Dispatch of Interruptible Loads Using Binary Particle Swarm Optimization: A Comparison of Constraint-Handling Methods

Michael Angelo Pedrasa[1,2], Ted Spooner[1], and Iain MacGill[1]
[1]University of New South Wales, Sydney, Australia
[2]University of the Philippines, Quezon City, Philippines

Abstract - Interruptible loads are consumers who agree to be interrupted, or whose consumption may be reduced by the utility, in order to maintain system security or to reduce market prices. Incorporating interruptible loads in day-ahead markets could maximize their potential in providing ancillary services because it enables them to plan their operations, and for them to offer reliable curtailment capacities and competitive bids. This paper investigates the use of binary particle swarm optimization to schedule a set of interruptible loads over a 16-hour period. The scheduling objective is to minimize the total payments to the interruptible loads and the frequency of interruptions, and to satisfy the required hourly curtailments and operational constraints imposed by the interruptible loads. The constraints in this multi-objective optimization problem were handled using four constraint-handling methods: using static and adaptive penalty functions, tracking only feasible solutions, and using repair algorithms. The suitability of these constraint-handling approaches to the problem at hand were investigated and compared.

I. Introduction

Interruptible load programs are a subset of load management programs implemented by utilities as an alternative to maintaining the security of the power system. These programs are designed to reduce the peak demand or shift the demand from peak periods to non-peak periods. In this context, interruptible loads (IL) are consumers who agree to be interrupted, as required and within constraints, to maintain system security or reduce market prices. They are compensated by paying reduced tariffs [1] or by incurring credits in their electricity bills. These consumers may be industrial customers with their own backup generation or those that can easily re-schedule production. They can also be residential customers who want to save on their electricity bill, or retail electricity providers that aggregate the consumption of small customers. The utility can directly interrupt supply to the customer, or the customer can disconnect or reduce consumption at the direct request of the utility.

Several IL dispatch algorithms have been reported in the literature. Some of the approaches are mixed-integer linear programming [2], iterative dynamic programming [3] and fuzzy dynamic programming [4]. On the other hand, dispatch methods used in actual power systems are much simpler. The utility requests participating loads to reduce consumption whenever the system reserves have breached a pre-defined critical margin or when the market prices have exceeded a certain value.

We investigated the use of the binary version of particle swarm optimization (BPSO) to create a day-ahead dispatch schedule for interruptible loads. Given a required curtailment schedule over a 16-hour time period, BPSO will create a dispatch schedule from a list of available loads to satisfy the required curtailments. The problem at hand is a constrained nonlinear optimization problem. We implemented four different constraint-handling approaches to determine which works best in generating a feasible dispatch schedule.

II. Particle Swarm Optimization

Kennedy and Eberhart introduced particle swarm optimization in 1995 [5]. The algorithm works by mimicking the behavior of a flock of birds that is searching for food. The particles in the swarm have simple behaviors. However, the collective behavior of the swarm achieved through interaction of the particles enables it to find the solution to a difficult optimization problem.

A particle represents a candidate solution to the problem, and the search for the solution is realized by having the population of particles fly throughout the solution space. The trajectory of a particle is affected by the best performing particle, or global best, and by the best position it has achieved, or personal best. The global and personal bests are selected by evaluating an objective (or cost) function.

The following outline describes the algorithm:

1. *Initialization.* All particles are randomly initialized. All initial particles are also designated as personal bests, and the global best is chosen among them.
2. *Particle movement.* The speeds $V(v_1, v_2, \ldots, v_n)$ and positions $P(x_1, x_2, \ldots, x_n)$ of all particles are updated using (1) and (2). The three terms in (1) are the momentum of the particle, the pull of the personal best position, and the pull of the global best particle.

$$v_i^{t+1} = \omega \cdot v_i^t + c_1 \cdot rand() \cdot (x_{pbest,i}^t - x_i^t) + c_2 \cdot rand() \cdot (x_{gbest,i}^t - x_i^t) \quad (1)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1}, \quad (2)$$

rand() is a uniform number generator from 0 to 1.

3. *Fitness evaluation.* The fitness of all current particles are evaluated and compared to their personal bests. The personal bests and the global bests are then updated if needed.
4. *Iterate.* Steps 2 and 3 are repeated until the maximum number of iterations is reached or when a convergence criterion is satisfied. The global best at the end of the simulation is taken as the solution to the problem.

In the binary version of particle swarm optimization [6], a coordinate x_i is either 0 or 1. The speed of that coordinate, v_i, is also computed using (1), and is restrained to lie within the range $[-V_{max}, V_{max}]$.

The speed, v_i, is used to determine whether x_i is 0 or 1. It is mapped to a real value between 0 and 1 using a sigmoid function (3) then compared to a random number between 0 and 1. If $rand() < S(v_i)$, then $x_i = 1$, otherwise, $x_i = 0$.

$$S(v_i) = \frac{1}{1 + \exp(-v_i)}. \quad (3)$$

III. Methodology and Implementation

A. Description of the Problem

We used BPSO to solve an IL scheduling problem that should meet a sequence of required hourly curtailments over a 16-hour period using a group of participating ILs. The required hourly curtailments are shown in Fig. 1.

The schedule should also satisfy the operational constraints imposed by the ILs. The ILs have different capacities and curtailment incentive rates. Each also requires a maximum curtailment period (Max OFF) and a minimum duration between curtailments (Min ON). The IL characteristics are summarized in Table I. These ILs were adapted from an IL scheduling problem discussed in [4].

The other objectives of the optimization problem are to minimize the total payment to the ILs and the total number of interruptions. The total payment is the sum of the products of the curtailed capacities and the corresponding curtailment rates.

The minimization of frequency of interruptions is a feature added to the original scheduling objectives tackled in [4]. This feature makes the dispatch schedule socially acceptable, especially when the required curtailment is large. It distributes the interruptions over all ILs and avoids interrupting a particular IL frequently. To minimize the frequency of interruptions, a penalty is incurred whenever a dispatched IL is dispatched again.

B. Implementation of BPSO for IL Dispatch

Each particle of the swarm is a 19 × 16 matrix, *Sch*, corresponding to the number of ILs and time intervals. The element at the i^{th} row and j^{th} column, *Sch*(*i*,*j*), is 1 if the i^{th} IL is interrupted during the j^{th} hour, otherwise, *Sch*(*i*,*j*) = 0. The speed of *Sch*(*i*,*j*), *Vel*(*i*,*j*), is computed by

$$\begin{aligned} Vel(i,j)^{t+1} = {} & Vel(i,j)^t \\ & + \phi \cdot rand() \cdot \left(Sch^t_{Pbest}(i,j) - Sch(i,j)^t \right) \\ & + \phi \cdot rand() \cdot \left(Sch^t_{Gbest}(i,j) - Sch(i,j)^t \right). \end{aligned} \quad (4)$$

The values of *Sch*(*i*,*j*) are computed by mapping *Vel*(*i*,*j*) to a probability using the sigmoid function (3) and comparing the result to the output of random number generator.

The total payments to the ILs, f_1, is computed by

$$f_1(Sch) = \sum_{i=1}^{19} \sum_{j=1}^{16} Sch(i,j) \cdot P_i \cdot \lambda_i, \quad (5)$$

where P_i and λ_i are the capacity and curtailment rate of the i^{th} IL.

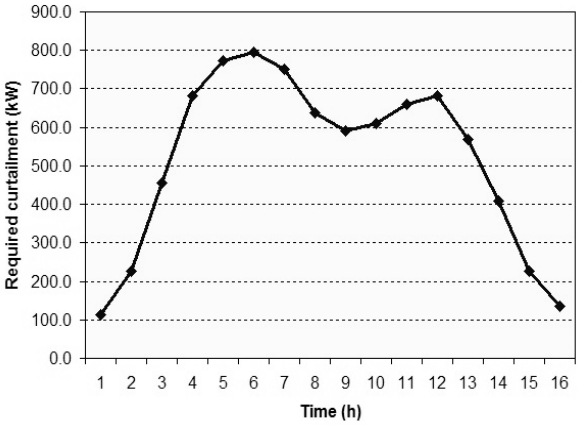

Figure 1. Required hourly curtailments.

TABLE I
INTERRUPTIBLE LOAD CHARACTERISTICS

IL no.	Capacity *P* (kW)	Max OFF (hours)	Min ON (hours)	Curtailment Rate, λ ($/kWh)
1	320	4	2	23.8
2	200	4	2	25.7
3	80	4	2	16.9
4	84	4	2	16.9
5	100	4	2	19.2
6	160	4	2	23.6
7	100	3	1	19.2
8	60	3	1	16.9
9	200	3	1	25.7
10	40	4	1	16.9
11	40	3	1	16.9
12	72	3	3	16.9
13	140	4	3	21.4
14	80	3	3	16.9
15	40	3	3	25.7
16	180	3	3	25.7
17	180	4	2	25.7
18	160	4	2	23.6
19	60	4	2	16.9

The penalty for interruptions, f_2, is computed as follows:

- No penalty is incurred on the first interruption of any IL.
- The penalty for the second interruption of any IL is 1000.
- To minimize the frequency of interruptions, the penalty doubles after each interruption of a particular IL. That is, the penalty for the third interruption is 2000, and for the fourth interruption, 4000.

Mathematically,

$$f_2(Sch) = \begin{pmatrix} C_{\text{int}} \cdot 2^0 \cdot FI_2 + C_{\text{int}} \cdot 2^0 \cdot FI_2 + \\ \ldots + C_{\text{int}} \cdot 2^{\beta-2} \cdot FI_\beta \end{pmatrix}, \quad (6)$$

where

C_{int} penalty weight for interruptions = 1000
FI_k number of ILs interrupted at least k times
β max. no. of interruptions incurred by any IL.

The objective of the optimization problem is to achieve the required hourly curtailments, while minimizing the payments to the ILs and number of interruptions, and satisfying the IL operational constraints. We re-stated the problem by assigning the payments (f_1) and the interruption frequency (f_2) as quantities to be minimized, and by treating the required hourly curtailments and IL requirements as constraints. The problem is further simplified by combining the two objective functions, f_1 and f_2, to form a single cost function.

The optimization problem is mathematically formulated as

Minimize $f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})$ (7)
Subject to $g_1(\boldsymbol{x}) = 0$ and $g_2(\boldsymbol{x}) = 0$, (8)

where

$\boldsymbol{x}$ dispatch schedule
$f_1(\boldsymbol{x})$ total payments to the ILs
$f_2(\boldsymbol{x})$ total penalty for IL interruption
$g_1(\boldsymbol{x})$ number of violations of required hourly curtailments
$g_2(\boldsymbol{x})$ number of violations of IL operational constraints

We used 250 particles and 250 iterations in the simulations. The particles are initialized in a way that the number of 1's has a binomial distribution with probability of 50%.

C. Handling of Constraints

Several methods are available in handling constraints in optimization problems with heuristic algorithms as solvers. The four techniques we investigated are a) the use of static and b) adaptive penalty functions, c) the tracking of feasible solutions, and d) the use of repair algorithms.

1. *Static Penalty Functions (SPF)* [7]

In this approach, a penalty is incurred whenever a constraint is violated, and the penalties are added to the objective function to form an aggregate cost function. The problem described by (7) and (8) transforms into an unconstrained optimization problem that minimizes the aggregate cost function $\Phi(\boldsymbol{x})$, where

$$\Phi(\boldsymbol{x}) = f_1(\boldsymbol{x}) + f_2(\boldsymbol{x}) + [k_1 \times g_1(\boldsymbol{x})] + [k_2 \times g_2(\boldsymbol{x})]. \quad (9)$$

For the problem at hand, the satisfaction of required hourly curtailments is more important than satisfying the IL operational constraints. Therefore, we assigned a larger penalty for each time interval that the required curtailment is not reached than when an IL constraint is violated.

2. *Adaptive Penalty Functions (APF)* [7]

This is similar to the previous approach except that the constraint penalty weights changes adaptively throughout the evolution. The cost function is revised to

$$\Phi(\boldsymbol{x}) = f_1(\boldsymbol{x}) + f_2(\boldsymbol{x}) + \lambda(t)\{[k_1 \times g_1(\boldsymbol{x})] + [k_2 \times g_2(\boldsymbol{x})]\}, \quad (10)$$

where

$\lambda(t+1) = \dfrac{\lambda(t)}{\beta_1}$ if the global bests during the last n iterations are feasible (i.e. does not violate any constraint)
$= \beta_2 \times \lambda(t)$ if the global bests in at least one of the last n iterations is not feasible
$= \lambda(t)$ otherwise, (11)

and $\beta_1 > \beta_2 > 1$.

In APF, the penalties due to the constraints are relaxed whenever the global bests during the last n iterations are in the feasible region (i.e. satisfy all constraints), and increased when the global bests are in the infeasible region (i.e. violates at least one constraint). The method gives more weight to the constraints (g_1 and g_2) at the start of the simulation, while the candidate solutions violate them. Emphasis then shifts to the minimization of the cost function ($f_1 + f_2$) when feasible candidate solutions have been generated.

3. *Feasible Solutions Method (FSM)* [8]

In FSM, the particles are initialized such that they are in the feasible region. During the evolution, the particles can enter the infeasible region, however, only those remaining and returning to the feasible region are considered when choosing the global and personal bests. In this approach, the iterative process is expedited because only the feasible particles are tracked [9].

4. *Repair Algorithms (RA)* [7]

All particles are also initialized such that they are in the feasible region. In an iteration, all particles that entered the infeasible region are repaired so they are returned to the feasible region.

The schedules are repaired by turning off dispatched ILs during the time periods when their operational constraints are violated. Afterwards, randomly selected available ILs are dispatched during the time periods when the required curtailment is not satisfied. An IL is available during a time

period if its operational constraints will not be violated if it will be dispatched during that period.

The particles that cannot be repaired using this procedure are left in the infeasible region.

D. Choosing the Simulation Parameters

Sensitivity analysis was used to choose the values of V_{max} and ϕ in (4). The values were chosen by running 100 simulations of BPSO using static penalty functions using each combination of V_{max} = {3, 4, 5, 6, 7} and ϕ/V_{max} = {1.5, 2.0, 2.5, 3.0}. Only the combination of V_{max} = 5.0 and ϕ = 7.5 yielded 100 solutions without constraint violations, so these values were chosen for all simulations.

The penalty values for not meeting the required hourly curtailments and for violating an IL constraint were chosen experimentally. Small values were initially used, and they were increased until the resulting schedules do not violate the constraints. The chosen values are k_1 = 10000000 and k_2 = 1000000. The ratio between k_1 and k_2 implies that at least 10 IL constraints are violated before a time interval would have insufficient curtailments.

For the APF method, the values used during the simulations are β_1 = 1.03, β_2 = 1.02 and n = 5. It was observed that the number of feasible solutions decreases when the values of β_1 and β_2 are increased. When β_1 = 1.04 and β_2 = 1.03, only 8 of 10 simulations return feasible solutions. Smaller values for β_1 and β_2 were not investigated because pushing β_1 and β_2 closer to 1.00 would reduce APF to SPF.

IV. Simulation Results and Analysis

BPSO using the four constraint-handling methods were executed 100 times. The best schedules are summarized in Table II, and the average performance of the methods are summarized in Table III.

TABLE II
BEST SOLUTION PRODUCED BY EACH CONSTRAINT-HANDLING METHOD

	Fitness	Total payment to ILs	No. of interruptions	No. of constraint violations
SPF	203053.2	188053.2	33	0
APF	202316.0	186316.0	33	0
FSM	227642.8	192642.8	44	0
RA	236144.0	202144.0	44	0

TABLE III
AVERAGE PERFORMANCE OF THE FOUR CONSTRAINT-HANDLING METHODS

	Average fitness	Number of feasible solutions (out of 100)	Average simulation time (s)
SPF	212070.05	100	44.4
APF	213046.11*	97	44.5
FSM	246905.24	100	42.1
RA	251689.02	100	231.7

*Does not include the 3 infeasible schedules.

The results show that BPSO with all constraint-handling methods were able to generate curtailment schedules using varied interruptible loads. BPSO-APF was able to generate the best schedule, while the best schedule generated by BPSO-SPF is very close behind. In fact, the fitness of the two best schedules are within 0.4% of each other. Table III, however, shows that 3 of the 100 schedules generated by BPSO-APF have constraint violations while all BPSO-SPF schedules are feasible. Furthermore, the average fitness of the feasible schedules generated by BPSO-SPF is better than that of BPSO-APF, and the former is also marginally faster. BPSO-FSM is fastest, however, the schedules it generated are not as good as those generated when using penalty functions.

Fig. 2 shows the hourly curtailments produced by the BPSO-SPF and BPSO-APF schedules. The curtailed demands closely follow but slightly exceed the required curtailment. Fig. 3 shows the improvement of the fitness of the global best particle throughout the simulation for BPSO-SPF and BPSO-APF. It shows that BPSO-SPF converges faster than BPSO-APF.

The results in Tables II and III also show that BPSO-FSM and BPSO-RA were able to generate feasible solutions. However, the total curtailed load exceeds the required curtailment by larger margins, as shown in Fig. 4. Based on these results, it may be concluded that FSM and RA are unattractive constraint-handling methods for this scheduling problem. The resulting payments to the ILs are larger and there are much more interruptions. It is also possible that some ILs are interrupted unnecessarily or longer than needed. Furthermore, RA is the slowest among the four methods. The repairing of infeasible particles consumes a significant amount of processing time.

The inferior performance of BPSO-FSM compared to BPSO-SPF and BPSO-APF may be attributed to the way the particles are initialized. By confining the particles within the feasible regions at the start of the simulations in BPSO-FSM, they were not able to explore the infeasible region as much as the particles guided by SPF and APF. The particles in BPSO-SPF and BPSO-APF are much more dispersed throughout the

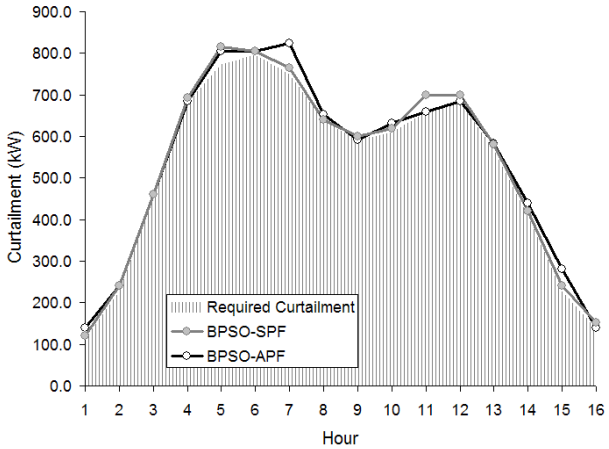

Figure 2. Curtailments produced by BPSO-SPF and BPSO-APF compared to the required curtailment

solution space at the start of the evolution. As they swarm towards the feasible region, they were able to gather valuable information from the infeasible region and carry that information when they reach the feasible regions. Some areas located inside the infeasible region may have good qualities, and these qualities are the information that the particles pick up when they pass through these areas. These information could help the particles achieve better solutions when they reach the feasible region.

It may be deduced that a significant portion of the solution space is infeasible because when the 250 particles in BPSO-SPF and BPSO-APF are initialized, all of them are infeasible. Some of these particles may even be initialized to be deep inside the infeasible region. These particles, therefore, have greater chances of encountering good quality regions located in the infeasible region.

The performance of BPSO-RA may be attributed to the way the constraints are handled. If the current global best particle is lying near the edge of the feasible region, some of the particles could overshoot the boundary when they follow this particle. When these particles are repaired, the repair process could lose important information that the particles have previously gathered. In BPSO-SPF and BPSO-APF, infeasible particles are not forcibly altered, instead, they are guided back to the feasible region, bringing with them important information from the infeasible region.

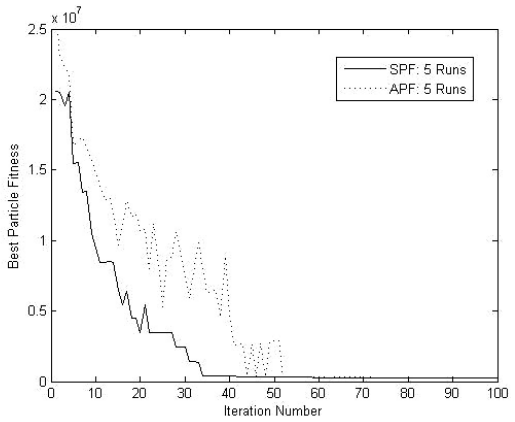

Figure 3. Fitness of the global best vs. iteration number for BPSO-SPF and BPSO-APF.

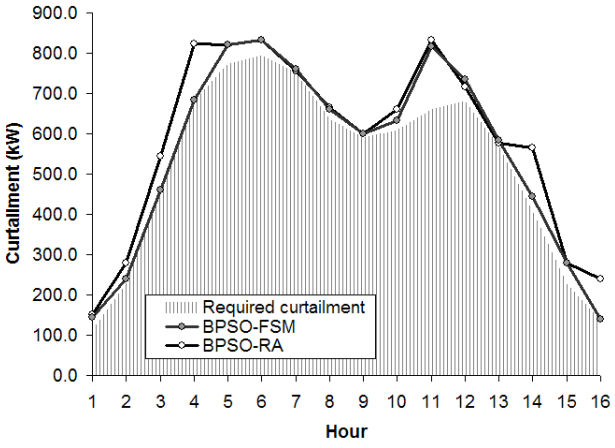

Figure 4. The demand curtailed by BPSO-FSM and BPSO-RA compared to the required curtailment

The enormous size of the solution space, the formulation of the cost function, and the demanding constraints underscore the complexity of the scheduling problem. Scheduling 19 interruptible loads over a 16-hour interval implies that there are $2^{(19 \times 16)}$ (roughly 3.26×10^{91}) distinct points in the solution space. Having 250 particles wandering around the solution space in 250 iterations means that only at most 62,500 of these points are visited. Furthermore, the intricacy of the cost function and the constraints imply that the quality of adjacent points could differ significantly. As an illustration, a change in one coordinate of a feasible solution may result in a violation of an IL constraint, or worse, an under-curtailment. Due to the complexity of the scheduling problem, it is unlikely for the 250 particles to find the absolute optimal schedule. In fact, finding a valid curtailment schedule that closely surpasses the required hourly curtailments is an achievement by itself. However, it may be inferred that the fitness of the best schedules produced by BPSO-SPF and BPSO-APF are close to the fitness of the absolute optimal schedule because (a) the excess curtailment is less than the smallest activated IL in all time periods, (b) all ILs are dispatched at least once, and (c) any IL is dispatched at most thrice. The last two reasons suggest that the number of interruptions is minimized and interruption is distributed across all loads.

In actual implementation, it might be impractical to run the simulation 100 times to generate one schedule especially if the scheduling decisions are needed immediately. To arrive at a good schedule, BPSO may be run 5 or 10 times and the best schedule is picked. Table IV shows the results of 5-run simulations for BPSO-SPF and BPSO-APF. The quality of the schedules are close to those achieved in the 100-run simulations.

V. Conclusion

The binary version of particle swarm optimization has been shown as an effective dispatch algorithm for interruptible loads. BPSO was able to generate acceptable solutions given the enormous complexity of the scheduling problem. We were able to show that for this problem, combining the cost function with the penalties due to constraint violations to form a single

TABLE IV
COMPARISON BETWEEN BPSO-SPF AND BPSO-APF 5-RUN SIMULATIONS

		BPSO-SPF	BPSO-APF
No. of feasible schedules		5	5
Average Fitness		209828.88	212550.96
Best Schedule	Fitness	204916.80	207391.60
	IL Payments	188916.80	191391.60
	Interruptions	34	34

aggregate cost function is an effective constraint-handling approach. The results when using static penalty functions and adaptive penalty functions are comparable to each other. The best schedule is produced when using adaptive penalty functions. However, the average schedule generated when using static penalties is better than the average schedule generated when using adaptive penalties. BPSO with static penalties also converges slightly earlier than BPSO with adaptive penalties. Finally, the results have shown that the tracking of feasible solutions and the repair algorithm methods cannot generate good quality solutions.

REFERENCES

[1] C. D. Vournas, "Interruptible load as a competitor to local generation for preserving voltage security," in Proc. 2001 IEEE Power Engineering Society Winter Meeting, vol. 1, pp. 236–240.

[2] L. M. Xia, H. B. Gooi, and J. Bai, "Probabilistic spinning reserves with interruptible loads," in Proc. 2004 IEEE Power Engineering Society General Meeting, vol. 1, pp. 146–152.

[3] K. Y. Huang, "Demand subscription services – an iterative dynamic programming for the substation suffering from capacity shortage," IEEE Trans. Power Systems, vol. 18, no. 2, pp. 947–953, May 2003.

[4] K.-Y. Huang, H.-C. Chin, and Y.-C. Huang, "A model reference adaptive control strategy for interruptible load management," IEEE Trans. Power Systems, vol. 19, no. 1, pp. 683–689, Feb. 2004.

[5] J. Kennedy and R. Eberhart, "Particle swarm optimization," in Proc. 1995 IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948.

[6] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in Proc. 1997 IEEE International Conference on Systems, Man and Cybernetics, vol. 5, pp. 4104–4108.

[7] C.A. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art," Computer Methods in Applied Mechanics and Engineering, Volume 191, Issues 11-12, 4 January 2002, Pages 1245-1287

[8] X. Hue and R. Eberhart, "Solving constrained nonlinear optimization problems with particle swarm optimization," in Proceedings of the Sixth World Multiconference on Systemics, Cybernetics and Informatics 2002.

[9] Coath, G., Halgamuge, S.K., "A comparison of constraint-handling methods for the application of particle swarm optimization to constrained nonlinear optimization problems," in Proc. 2003 Congress on Evolutionary Computation, vol. 4, pp. 2419 – 2425.