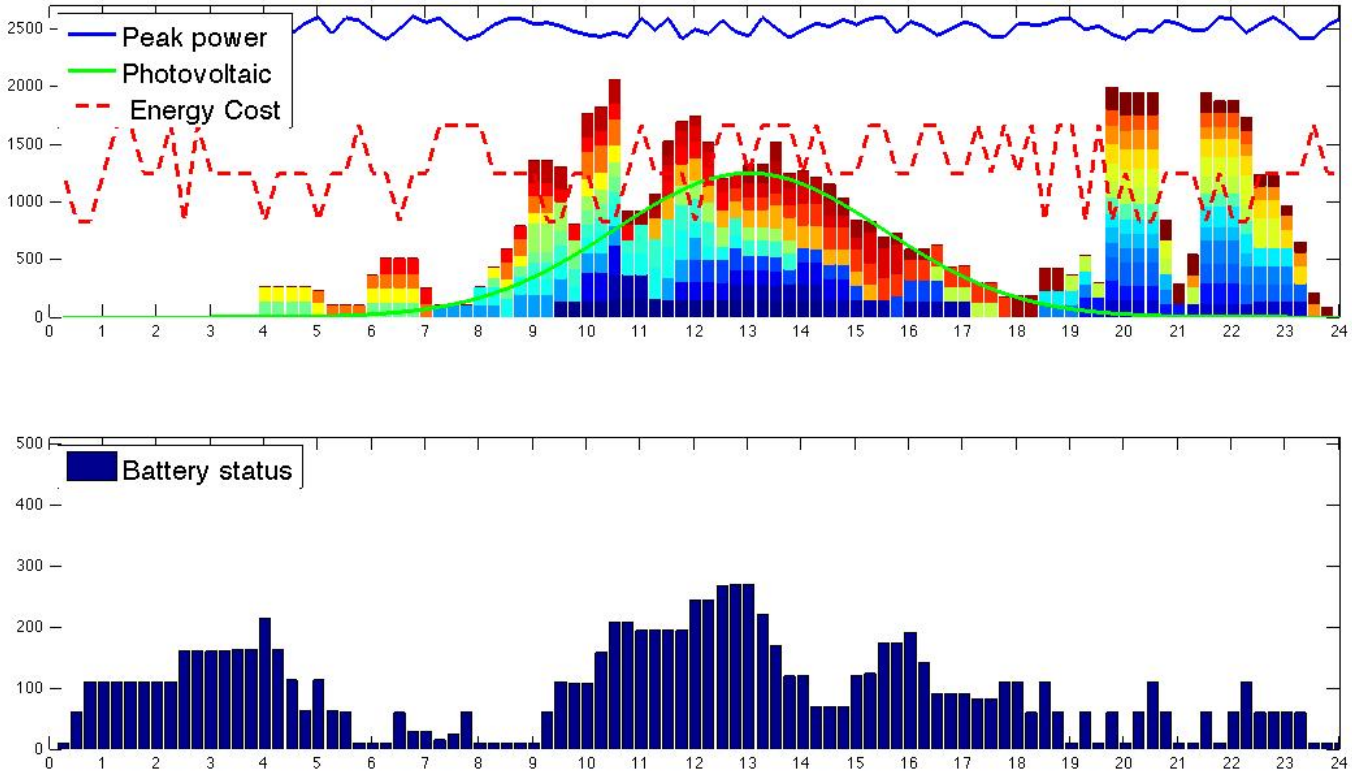


Figure 12: Example of 30 appliances scheduling and battery behavior on HFTC using `Greedy+Battery`. Each color represents a different appliance.

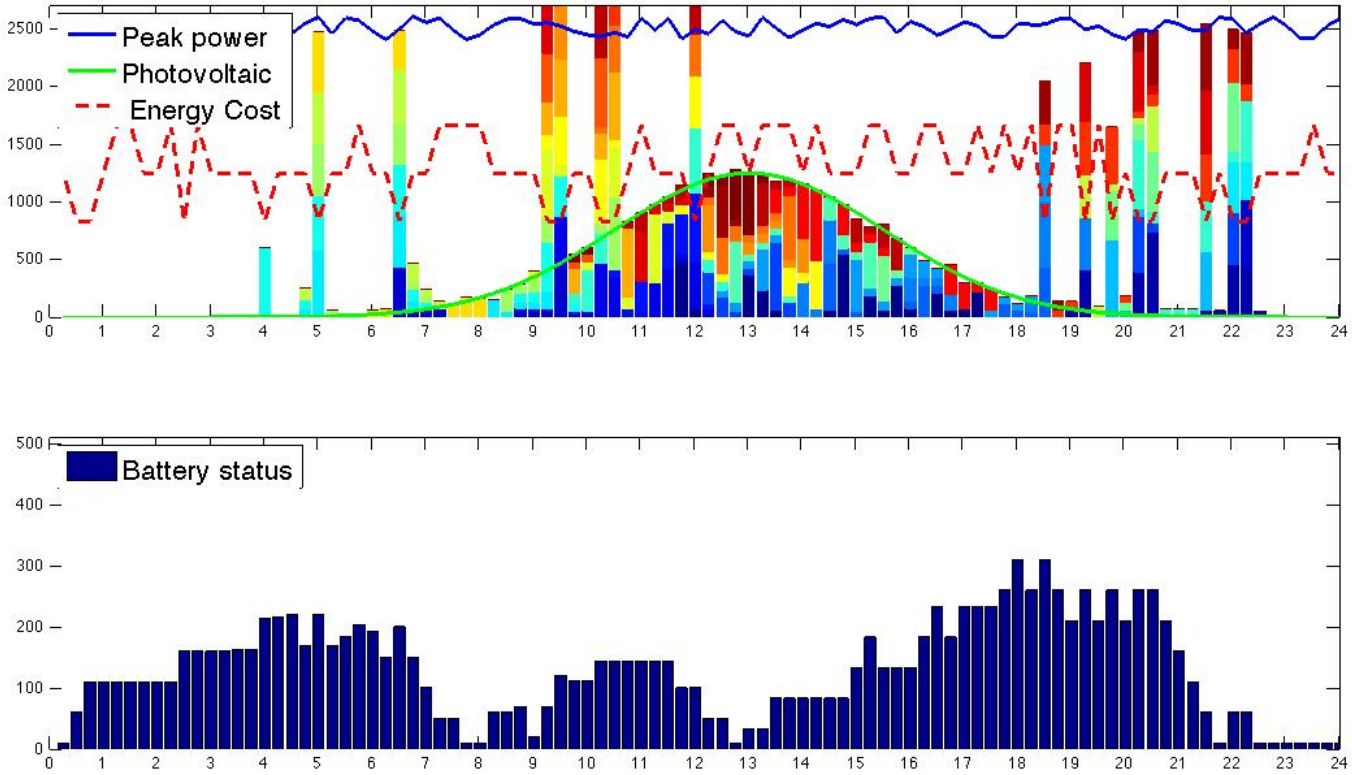


Figure 13: Example of 30 appliances scheduling and battery behavior on HFTC using `Greedy+Polish`. Each color represents a different appliance.

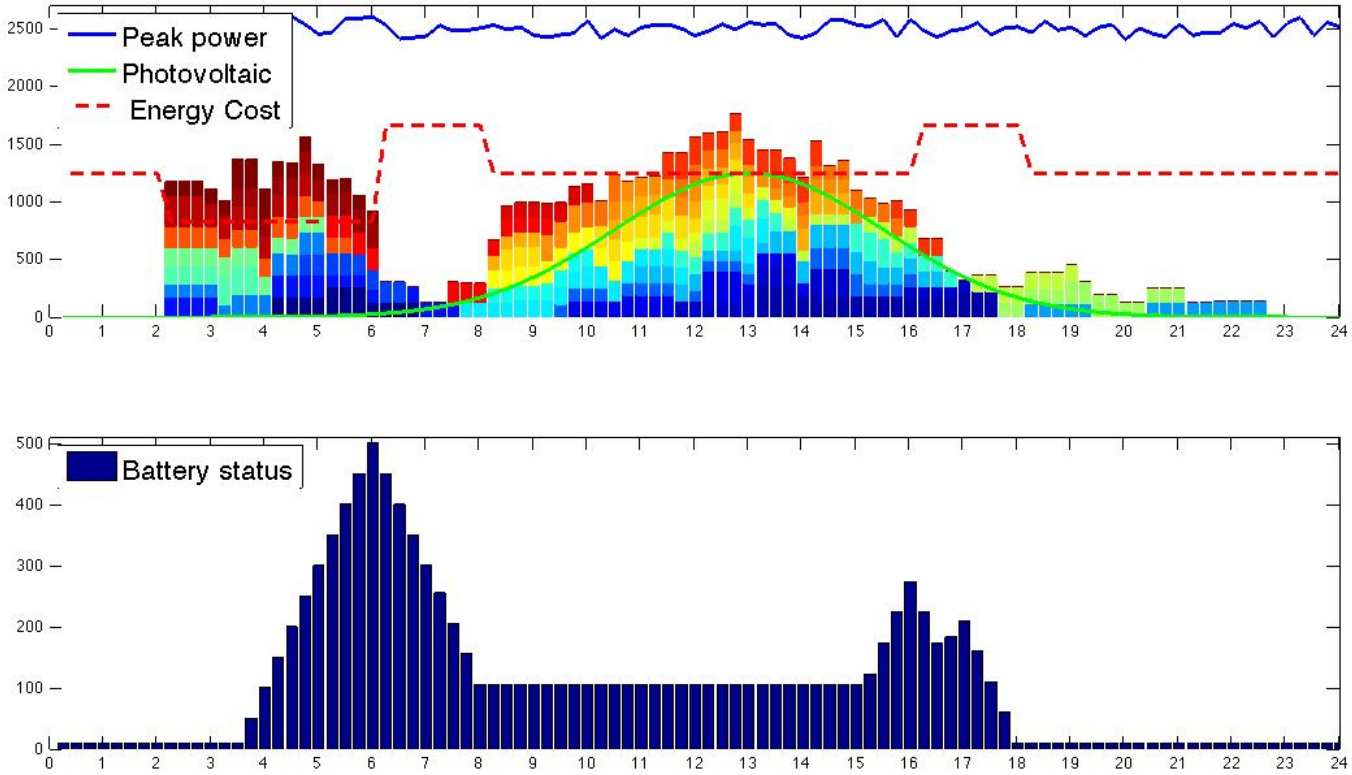


Figure 14: Example of 30 appliances scheduling and battery behavior on MFBC using `Greedy+Battery`. Each color represents a different appliance.

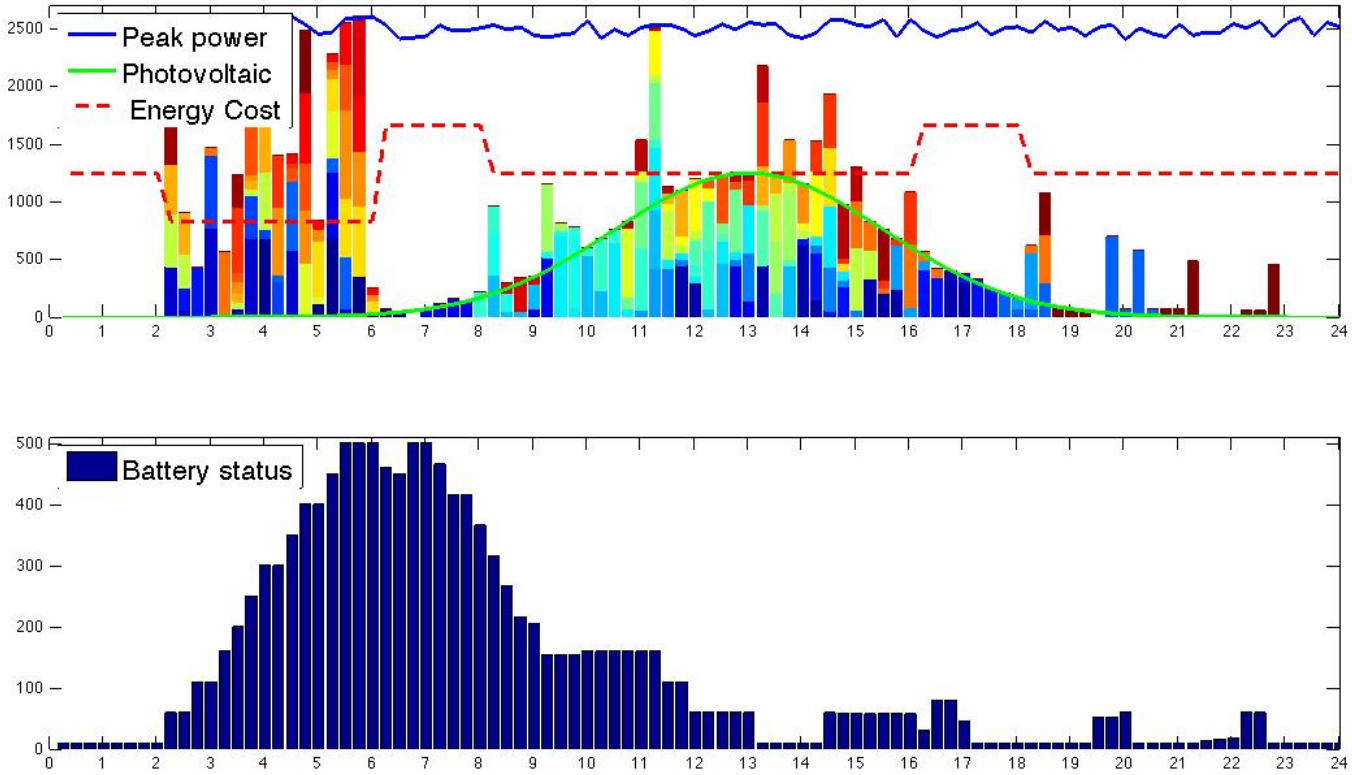


Figure 15: Example of 30 appliances scheduling and battery behavior on MFBC using `Greedy+Polish`. Each color represents a different appliance.

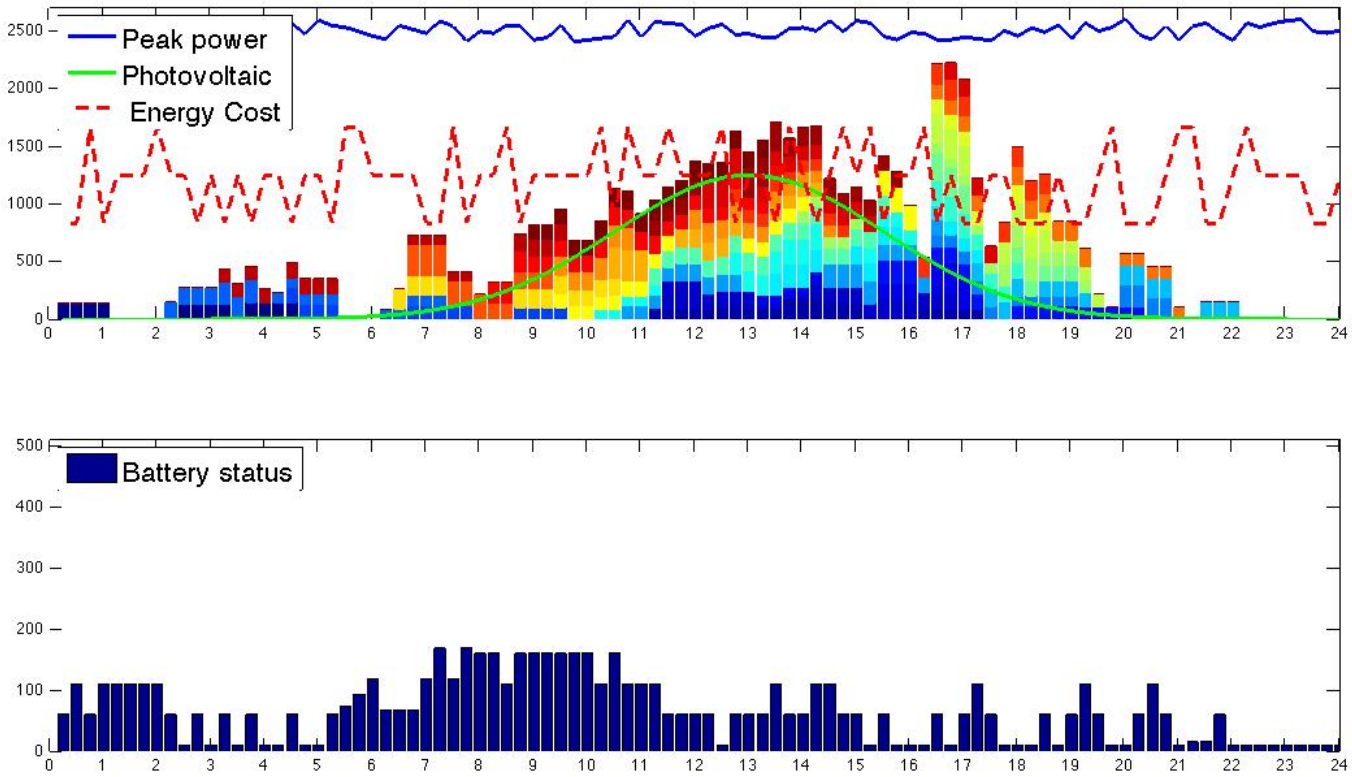


Figure 16: Example of 30 appliances scheduling and battery behavior on MFTC using `Greedy+Battery`. Each color represents a different appliance.

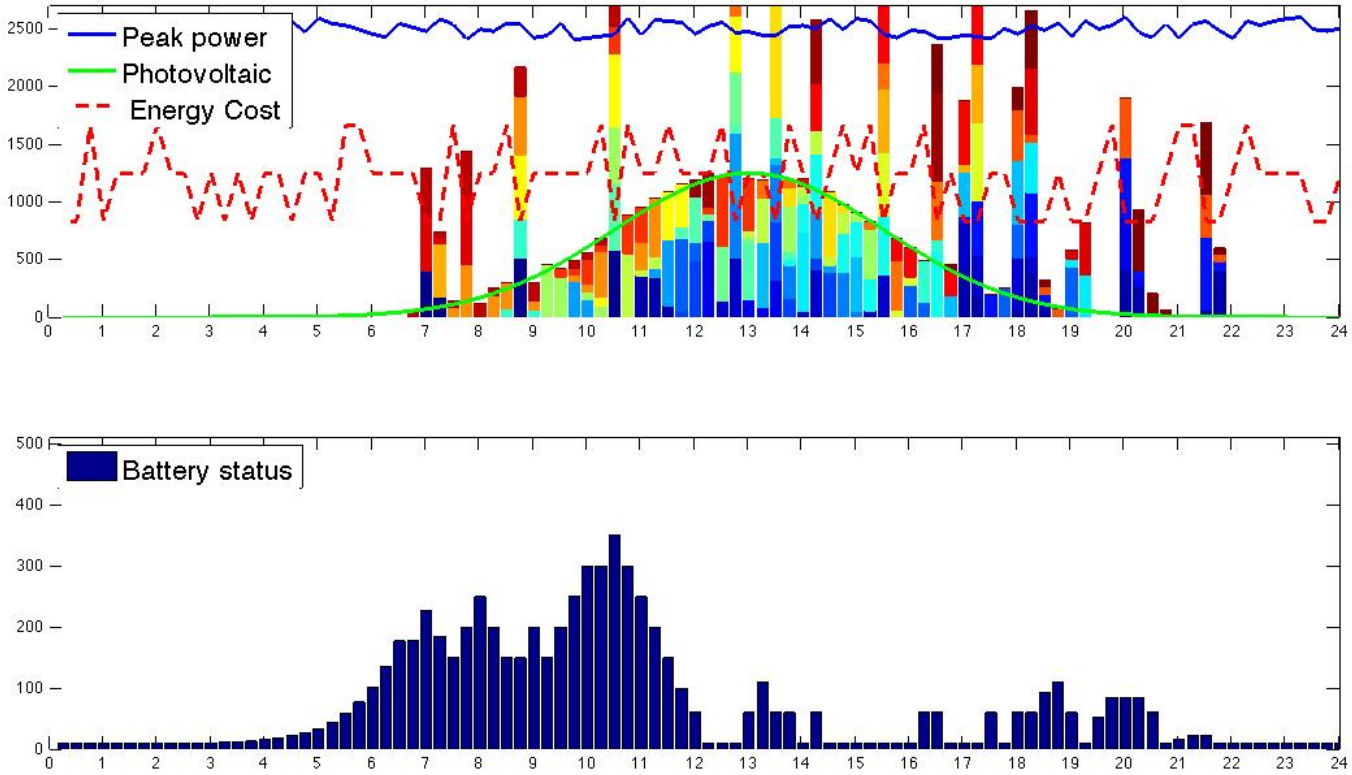


Figure 17: Example of 30 appliances scheduling and battery behavior on MFTC using `Greedy+Polish`. Each color represents a different appliance.

Ringraziamenti

Voglio ringraziare il Professor Matteo Fischetti, cha mi ha dato fiducia fin dall'inizio e mi ha sempre seguito con estrema disponibilità.

Voglio ringraziare anche l'Ing. Arrigo Zanette, che ha seguito tutto il progetto con molta attenzione e mi ha sempre dato una mano quando ne avevo bisogno.

Colgo l'occasione per ringraziare anche l'azienda che mi ha ospitato, l'M31 Italia S.r.l., che mi ha permesso di trasformare in realtà qualcosa che era solamente una successione di righe di codice.

Come non ricordare tutti gli amici che in questi anni mi hanno spronato e mi sono stati vicini. In particolare ringrazio David, mio fedele compagno di classe alle superiori e compagno di stanza all'università. Ringrazio anche il mio collega di "MIT's courses" Davide Fauri (detto Fax). Ma soprattutto ringrazio i tre disgraziati che mi hanno sopportato durante l'ultimo anno, Mattia, Luca e il "nonno" Claudio.

Un ringraziamento particolare lo devo ad Elena. Mi hai sopportato, incoraggiato, consigliato, aiutato in un modo che neanche avrei potuto immaginare. Sei stata la luce che mi ha indicato la strada e che mi ha accompagnato lungo il cammino.

Ringrazio i miei nonni per aver sempre pregato per me e per le loro "paghette". Un saluto particolare alla nonna Alda che non ha mai smesso di sgridarmi quando credeva studiassi poco. Voglio ricordare con affetto anche il nonno Franco che avrebbe tanto voluto poter assistere alla mia laurea.

...

L'ultimo ringraziamento e anche il più speciale va ai miei genitori e a mio fratello Luca, per i quali avrei così tante cose da scrivere che preferisco condensarle in un sola parola, che però proviene dal cuore: GRAZIE.

E grazie a chiunque non abbia citato ma che mi ha dato una mano in questi anni speciali.