QPSOL: Quantum Particle Swarm Optimization with Levy's Flight

Optimization of appliance scheduling for smart residential energy grids

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

***Abstract*— This paper considers the minimum electricity cost scheduling problem of smart home appliances in the context of smart grids. Functional characteristics, such as expected duration and peak power consumption of the smart appliances can be adjusted 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 Quantum inspired Particle Swarm with Lévy flights. The algorithm runs on the limited computational power provided by the home gateway device and in almost real-time as of user perception.**

Keywords: scheduling, swarm intelligence, methaeuristic smart grids.

I. INTRODUCTION

This paper considers the minimum electricity cost scheduling problem of smart home appliances in the context of the Energy@Home international project [1]. Functional characteristics, such as expected duration and peak 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 Quantum inspired Particle Swarm with Lévy flights. The algorithm runs on the limited computational power provided by the home gateway device and in almost real-time as of user perception.

The innovative Quantum inspired Particle Swarm Optimization (QPSO) with Lévy flights metaheuristic algorithm for scheduling home smart appliances, capturing all relevant appliance operations, is not only described in the paper but also validated, since the results of the implementation of it running on an embedded platform are presented. With appropriately dynamic tariffs and short-term load forecasting, the proposed framework can calculate and propose a schedule for achieving high cost savings and overloads prevention, improving the user experience of energy management services. Good quality approximate solutions can be obtained in a short amount of computation time, in the order of about 2 seconds an almost optimal approximate solution can be obtained, which enables the usage of this algorithm on very embedded and low cost platforms .

It is also described in the paper how the proposed framework could be extended to incorporate solar power forecasting in the presence of a residential Photovoltaic (PV) system by tuning the objective function and using the solar energy forecaster as further input to the scheduler ([16], [18]).

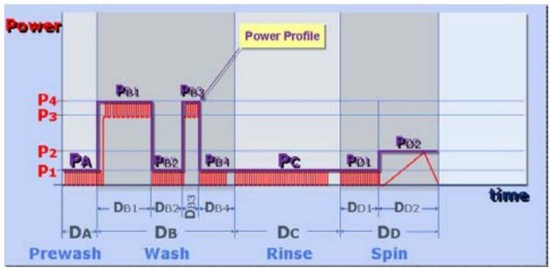

Figure 1. Example Power Profile with its phases generated by a washing machine

The paper is structured as it follows. Section 2 describes and models the problem of scheduling of smart appliance. Section 3 highlights how this problem can be classified as a NP-Hard Combinatorial Optimization Problem. In Section 4 a review of metaheuristic algorithms is performed while in Section 5 the new QPSOL algorithm proposed in the paper is described. Section 6 reports the results of the simulations of the QPSOL algorithm applied to the problem of scheduling of smart appliance while Section 7 presents the conclusions and future work.

II. SCHEDULING PROBLEM OF SMART HOME APPLIANCES

Europe has set the ambitious target of a 20% share of overall energy demand to be supplied from renewable energy by 2020. In order to achieve this target, the share of renewable energy will need to increase to some 35%. Most of the increase will come from wind and solar energy, which are both fluctuating resources by nature.

Electricity consumption varies between different hours of the day, between days of the week, and between seasons of the year. In recent years, the power demand has reached new peak levels and environmental / economic reasons will require more complex power balance scenarios also based on the introduction of residential renewable electricity generation to reduce the carbon footprint and CO2 emission.

One of the major challenges associated with this drastic restructuring of the energy supply with renewables is how electricity networks can cope with the extreme variability of wind and solar energy production. In the past, the ideal load curve was flat in order to allow for the full load operation of conventional power plants. In the future, the ideal demand needs to be variable in order to adapt to the current production from renewable energy sources.

It is expected in the near future that time-varying and dynamic electricity tariffs will increase popularity, especially for the reduction of peak power consumption which are the most detrimental from the grid operators. However, such load balancing is only feasible if consumers are both able and willing to consider tariff information, but it is still unrealistic to expect most consumers to identify the most economical operations of their appliances with dynamic tariffs, or in the presence of a small-scale photovoltaic (PV) power generation system which adds even more complexity in determining the economic convenience between immediate power usage versus selling the power to the main grid operator.

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 Energy@Home

The Energy@Home (E@H) consortium is based on ZigBee communication between smart appliances in a home mesh wireless network [1]. The "core" element of this 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 ahead, based on the (possibly) varying tariff of the day, the forecasted energy power consumption, and also the forecasted home PV power generation, if available.

The proposed scheduling framework is based on the Power Profile Cluster defined in the E@H specifications [1], which specifies that each appliance operation process is modeled as a Power Profile divided into a set of sequential energy phases, as presented in Figure 1. An energy phase is an uninterruptible logic subtask of the appliance operation, which uses a pre-specified amount of electric energy. The energy phases are sequential since the next phase cannot start until the previous phase is completed, e.g., a washing machine agitator cannot start until the basin is filled with water.

In addition to having a specified energy usage, each energy phase is characterized by peak maximum power, a specific duration, and a possible maximum activation delay after the end of the previous phase. Some phases cannot be delayed and must start soon after the previous phase completes (maximum delay is zero). Other phases may be delayed adding extra flexibility in the scheduling of the Power Profile, e.g., the washing machine agitator must start within ten minutes of the basin being filled.

In addition, the scheduler needs to take into account user specified time preferences, requiring that certain appliances should be run within some particular time intervals, e.g., the dishwasher must complete washing dishes 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.

The scheduling execution interval is divided into 1440 1-minute time slots for a 24-hour period. The number of appliances considered for scheduling is denoted N, and the number of energy phases for each appliance is denoted ni for i = 1, 2,.., N. The problem dimension, i.e., the number of independent variables that make up the problem, is

$$\sum_{i=1}^{N} \sum_{j=1}^{ni} p_{ij} \tag{1}$$

where pij is the jth phase of power profile i. The objective of the scheduler is to minimize the total electricity cost for operating the appliances based on a given 24-hour ahead electricity tariff while taking into account user comfort criteria (earlier executions are preferable than delayed execution) and respecting time and energy constraints.

B. Modeling Time and Energy Constraints

Even in the presence of the HG scheduler controlling a set of smart appliances, the real number of home appliances and other electric powered devices that consume energy in the hose is higher and outside the control of the scheduler. For that reason any sensible scheduling system must be complemented by an appropriate forecasting module that provides good estimation of the overall power consumption based on past statistics.

The energy constraints imposes that for each time slot, the total sum of power required by all phases running in that slot, plus the forecasted power consumption "outside" the control of the scheduler, be less than peak power threshold provided by the grid operator,

$$\sum_{i=1}^{N} \sum_{j=1}^{ni} \mathrm{Power}(p_{ij}) + \mathrm{Load}_{forecast} \le \mathrm{PeakPower} \tag{2}$$

Time constraints are twofold. On the one hand the user can set up time preference constraints, specifying the time interval that a particular appliance must be scheduled in terms of an earliest start time (EST), e.g., after 13:20, and a latest end time (LET), e.g., before 18:00.

$$\mathrm{EST} \le PP_i^{start} \le PP_i^{end} \le \mathrm{LET} \tag{3}$$

where PP^i_{start} and PP^i_{end} are respectively the scheduling start slot of the 1st phase of Power Profile *i* and the end slot of the last phase.

The second time constraint is the maximum activation delay of each of the sequential phases that make up the Power Profiles. 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.

$$p_{ij}^{end} \leq p_{ij+1}^{start} \leq p_{ij}^{end} + p_{ij+1}^{maxdelay} \quad (4)$$

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 [12] [13].

Moreover, the presence of time constraints introduce 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 user wants a perceived immediate answer), make the formulation of the problem and its solution a challenging endeavor.

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 really important for the great amount of real-life applications that can be modeled in this way. For example, areas like routing or scheduling contain plentiful hard challenges that can be expressed as a combinatorial optimization problem.

For easy problems, exact methods can be exploited, such as Branch&Bound 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, [2] 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 or B&B is hardly acceptable for scheduling home appliances, and other more efficient methods need to be investigated.

A. *Convex Constraints and Smooth Objective Functions*

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

Convex constraints form a series of convex regions where exact methods could be applied (e.g., B&B, linear-programming, etc.). 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 convex. 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.

Most “real” problems are neither convex nor smooth, so traditional exact methods cannot be applied. Finding a solution not the best one but “acceptable”, or even finding a feasible solution is NP-Hard.

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 (GRASP), 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 next 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 [3]. 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 [5] 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 global minimum.

C. Particle Swarm Optimization

Particle swarm optimization (PSO), introduced in 1995 by American social psychologist James Kennedy, and engineer Russell C. Eberhart [6], 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 found solutions and also share the global best so as to focus on the quality solutions. 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 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:

$$v_i^{k+1} = w\, v_i^k + c_1 r_1 (p_i - x_i^k) + c_2 r_2 (g^* - x_i^k) \quad (5)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \quad (6)$$

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, $(p_i - x_i^k)$, and towards the global best solution, $(g^* - x_i^k)$, 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.

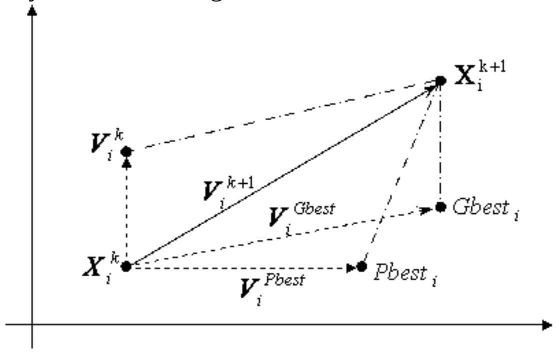

Figure 2. New particle position in PSO

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. We can interpret the motion of a particle as the integration of Newton's second law where the component $c_1 r_1 (p_i - x_i^k) + c_2 r_2 (g^* - x_i^k)$ 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 and al. [7], 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

$$a_i = (c_1 p_i + c_2 g^*) / (c_1 + c_2) \quad (7)$$

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

$$a_i = r\, p_i + (1 - r)\, g^* \quad (8)$$

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:

$$x_i^{k+1} = a_i + \beta\, |x_{mbest}^k - x_i^k| \ln(1 / u), \text{ if } v \geq 0.5 \quad (9)$$
$$x_i^{k+1} = a_i - \beta\, |x_{mbest}^k - x_i^k| \ln(1 / u), \text{ if } v < 0.5 \quad (10)$$

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

$$x_{mbest} = (1 / N) \sum_{i=1}^{N} p_i \quad (11)$$

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 / 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 second way is to incorporate the 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, and subject to the constraints *x* in domain [x_{lower}, x_{upper}], we rewrite the objective function as

$$f'(x) = f(x) + \sum_{i=1}^{N} w_i \, g_i(x) \qquad (12)$$

$g_i(x)$ measure the amount of constraint violation and is zero if *x* is within the domain boundaries, or it is some function *g* ($x - x_{lower}$), ($x_{upper} - x$) otherwise. w_i are the penalty weights 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}.

Note that the two approaches described above to deal with constraints need not be mutually exclusive and can both be incorporated in the formulation of a problem; some constraints may very well be modeled with the first method, while other are modeled with the penalty method.

F. *Nature Inspired Random Walks and Lévy Flights*

A random walk is a series of consecutive random steps starting from an original point: $x_n = s_1 + \ldots + s_n = x_{n-1} + s_n$, 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

$$x^{k+1} = x^k + s\,\sigma_k \qquad (13)$$

where σ_k is a random number drawn from a certain probability distribution. In mathematical terms, each random variable follows a probability distribution, for example a *Gaussian* (normal) distribution is the most well-known because many physical phenomena obey this distribution and the random walk becomes the *Brownian* motion. But if the step length obeys other non-Gaussian distributions we have to deal with a more generalized form of random walks.

Various 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 Lévy fight [8] [9] [10]. The Lévy distribution has the nice mathematical feature 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

$$\sum_{i=1}^{n} x_i = n^{1/\alpha}\, x \qquad (14)$$

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 Gaussian 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 at large *x* to $\lambda(x) \sim |x|^{-1-\alpha}$

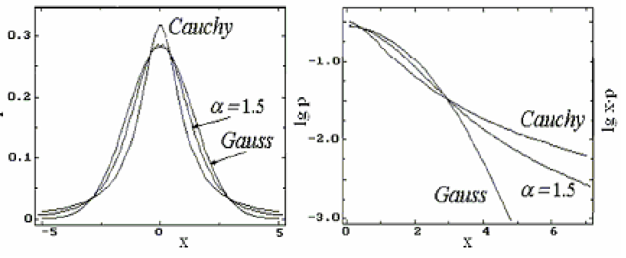

Figure 3. Cauchy

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

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. But also due to the heavy-tail (divergence of the variance), 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 $d_f = \alpha$.

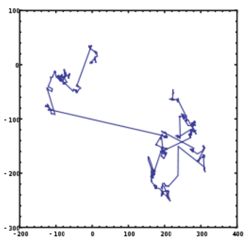

Figure 4. Levy's flight

In that sense, the Gaussian distribution in Figure 5. represents the limiting case of the basin of attraction of the so-called generalized central limit theorem for α = 2 and the motion of the walker follows a Brownian path.

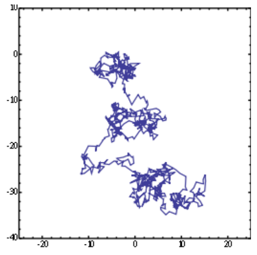
Figure 5. Brownian path

Due to the remarkable properties of stable, heavy-tailed distributions it is now believed that the Lévy statistics provides a framework for the description of many natural phenomena in physical, chemical, biological, economical systems from a general common point of view. For instance, the foraging behavior of bacteria and higher animals relies on the advantages of Lévy distributed excursion lengths, which optimize the search compared to Brownian search giving a better chance to escape from local optima.

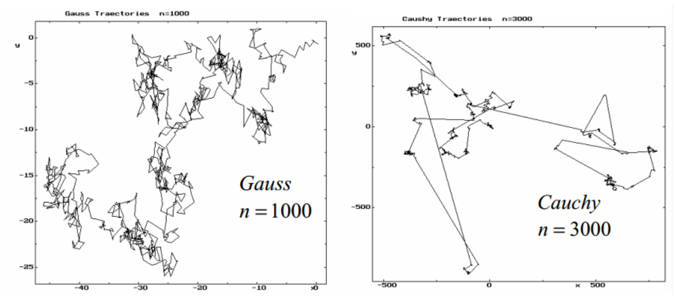

Figure 6. the trajectories of a Gaussian (left) and a Lévy (right) walker

The Figure 6. above shows the trajectories of a Gaussian (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

$$D = s \cdot \sqrt{k\,N} \qquad (15)$$

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

$$s \approx 0.1\ L / \sqrt{k\,N} \qquad (16)$$

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 = 0.01 L, and to another extreme with 1000 iterations and N=10 we have s = 0.001 L. 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. Quantum inspired Particle Swarm Algorithm with Lévy flights

After several experimental and simulated alternative metaheuristic approaches, we have come to the definition of a novel variant of the PSO algorithm that can be described as Quantum inspired PSO with Lévy flights (QPSOL). The algorithm tries to capture and exploit some of the best characteristics of various algorithms described in the previous sections. The result being an algorithm that provides a good balance between exploration and exploitation that gives quasi-optimal solutions within a very short time even 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 two main assumptions of the QPSOL algorithm are: first, as in Quantum PSO, particles have no mass and move around their attractor within a probability distribution. Secondly, rather than follow the quantum physics that uses the exponential distribution, in QPSOL particles move according to the nature-inspired Lévy distribution. From our experiments and simulations, the quantum inspired PSO, coupled with the Lévy distribution, has proven to outperform the classical PSO and traditional QPSO.

For our purposes, the Lévy distribution coefficient *α* chosen in QPSOL is actually the Cauchy coefficient α = 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. Since Cauchy is simply a special case of the general Lévy distribution, henceforth we will continue to refer to the algorithm as a Quantum PSO with Lévy flights QPSOL.

A. QPSOL for Scheduling Appliances

As any population based metaheuristic algorithm, each particle represent a complete solution to the problem, i.e., a complete schedule for all the Power Profiles of the appliances. Since each Power Profile is itself composed by a sequence of phases, we model each particle (complete solution) as a set of *N* sub-particles, where *N* is the number of Power Profiles and where each sub-particle represents the schedule for the energy phases of that Power Profile. Below we report the Java code of the evolution of the sub-particles in the swarm and it represents the core of the QPSOL algorithm.

The Lévy light of a sub-particle is a loop on the sequence of the energy phases. First, the maximum delay for each phase is determined. The maximum delay for the first phase (index == zero) is actually the maximum slack interval of the whole Power Profile as imposed by the user.

The maximum delay of each subsequent phase is the minimum between its maximum delay as per Power Profile specification, and the remaining slack for the whole remaining phases updated at each step after a phase is moved with the Lévy flight.

After calculating the maximum delay for each phase *i*, the Lévy flight is performed with these equations:

$$a_i = r\, p_i + (1 - r)\, g^* \qquad (17)$$

$$x_i = a_i + \beta\, (a_i - x_i)\, \lambda_i \qquad (18)$$

where *r* is a random number with uniform distribution in [0 - 1], λ_i is a random number with Cauchy (Lévy) distribution, and β is the *constriction coefficient* that controls the step size of the flight. Finally, the maximum delay constraint is enforced on the new position of the sub-particle to keep its feasibility by resetting the delay to zero if the flight exceed the allotted maximum delay.

Borrowing from QPSO, the attractor of the sub-particle a_i can be thought of as a point randomly chosen in the hyper-plane that connects the particle's best position and the global best position. This attractor is the next starting point for the Lévy flight, and the next equation updates the sub-particle position with the value of the attractor modified by the flight, which is itself a random number generated with Cauchy distribution multiplied by the value $(a_i - x_i)$, i.e., the difference between the attractor and the current position.

This value provides the scaling factor of the flight around the attractor and is crucial in the balance between exploration of new solutions and exploitation focusing in the proximity of the current solution.

On the other hand the *β* parameter need not be modified in the course of the algorithm and is tied to a probability density function "attitude" to generate large numbers, for instance with Cauchy (α = 1) we set β = 0.35, while with a general Lévy with α = 1.4, which we have found as a good Lévy coefficient, we set β = 0.75.

Finally, note that contrary to QPSO formulation, we always execute the random flight "away from" the current position x_i. In fact $(a_i - x_i)$ is indeed a signed value that provides the direction of the attractor away from the current position. We have found through experiments that this gives better results in the exploration, trying to explore away from current "beaten track".

B. Modeling Constraints and Objective Function

An important aspect of the algorithm is the formulation of the objective function and the time and energy constraints. As described before, feasibility time constraints are enforced directly when updating the particles positions within their lower and upper bound limits.

Energy constraints are instead formulated as penalty components of the objective function to be minimized, which is defined as the sum of three elements with their respective weighting coefficients:

minimize: $f(x) = w_1O(x) + w_2C(x) + w_3T(x)$

where w_1, w_2, and w_3 are the weight coefficients assigned respectively to the overload amount O(x), the energy cost C(x), and the tardiness in the execution T(x).

The overload is the penalty component: if there is power overload the constraint is violated and therefore all other components can be ignored as their contribution would me trifle to the whole objective function f(x). As such w_1 is chosen large enough to privilege constraints satisfaction before anything else, $w_1 \approx 10^9$.

w_2 is the weight of the energy cost and is normalized to the value 1. Finally the tardiness component is an added contributing element to the objective function and corresponds to perceived user comfort and tends to privilege schedule solutions that complete sooner rather than later (tardiness of the execution). The relation between w_2 and w_3 is the "sensible" balance between low energy cost on the one hand and low tardiness on the other. We typically set w_3 small enough (e.g., 10^{-3}) so as to attribute much more importance to energy cost, but still prefer earliest completions within a very small cost difference.

VI. SIMULATION AND RESULTS

We ran a number of simulation modeling the same scheduling problem both in the QPSOL 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.

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 QPSOL 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.

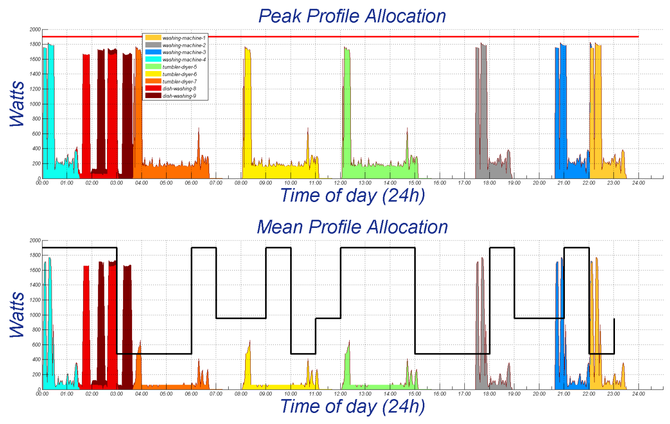

Figure 7. QPSOL simulation results: appliance scheduling with constant overload threshold, variable tariff, no photovotaic.

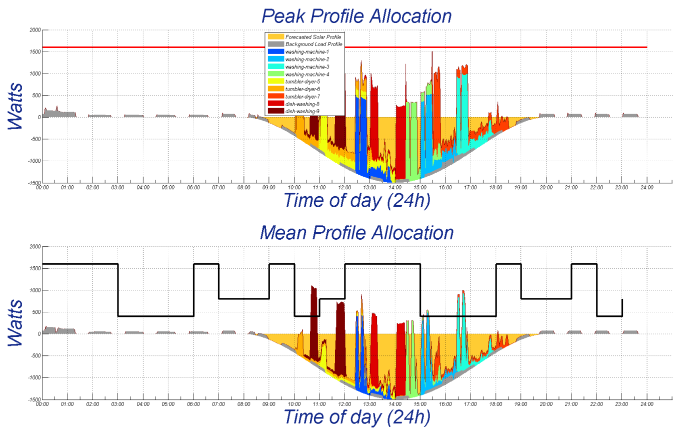

Figure 8. QPSOL simulation results: appliance scheduling with constant overload threshold, variable tariff, photovotaic.

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 QPSOL. Results show that although QPSOL 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 house 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.

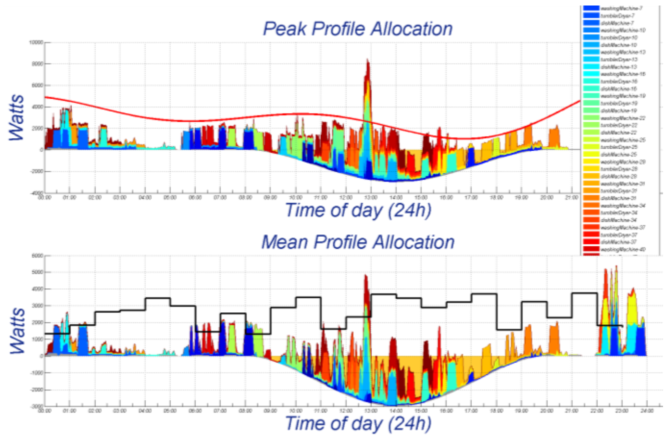

Figure 9. QPSOL simulation results: appliance scheduling for different apartments with variable overload threshold, variable tariff, photovotaic.

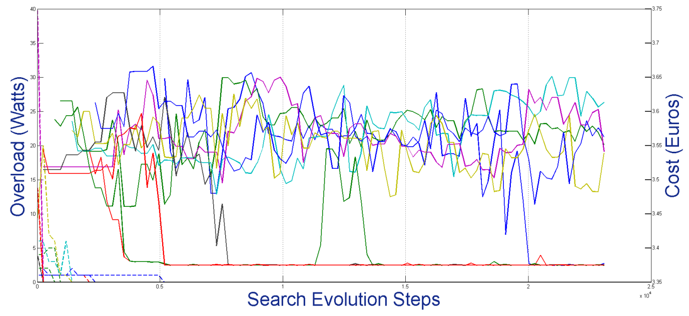

Figure 10. QPSOL simulation results: overload avoidance and optimization of cost

The 3 case studies described here show the remarkable flexibility of the QPSOL 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. In a rapidly changing world, algorithmic paradigms that are the most flexible to new conditions and can contribute to a time-based competitive advantage are more likely to be utilized. In such volatile environments, the utility of an algorithm framework will not be derived from the ability to solve a static problem. Instead it will be the ability to adapt to changing problem conditions that is likely to define the success or failure in the optimization algorithms of tomorrow.

Exact and formal techniques decompose the optimization problems into mathematically tractable domains 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. Mathematical techniques not only place constraints on the current problem definition but also on how that problem definition can 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 [20].

VII. Conclusion

This document describes an innovative Quantum inspired PSO with Lévy flights metaheuristic algorithm for scheduling home smart appliances, capturing all relevant appliance operations. With appropriately dynamic tariffs and short-term load forecasting, the proposed framework can propose a schedule for achieving high cost savings and overloads prevention. Good quality approximate solutions can be obtained in a short amount of computation time, in the order of about 2 seconds an almost optimal approximate solution can be obtained.

Finally, the proposed framework can be extended to incorporate solar power forecasting in the presence of a residential PV system by tuning 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] Energy@Home project, "Energy@Home Technical Specification version 0.95," December 22, 2011.

[2] 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.

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

[4] F. Glover, and M. Laguna, "Tabu Search", Kluwer Academic Publishers, Boston, 1997.

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

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

[7] 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.

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

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

[10] 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.

[11] D. Ionescu, A. Juan, J. Faulin, and A. Ferrer, "A Parameter-Free Approach For Solving Combinatorial Optimization Problems Through Biased Randomization Of Efficient Heuristics", in Proceedings of the Conference on Numerical Optimization and Applications in Engineering (NUMOPEN-2010), Barcelona, Spain. October 13-15, 2010.

[12] R. Kolisch, and S. Hartmann, "Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis", pp. 147–178, Kluwer, Amsterdam, the Netherlands, Kluwer academic publishers, 1999.

[13] 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.

[14] J. W. Taylor, "Short-Term Electricity Demand Forecasting Using Double Seasonal Exponential Smoothing" Saïd Business School University of Oxford - Journal of Operational Research Society, vol. 54, pp. 799-805, 2003.

[15] J. W. Taylor and P. E. McSharry, "Short-Term Load Forecasting Methods: An Evaluation Based on European Data", IEEE Transactions on Power Systems, vol. 22, pp. 2213-2219, 2008.

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

[17] Í. Goiri, K. Le, M. E. Haque, R. Beauchea, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini "GreenSlot: Scheduling Energy Consumption in Green Datacenters", SC'11, Seattle, Washington, USA, November 2011.

[18] 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.

[19] P. Bacher, H. Madsen, and H. A. Nielsen, "Online Short-term Solar Power Forecasting", Sol. Energy, vol. 83, pp. 1772–1783, 2009.

[20] 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.