Ranked Particle Swarm Optimization with Lévy's Flight

Optimization of appliance scheduling for smart residential energy grids

Ennio Grasso, Giuseppe Di Bella, and Claudio Borean
Swarm Joint Open Lab
TELECOM ITALIA
Turin, Italy
e-mail: ennio.grasso@telecomitalia.it, giuseppe.dibella@telecomitalia.it, claudio.borean@telecomitalia.it

***Abstract*— This paper analyzes the problem of scheduling home appliances in the context of smart home applications. The optimization problem is modeled and different approaches to tackle it are presented and discussed. A new metaheuristic algorithm named Ranked Particle Swarm with Lévy flights (RaPSOL) is then proposed and described. The algorithm runs on the limited computational power provided by the home gateway device and in almost real-time as of user perception. Simulation results of RaPSOL algorithm applied in different use case scenarios are presented and compared with other approaches. The simulations include validation of the method in variable conditions considering both consumption, micro-generation and imposed user constraints.**

***Keywords*— scheduling; swarm intelligence; metaheuristic smart grids; smart homes.**

I. INTRODUCTION

This paper considers the minimum electricity cost scheduling problem of smart home appliances. Functional characteristics, such as expected duration and power consumption of the smart appliances can be modeled through a power profile signal. The optimal scheduling of power profile signals minimizes cost, while satisfying technical operation constraints and consumer preferences. Time and power constraints, and optimization cost are modeled in this framework using a metaheuristic algorithm based on a variant of Particle Swarm Optimization (PSO), presented in [1]. The algorithm runs on the limited computational power provided by the home gateway device and in almost real-time as of user perception. The context refers to the smart home environment, described in the INTrEPID European project [2], where a home environment equipped with plug-sensors and smart appliances can be used for enhanced smart energy management services.

The proposed framework can optimize appliance scheduling to minimize energy cost while avoiding the overload threshold. Very good quality solutions can be obtained in short computation time, in the order of a few seconds, which enables the deployment of this algorithm in low-cost embedded platforms.

Owing to the pliable characteristics of metaheuristic algorithms, the proposed algorithm is easily extended to incorporate solar power production forecasting in the presence of residential photovoltaic (PV) systems by simply adapting the objective function and using the solar energy forecaster as further input to the scheduler ([3][4]).

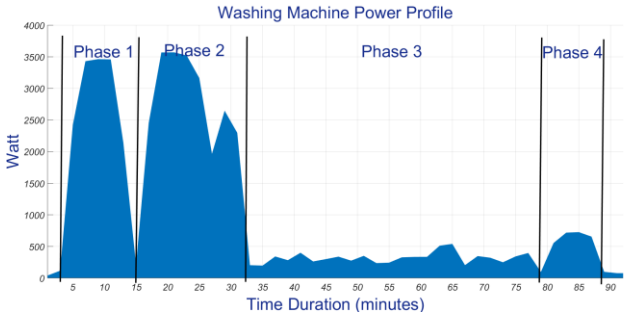

Figure 1. Example Power Profile with its phases generated by a washing machine

The paper is structured as it follows. Section II describes a model of the problem for the scheduling of smart appliance. Section III highlights how this problem can be classified as a NP-Hard Combinatorial Optimization Problem. In Section IV, a give a broad review of metaheuristic, while in Section V the new algorithm proposed in the paper is described. Section VI reports the results of the simulations of the proposed algorithm applied to the problem of scheduling smart appliances. Finally, Section VII contains concluding remarks and future analysis.

II. SCHEDULING PROBLEM OF SMART HOME APPLIANCES

Smart home applications are becoming one of the driving force of the Internet-of-Things (IoT), since connecting smart devices such as smart appliance to the internet envisions new scenarios that provide added value to both the final users and the other stakeholders. Possible applications are for instance the remote monitoring of smart appliances, remote activation/deactivation, automatic failure detection and alarm notification. Likewise, applications for appliance makers range from the remote diagnosis and assistance of appliances, thus reducing the assistance costs, to the collection of appliance statistics information useful to improve strategies for marketing new products, i.e., the appliance vendor could offer discounts in exchange for being allowed to get access to usage patterns and uncover the

features more appealing to their customers. To foster the pervasive adoption of these new IoT services, a common set of features need to be shared among the connected devices, so that "silo services" provided by each vendor are replaced by a smart home ecosystems where the connected appliance share value in participating.

One of the most successful application of smart home systems is energy management, since smart energy applications are enabled by IoT technologies and are shared by many home devices, which are mains powered. With the increased needs for energy sustainability, both regulatory and nationwide organizations are urging to the adoption of Renewable Energy Sources (RES) to reach the compelling target of the Horizon-2020 strategy. Since RES are by their nature variable and oftentimes difficult to predict exactly subject to variable weather conditions (PV performance mainly depends on cloud-cover condition, while wind turbines depends on the wind strength and direction), the final tariffs of electricity should match the fickle dynamics of the effective production cost instead of current two, or at most three tier model in most countries.

To enable a scenario with highly dynamic energy tariffs, it is essential the introduction if intelligent systems that can autonomously and conveniently schedule appliances to optimize energy use in presence of RES and variable tariffs.

On top of the above considerations, new actors such as Energy Aggregators are entering the market to collect and manage demands in so called "energy-districts". From the Aggregator standpoint, the proper management of energy demands of a set of users allows purchasing energy in the gross market and sharing the savings with the end users. These scenarios are explored in the INTrEPID project. Another important requirement is also the "shaving" of peak energy demands that cause inefficiencies in the electricity network (e.g., over-sizing electricity network to avoid blackouts) with additional costs and increased hazards (e.g., blackouts in case of power peaks not properly managed by the electricity network).

The management of users energy demand can be leveraged by the introduction of IoT systems, such as connected appliances, smart-plugs, smart-meters, apps for smartphones and tablet in order to visualize proposals to the users. These systems can take part in an energy management application with the aim to optimize the scheduling of appliances in the homes of district.

Taking into account the above considerations, not only is an automatic decision system highly desirable but even necessary in most cases, which either directly takes control of the appliances' operations (depending on the availability of smart appliances in the market), or at the very least is capable of providing advice to the home consumers (in case where using IoT system the appliance consumptions patterns can be learned and used for the scheduling).

This paper considers the minimum electricity cost scheduling problem of smart home appliances in the context of the INTrEPID Project. Functional characteristics, such as expected duration, mean and peak power consumption of smart appliances can be modeled through a power profile signal in time. Such power profiles could also be inferred by proper disaggregation of the cumulated power of a single smart meter with Non-Intrusive Load Monitoring (NILM) techniques. In other more advanced scenarios, the power profiles are notified by the smart appliances themselves. Protocols that enable that scenario have already been specified in several standard bodies and associations such as Energy@home [5].

In view of the above considerations, not only is an automatic decision system highly desirable but even necessary in most cases, which either directly takes control of the appliances' operations, or at the very least is capable of providing advice to the home consumers.

A. Smart Appliances in smart home

The smart home applications are enabled by communication between devices (e.g., smart appliances) in a home network typically enabled by wireless technologies.

The core element of a home network is the Home Gateway (HG) that coordinates and manages the smart appliances as end-devices. Among its functionalities, the HG provides the intelligence for real-time scheduling of residential appliances, typically in the time interval 24 hours, based on the tariff of the day, the forecasted energy power consumption, and possibly the forecasted wind/PV power generation.

The proposed scheduling framework borrows from the Power Profile Cluster defined in the E@H specifications [5], which specifies that each appliance operation cycle is modeled as a power profile composed by a set of sequential energy phases, as depicted in Figure 1. In some situation, and without loss of generality, a power profile has just a single phase, and in that simple case the power profile and its phase simply coincide.

In the more general case in which a power profile is composed of several energy phases, each phase represents an atomic subtask of the appliance's operation cycle. All phases are ordered sequentially since a phase cannot start until the previous phase is completed[1], however, there may be some degree of freedom in the time slack between one phase and the next.

Therefore, in general, each energy phase is characterized by a time duration and a power signal in time domain with the chosen sampling frequency[2], and a maximum activation delay after the end of the previous phase. Some phases have a maximum delay of zero, meaning that they cannot be delayed and must start soon after the previous phase completes. Other phases may be delayed adding extra flexibility in the scheduling of the power profile, e.g., the

[1] e.g., a washing machine agitator cannot start until the basin is filled with water

[2] Typical sampling frequency are 1 Hz or 1/60 Hz

washing machine agitator must start within ten minutes of the basin being filled.

Another input to the scheduler is the user's time constraints, demanding that certain appliances be scheduled within some particular time intervals, e.g., the dishwasher must run between 13:00 and 18:00.

The objective of the HG scheduler is to find the least expensive scheduling for a set of smart appliances, each characterized by a power profile with its energy phases, while satisfying the necessary operational constraints.

B. Modeling the Scheduling Problem

A first step in the scheduling problem modeling is to determine its dimension. Being *N* the number of appliances considered, and denoting by np_i the number of energy phases associated with each appliance i, the problem dimension, corresponding to the overall number of phases, is trivially given by

$$|P| \stackrel{\text{def}}{=} \sum_{i=1}^{N} np_i \quad (1)$$

The objective of the scheduler is to minimize the total electricity cost for operating the appliances based on the 24-hour electricity tariff while respecting time and energy constraints.

Denoting with $\boldsymbol{x} \in T^{|P|}$ the vector of start times of the $|P|$ phases, where T is the scheduling time interval, the problem can be stated as:

$$\boldsymbol{x} = \arg\min_{\boldsymbol{x}}(C(\boldsymbol{x})) \quad (2)$$

being $C(\boldsymbol{x})$, the total cost, expressed as

$$C(\boldsymbol{x}) = \sum_{i=1}^{N} \sum_{j=1}^{np_i} C(x_{ij}) \quad (3)$$

and $C(x_{ij})$ the cost of starting phase j of appliance i at time x_{ij}. The cost of a single phase at a given time is simply the product of the power phase signal and the tariff in the subinterval, L_{ij}, from the start time to the end of the energy phase.

$$C(x_{ij}) = \int_{x_{ij}}^{x_{ij}+L_{ij}} tariff(t)\, power_{ij}(t - x_{ij})dt \quad (4)$$

The integral notation assumes that the mean power is a Lebesgue integrable function. The above formulation is the most general possible, which assumes the power signal is a continuous function. An approximate formulation is to discretize the problem by choosing a reasonable sampling frequency, i.e., a trade-off with regard to the power profile signal variability and the desired system accuracy.

Following this idea, a reasonable approximation is to discretize the day time interval into 1440 time slots of 1 minute each. In such formulation, the above integral reduces to its summation approximate

$$C(x_{ij}) = \sum_{t=x_{ij}}^{x_{ij}+L_{ij}} tariff(t) \cdot power_{ij}(t - x_{ij}) \quad (5)$$

The max power constraint imposes that at any given time the amount of power required by all appliances' active phases be less than the peak power threshold specified by the grid operator. Let us define the auxiliary allocation function on the whole support of the scheduling interval *T*,

$$allocPower_{ij}(t) = \begin{cases} power_{ij}(t - x_{ij}) & if\ t \in [x_{ij}, x_{ij} + L_{ij}] \\ 0 & otherwise \end{cases}. \quad (6)$$

Now we can define the max power constraint as

$$\sum_{i=1}^{N} \sum_{j=1}^{np_i} allocPower_{ij}(t) \quad < maxPower\ ,\ \forall t \in T \quad (7)$$

While the max power constraints apply to the optimization problem, time constraints simply restrict the scheduling interval. Time constraints are twofold. On the one hand, the end user can impose a scheduling interval for any appliance, in terms of an earliest start time (EST), e.g., after 13:20, and a latest end time (LET), e.g., before 18:00.

$$EST_i \leq x_{i1}\ ;\ x_{iP} + L_{iP} \leq LET_i \quad (8)$$

The above time constraint means that start time of the 1st phase of appliance i, x_{i1}, must occur after the imposed EST_i. Likewise, the completion time of the last phase, denoted by $x_{iP} + L_{iP}$, must occur before the imposed LET_i.

The second time constraint is the maximum activation delay of each of the sequential phases that make up each power profile. While the scheduling interval specified in the first constraint is absolute, the maximum activation delays are relative and, therefore, the lower and upper bound time limits of each phase need to be adjusted based on the scheduling decisions for the previous phase.

$$(x_{ij} + L_{ij}) \leq x_{i(j+1)} \leq (x_{ij} + L_{ij}) + maxDelay_{i(j+1)} \quad (9)$$

III. NP-Hard Combinatorial Optimization Problems

Given the problem formulation, the scheduling of power profiles, each composed by a set of sequential and possibly delayable phases, under energy constraints is classified in the more general family of Resource Constrained Scheduling Problem (RCSP), which is known as being an NP-Hard combinatorial optimization problem [6][7] .

Moreover, the presence of time constraints introduces even another dimension to the complexity of problem, known as RCSP/max, i.e., RCSP with time windows. Combining the inherent complexity of the problem with the fact that the limited computing power of the HG which runs the logic of algorithm, and the almost real-time requirement for finding a solution (typically the end user wants a

perceived immediate answer), make the formulation a challenging problem.

From a theoretical perspective, combinatorial optimization problems have a well-structured definition consisting of an objective function that needs to be minimized (e.g., the energy cost) and a series of constraints. These problems are important for many real-life applications.

For some problems, exact methods can be exploited, such as branch-and-cut and Mixed Integer Linear Programming (MILP), with back-tracking and constraints propagation to prune the search space. However, in most circumstances, the solution space is highly irregular and finding the optimum is in general impossible. An exhaustive method that checks every single point in the solution space would be infeasible in these difficult cases, since it takes exponential time.

As a point of fact, [8] also addresses a similar scheduling problem of smart appliances, and relies on traditional MILP as a problem solver. They provide computation time statistics for their experiments, running on an Intel Core i5 2.53GHz equipped with 4GB of memory and using the commercial application CPLEX and MATLAB. According to their figures, discretizing the time interval in 10-minute discrete slots (for a total of 144 daily slots), takes their algorithm about 15.4 seconds to find a solution. With 5-minute slots the time rises to 83.6 seconds and with 3-minute slots to 860 seconds. From these figures, it is clear that a traditional approach like MILP is hardly acceptable for scheduling home appliances, and other more efficient methods need to be investigated.

A. Convex and Smooth Objective Functions

Generally speaking, optimization problems can be categorized, from a high-level perspective, as having either a convex or non-convex formulation.

A convex formulation enables to represent the objective function as a series of convex regions where traditional deterministic methods work best and fast, such as conjugate gradient descent and quasi-Newton variants, like L-BFGS (Limited memory Broyden–Fletcher–Goldfarb–Shanno). The main idea, in convex optimization problems, is that every constraint restricts the space of solutions to a certain convex region. By taking the intersection of all these regions we obtain the set of feasible solutions, which is also a convex region. Due to the nice structure of the solution space, every single local optimum is a global one. Most conventional or classic algorithms are deterministic. For example, the simplex method in linear programming is deterministic, and use gradient information in the search space, namely the function values and their derivatives.

Non-convex constraints create a many disjoint regions, and multiple locally optimal points within each of them. As a result, if a traditional search method is applied, there is a high risk of ending in a local optimum that may still be far away from the global optimum. But the main drawback is that it can take exponential time in the size of problem dimension to determine if a feasible solution even exists.

Another definition is that of smooth function, i.e., a function that is differentiable and its derivative is continuous. If the objective function is non-smooth, the solution space typically contains multiple disjoint regions and many locally optimal points within each of them. The lack of a nice structure makes the application of traditional mathematical tools, such as gradient information, very complicated or even impossible in these cases.

However, many real problems are neither convex nor smooth, and so deterministic optimization methods can hardly be applied.

B. An Overview of General Metaheuristic Algorithms

A problem is NP-Hard if there is not an exact algorithm that can solve the problem in polynomial time with respect to the problem's dimension. In other words, aside from some "toy-problems", an NP-Hard problem would require exponential time to find a solution by systematically "exploring" the solution space.

A common method to turn an NP-Hard problem into a manageable, feasible approach is to apply heuristics to "guide" the exploration of the search space. These heuristics are based on "common-sense" specific for each problem and are the basis for developing Greedy Algorithms that can build the solution by selecting at each step the most promising path in the solution space based on the suggested heuristics. Obviously, this approach is short-sighted since it proceeds with incomplete information at each step. Very rarely do greedy algorithms find the best solution or worse yet they might fail to find a feasible solution even if one does exist.

A better approach for solving complex NP-Hard problems that has shown great success is based on metaheuristic algorithms. The word *meta* means that their heuristics are not problem specific to a particular problem, but general enough to be applied to a broad range of problems. Examples of metaheuristic algorithms are *Genetic and Evolutionary Algorithms*, *Tabu search*, *Simulated Annealing*, *Greedy Randomized Adaptive Search Procedure*, *Particle-Swarm-Optimization*, and many others.

The idea of metaheuristics is to have efficient and practical algorithms that work most the time and are able to produce good quality solutions, some of them will be nearly optimal. Figuratively speaking, searching for the optimal solution is like *treasure-hunting*. Imagine we are trying to find a hidden treasure in a hilly landscape within a time limit. It would be a silly idea to search every single square meter of an extremely large region with limited resources and limited time. A more sensible approach is to go to some place almost randomly and then move to another plausible place using some hints we gather throughout.

Two are the main elements of all metaheuristic algorithms: intensification and diversification. *Diversification* via randomization means to generate diverse solutions so as to explore the search space on the global scale and to avoid being trapped at local optima.

Intensification means to focus the search in a local region by exploiting the information that a current good solution is found in this region as a basis to guide the next step in the search space. The fine balance between these two elements is very important to the overall efficiency and performance of an algorithm.

IV. CLASSIFICATION OF METAHEURISTIC ALGORITHMS

Metaheuristic algorithms are broadly classified in two large families: *population-based* and *trajectory-based*. Going back to the treasure-hunting metaphor, in a trajectory-based approach we are essentially performing the search alone, moving from one place to the next based on the hints we have gathered so far. On the other hand, in a population-based approach we are asking a group of people to participate in the hunting sharing all information gathered by all members to select the most promising paths for the next moves.

A. Genetic Algorithms

Genetic Algorithms (GA) were introduced by John Holland and his collaborators at the University of Michigan in 1975 [9]. A GA is a search method based on the abstraction of Darwinian evolution and natural selection of biological systems, and representing them in the mathematical operators: *crossover* (or recombination), *mutation*, *fitness evaluation* and *selection* of the best. The algorithm starts with a set of candidate solutions, the initial population, and generate new offspring through random mutation and crossover, and then applies a selection step in which the worst solutions are deleted while the best are passed on to the next generation. The entire process is repeated multiple times and gradually better and better solutions are obtained. GA algorithms represent the inseminating idea of all more recent population-based metaheuristics.

One major drawback of GA algorithms is the "conceptual impedance" that arises when trying to formulate the problem at hand with the genetic concepts of the algorithm. The formulation of the fitness function, population size, the mutation and crossover operators, and the selection criteria of the offspring population are crucially important for the algorithm to converge and find the best, or quasi-best, solution.

B. Simulated Annealing

Simulated Annealing (SA) was introduced by Kirkpatrick et al. in 1983 [10] and is a trajectory-based approach that simulates the evolution of a solid in a heat bath to thermal equilibrium. It was observed that heat causes the atoms to deviate from their original configuration and transition to states of higher energy. Then, if a slow cooling process is applied, there is a relatively high chance for the atoms to form a structure with lower internal energy than the original one. Metaphorically speaking, SA is like dropping a bouncing ball over a hilly landscape, and as the ball bounces and loses its energy it eventually settles down to some local minima. But if the ball loses energy slowly enough keeping its momentum, it might have a chance to overcome some local peaks and fall through a better minimum.

C. Particle Swarm Optimization

Particle Swarm Optimization (PSO), introduced in 1995 by American social psychologist James Kennedy, and engineer Russell C. Eberhart [11], represents a major milestone in the development of population-based metaheuristic algorithms. PSO is an optimization algorithm inspired by swarm intelligence of fish and birds or even human behavior. The multiple particles swarm around the search space starting from some initial random guess and communicate their current best solutions and also share their best. The greatest advantage of PSO over GA is that it is much simpler to apply in the formulation of the problem. Instead of using crossover and mutation operations it exploits global communication among the swarm particles. Each particle in the swarm modifies its position with a velocity that includes a first component that attracts the particle towards the best position so far achieved by the particle itself. This component represents the personal experience of the particle. The second component attracts the particle towards the best solution so far achieved by the swarm as a whole. This component represents the social communication skill of the particles.

Denoting with N the dimensionality of the search space, i.e., the number of independent variables that make up the exploring search space, each individual particle is characterized by its position and velocity N-vectors. Denoting with x_i^k and v_i^k respectively the position and velocity of particle i at iteration k, the following equations are used to iteratively modify the particles' velocities and positions:

$$v_i^{k+1} = w v_i^k + c_1 r_1 \left(p_i - x_i^k\right) + c_2 r_2 \left(g^* - x_i^k\right) \quad (10)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \quad (11)$$

where w is the *inertia* parameter that weights the previous particle's momentum; c_1 and c_2 are the *cognitive* and *social* parameter of the particles multiplied by two random numbers r_1 and r_2 uniformly distributed in $[0-1]$, and are used to weight the velocity respectively towards the particle's personal best, $\left(p_i - x_i^k\right)$, and towards the global best solution, $\left(g^* - x_i^k\right)$, found so far by the whole swarm. Then the new particle position is determined simply by adding to the particle's current position the new computed velocity, as shown in Figure 2.

The PSO coefficients that need to be determined are the inertia weight w, the cognitive and social parameters c_1 and c_2, and the number of particles in the swarm.

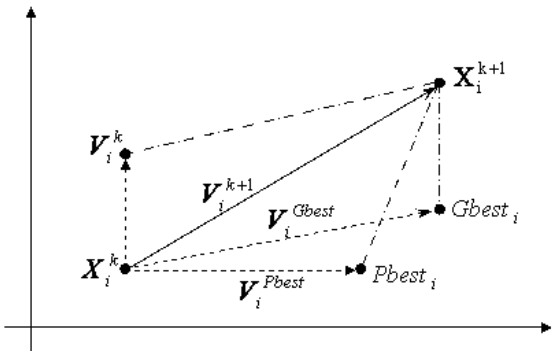

Figure 2. New particle position in PSO

We can interpret the motion of a particle as the integration of Newton's second law, where the components $c_1 r_1(p_i - x_i^k) + c_2 r_2(g^* - x_i^k)$ are the attractive forces produced by springs of random stiffness, while w introduces a virtual mass to stabilize the motion of the particles, avoiding the algorithm to diverge, and is typically a number such that $w \approx [0.5 - 0.9]$. It has been shown, without loss of generality, that for most general problems the number of parameters can even be reduced by taking $c_1 = c_2 \approx 2$.

D. Quantum Particle Swarm Optimization

Although much simpler to formulate than GA, classical PSO has still many control parameters and the convergence of the algorithm and its ability to find a near-best global solution is greatly affected by the value of these control parameters. To avoid this problem a variant of PSO, called Quantum PSO (QPSO) was formulated in 2004 by Sun et al. [12], in which the movement of particles is inspired by quantum mechanics.

The rationale behind QPSO stems from the observation that statistical analyses have demonstrated that in classical PSO each particle i converges to its local attractor a_i defined as

$$a_i = (c_1 p_i + c_2 g^*)/(c_1 + c_2) \quad (12)$$

where p_i and g^* are the personal best and global best of the particle. The local attractor of particle i is a stochastic attractor that lies in a hyper-rectangle with p_i and g^* being two ends of its diagonal, and the above formulation can also be rewritten as

$$a_i = r p_i + (1 - r) g^* \quad (13)$$

where r is a uniformly random number in the range $[0 - 1]$.

In classical PSO, particles have a mass and move in the search space by following Newtonian dynamics and updating their velocity and position at each step. In quantum mechanics, the position and velocity of a particle cannot be determined simultaneously according to uncertainty principle. In QPSO, the positions of the particles are determined by the Schrödinger equation where an attractive potential field will eventually pull all particles to the location defined by their local attractors. The probability of particle i appearing at a certain position at step $k + 1$ is given by:

$$x_l^{k+1} = a_i + \beta \left| x_{mbest}^k - x_l^k \right| \ln(1/u), if\ v \geq 0.5 \quad (14)$$

$$x_l^{k+1} = a_i - \beta \left| x_{mbest}^k - x_l^k \right| \ln(1/u), if\ v < 0.5 \quad (15)$$

where u and v are uniformly random numbers in the range $[0 - 1]$, x_{mbest}^k is the mean best of the population at step k defined as the mean of the best positions of all particles

$$x_{mbest}^k = \sum_{i=1}^{N} p_i \quad (16)$$

β is called *contraction-expansion* coefficient and controls the convergence speed of the algorithm.

The QPSO algorithm has been shown to perform better than classical PSO on several problems due to its ability to better explore the search space and also has the nice feature of requiring one single parameter to be tuned, namely the β coefficient. The exponential distribution of positions in the update formula makes QPSO search in a wide space.

Moreover, the use of the mean best position x_{mbest}, each particle cannot converge to the global best position without considering all other particles, making them explore more thoroughly around the global best until all particles are closer. However, this may be both a blessing and a curse; it may be more appropriate in some problems but it may slow the convergence of the algorithm in other problems. Again, there is a very fine balance between exploration and exploitation. How large is the search space, and how much time is given to explore before returning a solution.

E. Dealing with Constraints

Many real world optimization problems have constraints, for example, the available amount of certain resources, the boundary domain of certain variables, etc. So an important question is how to incorporate constraints in the problem formulation.

In some cases, it may be simple to incorporate the feasibility of solutions directly in the formulation of a problem. If we know the boundary domain of a certain dependent variable and the proposed solution violates such domain we can either reject the solution or modify it by constraining the variable within the boundaries. For example, suppose a time variable must satisfy the time interval between 9:00 and 13:00, while the proposed solution would place it at 14:34. One way to deal with the above violation is to constrain the variable to its upper bound (UB) 13:00 and reevaluate the objective function. This will be probably worse than before, but at least it will be feasible and need not be rejected altogether.

A common practice is to incorporate constrains directly in the formulation of the objective function through the

addition of a *penalty* element so that a constrained problem becomes unconstrained. If $f(x)$ is the objective function to be minimized, any equality / disequality constrains can be cast to penalty terms linearly added to the objective function, typically with a high weight w and a quadratic function in the measured violation $g(x) = \max(0, v(x)^2)$, where $v(\cdot)$ "measure" the amount of violation. Now the augmented optimization problem becomes

$$\arg\min_x (f(x) + w \cdot g(x)) \quad (17)$$

w is the penalty weight that needs to be large enough to skew the choice of the fittest solutions towards the smallest penalty component, typically in the range 10^9 - 10^{15}.

In our scheduling problem we have already defined the max power constraint as upper bound inequality.

$$g(t) \stackrel{\text{def}}{=} \max\left[0, \left(\sum_{i=1}^{N} \sum_{j=1}^{np_i} allocPower_{ij}(t) \right) - maxPower\right] \quad (18)$$

F. Nature Inspired Random Walks and Lévy Flights

A random walk is a series of consecutive random steps starting from an original point: $x_n = s_1 + \cdots + s_n = x_{n-1} + s_n$, which means that the next position x_n only depends on the current position x_{n-1} and the next step s_n. This is the typical main property of a Markov chain. Very generally, we can write the position in random walks at step $k+1$ as

$$x_{k+1} = x_k + s\sigma_k \quad (19)$$

where σ_k is a random number drawn from a certain probability distribution. In mathematical terms, each random variable follows a probability distribution. A typical example is the normal distribution and the random walk becomes a *Brownian* motion. Besides the normal distribution, the random walk may obey other non-Gaussian distributions.

For example, several studies have shown that the random walk behavior of many animals and insects have the typical characteristics of the *Lévy* probability distribution and the random walk is called a Lévy flight [13][14][15]. The Lévy distribution has the characteristic of being both stable and heavy-tailed. A stable distribution is such that any sum n of random number drawn from the distribution is finite and can be expressed as

$$\sum_{i=1}^{n} x_i = n^{1/\alpha} \cdot x \quad (20)$$

where α is called the index of stability and controls the shape of the Lévy distribution with $0 < \alpha \leq 2$. Notably, two value for α are special cases of two other distribution, the normal distribution for $\alpha = 2$, and the Cauchy distribution for $\alpha = 1$.

The heavy-tail characteristic implies that the Lévy distribution has an infinite variance, decaying for large x to $\lambda(x) \sim |x|^{-1-\alpha}$.

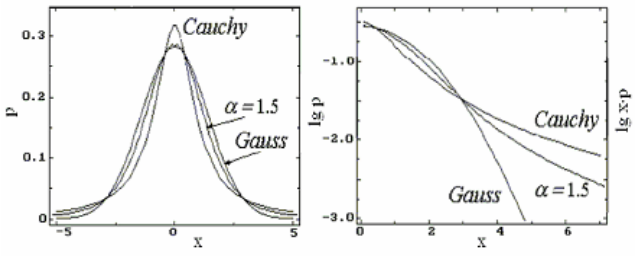

Figure 3. Cauchy

Figure 3 shows the shapes of the normal, Cauchy, and Lévy distribution with $\alpha = 1.5$. The difference becomes more pronounced in the logarithmic scale showing the asymptotic behavior of the Lévy and Cauchy distribution compared with the normal.

Due to the stable property, a random walker following the Lévy distribution will cover a finite distance from its original position after any number of steps. Also, due to the heavy-tail of the distribution, extremely long jumps may occur, and typical trajectories are self-similar, on all scales showing clusters of shorter steps interspersed by long excursions, as shown in Figure 4. In fact, the trajectory of a Lévy flight has fractal dimension $d_f = \alpha$.

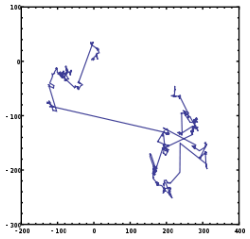

Figure 4. Levy's flight

In that sense, the normal distribution in Figure 5 represents the limiting case of the basin of attraction of the generalized central limit theorem for $\alpha = 2$ and the trajectory of the walker follows a Brownian motion.

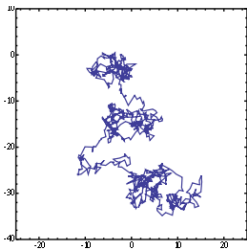

Figure 5. Brownian path

Due to the properties of being both stable and heavy-tailed, it is now believed that the Lévy distribution nicely describes many natural phenomena in physical, chemical, biological and economical systems. For instance, the foraging behaviors of bacteria and higher animals show typical Lévy flights, which optimize the search compared to Brownian motion giving a better chance to escape from local optima.

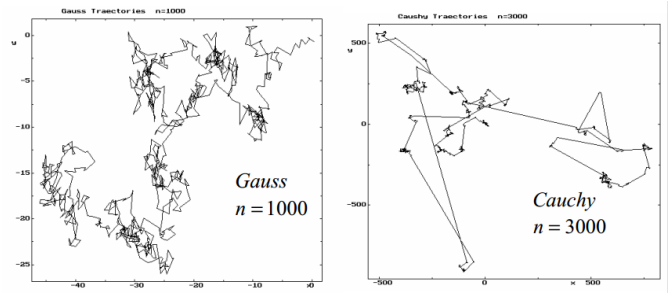

Figure 6. the trajectories of a Gaussian (left) and a Lévy (right) walker

Figure 6 shows the trajectories of a normal (left) and a Lévy (right) walker. Both trajectories are statistically self-similar, but the Lévy motion is characterized by island structure of clusters of small steps, connected by long steps.

G. Step Size in Random Walks.

In the general equation of a random walk $x_{k+1} = x_k + s\sigma_k$, a proper step size, which determines how far a random walker can travel after k number of iterations, is very important in the exploration of the search space. The two component that make up the step are the scaling factor *s* and the length of the random number in the distribution σ_k. A proper step size is very important to balance exploration and exploitation, too small a step and the walker will not have a chance to explore potential better places, on the other hand, too large steps will scatter the search from the focal best positions. From the theory of isotropic random walks, the distance traveled after k steps in N dimensional space is

$$D = s\sqrt{kN}\,. \tag{21}$$

In a length scale L of a dimension of interest, the local search is typically reasonably limited in the region $D = L/10$, which means that the scaling factor

$$s \approx \frac{L}{10\sqrt{kN}} \tag{22}$$

In typical metaheuristic optimization problems, we can expect the number of iterations k in the range 100 – 1000.

For example, with 100 iterations and $N = 1$ (a one dimensional problem) we have $s = 0.01L$, and to another extreme with 1000 iterations and $N = 10$ we have $s = 0.001L$. Therefore, a scaling factor between 0.01 – 0.001 is basically a reasonable choice in most optimization problems. L is still kept independent as each dimension of the problem may very well have a very different length scale.

V. RANKED PARTICLE SWARM WITH LÉVY FLIGHTS

In this section, we describe a variant of the QPSO, named Ranked PSO with Lévy flights (RaPSOL) that introduces some innovative strategies on the QPSO borrowed from other disciplines, like observations of natural phenomena, and the nice properties of ranking in descriptive statistics the nonparametric measures of dependence, namely Spearman's rho and Kendall's tau.

The result is an algorithm that provides a nice balance between exploration and exploitation and gives good-quality solutions in short time and with limited computing power.

In fact, the Home Gateway (HG) is a low power ARM embedded system running a Java Virtual Machine in the OSGi framework.

The first innovation is to replace the exponential probability density function with the Lévy distribution. A second innovation is to improve the global exploration search by shifting the attention from just the single best global leader, to all the ranked particles. In fact, one shortcoming of standard leader-oriented swarm algorithms is that they tend to converge very fast to the current best solution, sometimes missing other promising search area. With only one global leader, all particles quickly converge together, something missing better solutions.

To overcome that shortcoming, in RaPSOL, particles are ranked according to their fitness and instead of just considering the global best particle for determining the current attractor, any particle is entitled to choose any other better particle, not just the global best. This selection is uniform-random: the second best particle is only entitled to choose the best particle, and in general each particle may choose any other better particle as its current attractor.

The introduction of ranked selection is enough to guarantee a broader search in the problem domain avoiding premature convergence to local optima. The algorithm steps are thus:

1. Rank all particles according to their current fitness.
2. For each particle, randomly select any particle whose fitness is better than this particle's. Name such particle the relative leader.
3. Take a uniform random point in the linear hyperplane that intersects the particle's personal best position and the relative leader. Name this point the particle's attractor
4. Do a Lévy flight from the attractor with a step-size proportional to the swarm's current radius, by constraining on the current distance of the particle and the relative leader.

From our experiments and simulations, the effect of ranking, coupled with the Lévy distribution, has proven to

exhibit very good results compared to traditional PSO and QPSO.

For our purposes, the Lévy distribution coefficient α chosen in RaPSOL is actually the Cauchy coefficient $\alpha = 1$.

The Cauchy random generator is much simpler than the more general algorithm for Lévy generation and that is a determining factor in runtime execution. Since the random generation needs to be executed for an umpteen number of times (i.e., the dimension of the problem, by the number of particles in the swarm, by the number of iterations of the algorithm), the computing speed of the random generation is of paramount importance. From our experiments, within a given time limit allotted to the algorithm to find a solution, the Cauchy version of the algorithm is able to execute almost twice the number of iterations than the general Lévy version. Therefore, even if there was an optimal coefficient α that provides better results for the same number of iterations, it will be outperformed by the Cauchy variant that with more allowed iterations finds better solutions.

VI. SIMULATION AND RESULTS

We ran a number of simulations modeling the same scheduling problem both in the RaPSOL algorithm and a pure mathematical model with commercial linear programming (LP) solvers, namely XPress and CPLEX.

The scheduling problem was formalized with 4 instances of washing-machine power profiles, each profile being made of 4 phases, and 3 instances of dish-washing-machines each made of 5 phases, for a total of 31 independent variables to optimize in the scheduling problem instance.

Table I. Comparison of different tested algorithms over dataset described in the first two columns.

#Appl	Max Power	TiLim	XPRESS Cost	CPLEX Cost	SYMPHONY Cost	RaPSOL Cost
3	4000	10 s	2,5384	2,5381	3,2278	2,5381
		60 s	2,5381	2,5381	2,5381	2,5381
3	3000	10 s	2,5384	2,5381	3,2278	2,5381
		60 s	2,5381	2,5381	2,5381	2,5381
3	2000	10 s	2,5381	2,5381	2,5425	2,5381
		60 s	2,5381	2,5381	2,5381	2,5381
5	4000	10 s	4,8643	4,8643	4,8870	4,8643
		60 s	4,8643	4,8643	4,8685	4,8643
5	3000	10 s	4,8643	4,8671	4,8925	4,8643
		60 s	4,8643	4,8643	4,8925	4,8643
5	2000	10 s	4,9700	5,2445	–	4,9551
		60 s	4,9700	4,9545	–	4,9551
10	4000	10 s	12,8896	9,7318	–	9,1123
		60 s	9,1929	9,0141	–	9,0149
10	3000	10 s	12,3398	12,1620	–	9,6051
		60 s	10,0963	9,2056	–	9,3841
10	2000	10 s	11,4725	11,9863	–	11,022
		60 s	11,4561	11,6960	–	10,809
15	4000	10 s	–	19,2274	–	14,906
		60 s	15,0148	15,0497	–	14,681
15	3000	10 s	–	18,0702	–	16,347
		60 s	16,3381	16,8535	–	16,172

Due to the hard problem space for the brute-force exact algorithms, the scheduling horizon was limited to 12 hours and the time slots at multiples of 3 minutes, otherwise, with one-minute slot time, no feasible solutions were found even in 7 days of uninterrupted run.

Running 96 hours, XPress found a solution at a cost of € 2.57358. With the same problem and running 1 hour CPLEX found a solution at € 2.59123. Finally, the RaPSOL was given a bound time of 15 seconds, and run 10 times to have reliable statistics, finding a best solution at € 2.7877, with an average cost of € 2.9351 for the 10 times. We additionally report in Table I results of compared algorithms over an extended dataset (described in the first two columns), where missing entries mean that the target algorithm has not been able to achieve any result in the specified time limit.

The results obtained using linear programming and exact solvers are very important as they fix theoretical optima for benchmarking the convergence and performance of the metaheuristic approach of the RaPSOL. Results show that although RaPSOL finds a worse solution than the theoretical optimum by a 8 – 13 %, the very short allotted time to find a solution is anyway a very promising approach. In Figure 7 and Figure 8 are reported simulation results when considering appliance scheduling with constant overload threshold, variable tariff, and with the absence and presence of photovoltaic generation respectively. An interesting use case is the scheduling of an entire apartment building where tenants share a common contract with the utility provider in which the energy consumption of the apartment house as a whole must be below a given "virtual" threshold that changes in time. Figure 9 shows such scenario. The curved red line represents the virtual threshold that the apartment house should respect.

All energy above such threshold will not cause an overload but its cost grows exponentially with the net effect of encouraging a peak shaving of profile allocation. The case study of Figure 10 is a scheduling of 15 apartments, with 3 appliances each, for a total of 45 appliances. The apartment house is also provided with common PV-panels.

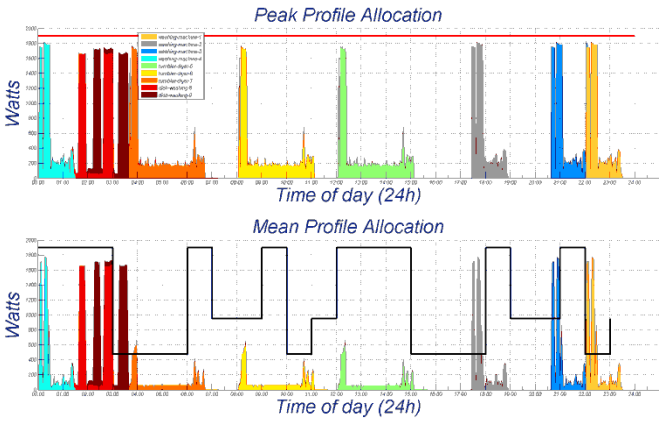

Figure 7. RAPSOL simulation results: appliance scheduling with constant overload threshold, variable tariff, no photovoltaic.

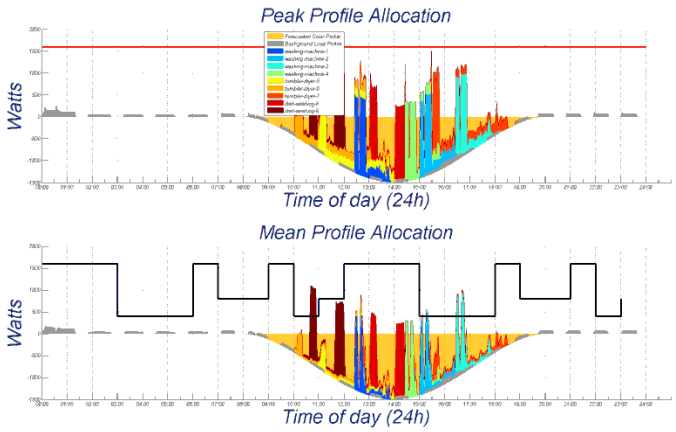

Figure 8. RAPSOL simulation results: appliance scheduling with constant overload threshold, variable tariff, photovolltaic.

The 3 case studies described here show the remarkable flexibility of the RaPSOL algorithm, and many other metaheuristic algorithms for that matter, i.e., the ability to adapt the algorithm to the unique attributes of a given problem and not based on predefined characteristics.

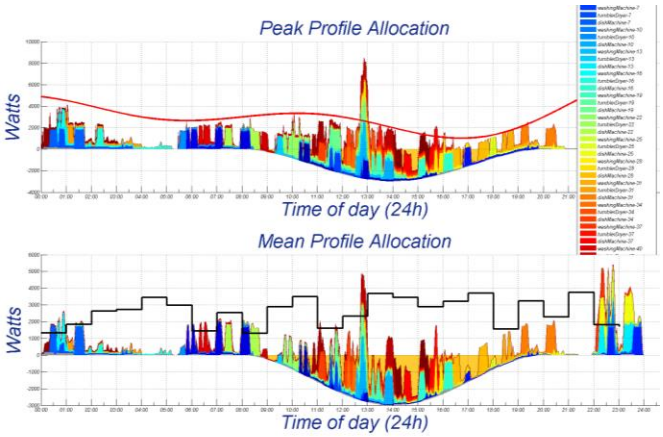

Figure 9. RaPSOL simulation results: appliance scheduling for different apartments with variable overload threshold, variable tariff, photovoltaic.

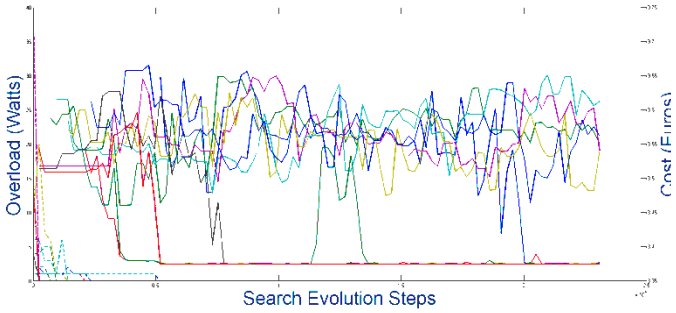

Figure 10. RaPSOL simulation results: overload avoidance and optimization of cost

A. *Extended simulations setups*

Extended simulations with a number of appliances equal to 50, overload threshold of 4kW and different tariffs schemes are shown in the following figures. The different conditions comprise:

- tariffs (in the lower part of each figure) highly dynamic or three tiers:
- solar generation: photovoltaic generation with clear sky conditions (present or not present).

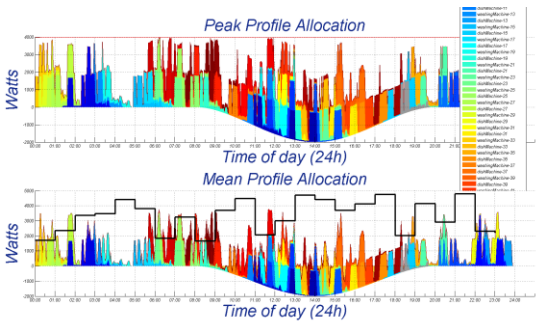

Figure 11. RaPSOL simulation results for the case: 50 appliances, overload threshold of 4kW, dynamic tariff, photovoltaic with clear sky condition

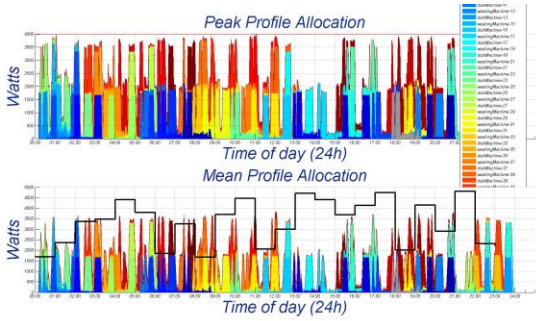

Figure 12. RaPSOL simulation results for the case: 50 appliances, dynamic tariff, overload threshold of 4kW, no photovoltaic

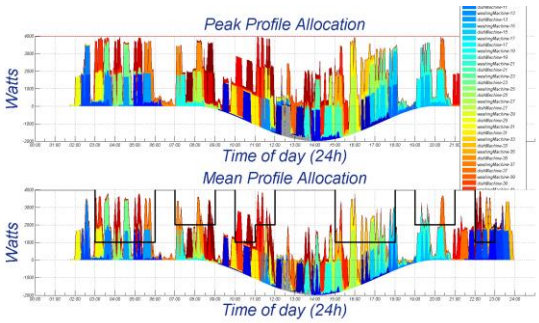

Figure 13. RaPSOL simulation results for the case: 50 appliances, overload threshold of 4kW, three-tiers tariff, photovoltaic with clear sky condition

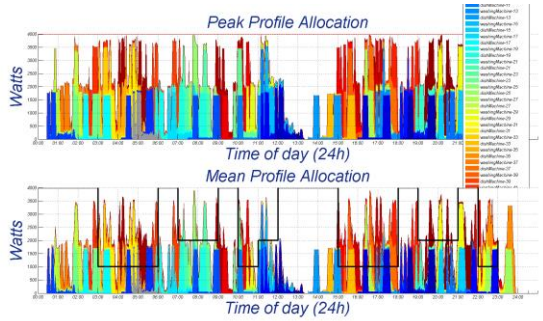

Figure 14. RaPSOL simulation results for the case: 50 appliances, three-tiers tariff, overload threshold of 4kW, no photovoltaic

B. Comparison with growing number of appliances

In order to verify the performances of RaPSOL considering a growing number of appliances, different simulations have been performed and compared in Figure 15. The cost normalized for a single appliance is shown. Clearly, the case with no solar generation has higher cost. As expected, increasing the number of appliances also downgrade the final solution because of the increased dimension and complexity of the optimization.

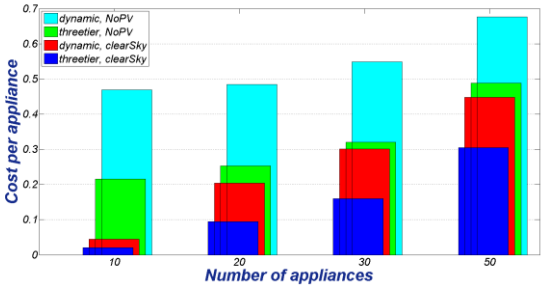

Figure 15. RaPSOL simulation results for a growing number of appliances for the different tariffs, overload threshold of 4kW, photovoltaic or no photovoltaic

C. Discussion and considerations

In a rapidly changing world, algorithmic paradigms that are flexible and easy to adjust offer a competitive advantage over rigid, tailor based methods. In such volatile domains, the usefulness of an algorithm framework will not be given by its ability to solve a static problem, rather its ability to adapt to changing conditions. Such requirement is likely to define the success or failure in optimization algorithms of tomorrow.

Exact and formal techniques decompose the optimization problems into mathematically tractable problems involving precise assumptions and well-defined problem classes. However, many practical optimization problems are not strictly members of these problem classes, and this becomes especially relevant for problems that are non-stationary during their lifecycle. Traditional deterministic techniques place constraints on the current problem definition and on how that problem definition may change over time. Under these circumstances, long-term algorithm survival / popularity is less likely to reflect the performance of the canonical algorithm and instead more likely reflects success in algorithm design modification across problem contexts [16].

VII. Conclusion

This work describes an innovative Ranked PSO with Lévy flights metaheuristic algorithm for scheduling home appliances, capturing all relevant appliance operations. With appropriately dynamic tariffs, the proposed framework can propose a schedule for achieving cost savings and overloads prevention. Good quality approximate solutions can be obtained in short computational time with almost optimal solutions.

The proposed framework can easily be extended to take into account solar power forecasting in the presence of a residential PV system by simply adapting the objective function and using the solar energy forecaster as further input to the scheduler.

Acknowledgment

This work has been partially supported by INTrEPID, INTelligent systems for Energy Prosumer buildIngs at District level, funded by the European Commission under FP7, Grant Agreement N. 317983.

The authors would like to thank Prof. Della Croce of Operational Research department of the Politecnico di Torino for the valuable insights and contribution on the linear programming solvers.

References

[1] E. Grasso, C. Borean, “QPSOL: Quantum Particle Swarm Optimization with Levy’s Flight,” ICCGI 2014, The Ninth International Multi-Conference on Computing in the Global Information Technology, pp. 14-23.

[2] INTrEPID FP7 project, “INTelligent systems for Energy Prosumer buildings at District level,” http://www.fp7-intrepid.eu .

[3] J. W. Taylor “Short-Term Load Forecasting with Exponentially Weighted Methods,” IEEE Transactions on Power Systems, vol. 27, pp. 458-464, February 2011.

[4] N. Sharma, J. Gummeson, D. Irwin, and P. Shenoy, “Cloudy Computing: Leveraging Weather Forecasts in Energy Harvesting Sensor Systems,” SECON 2010, Boston, MA, June 2010.

[5] Energy@Home project, “Energy@Home Technical Specification version 0.95,” December 22, 2011

[6] R. Kolisch and S. Hartmann, “Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis,” . in J. Weglarz, editor, Project scheduling: Recent models, algorithms and applications, pp. 147–178, Kluwer Academic Publishers, 1999.

[7] R. Kolisch and S. Hartmann, “Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update,” European Journal of Operational Research 174, pp. 23-37, Elsevier, 2006.

[8] K. Cheong Sou, J. Weimer, H. Sandberg, and K. Henrik Johansson, "Scheduling Smart Home Appliances Using Mixed Integer Linear Programming," 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC), Orlando, FL, USA, December 12-15, 2011.

[9] J. Holland, "Adaptation in Natural and Artificial systems," University of Michigan Press, Ann Anbor, 1995.

[10] S. Kirkpatrick, C. D. Gellat, and M.P. Vecchi, "Optimization by Simulated Annealing," Science, 220, pp. 671-680, 1983.

[11] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," in: Proc. of the IEEE Int. Conf. on Neural Networks, Perth, Australia, pp. 1942-1948, 1995.

[12] J. Sun, B. Feng, and W. Xu, "Particle swarm optimization with particles having quantum behavior," in IEEE Congress on Evolutionary Computation, pp. 325-31, 2004.

[13] X. Yang, "Nature-Inspired Metaheuristic Algorithms," Luniver Press, 2008.

[14] X. Yang "Review of metaheuristics and generalized evolutionary walk algorithm," Int. J. Bio-Inspired Computation, vol. 3, No. 2, pp. 77-84, 2011.

[15] A. Chechkin, R. Metzler, J. Klafter, V. Gonchar, "Introduction to the theory of lévy flights." In: Klages R, Radons G, Sokolov IM (eds) Anomalous Transport: Foundations and Applications, Wiley-VCH, Berlin, 2008.

[16] J. M. Whitacre "Survival of the flexible: explaining the recent dominance of nature-inspired optimization within a rapidly evolving world," Journal Computing, Vol. 93, Issue 2-4 , pp 135-146 2009.