A Simple Yet Effective Approach to Robust Optimization Over Time

Abstract

Abstract--Robust Optimization over time (ROOT) refers to considering the uncertainties and changes that occur over time in an optimization problem. It is an optimization method aimed at finding stable solutions in dynamic environments. Traditional optimization methods typically assume fixed parameters, deterministic objective functions, and static constraints. However, in reality, many problems involve various uncertainties and dynamic factors, such as changing data, environmental variations, and evolving constraints. These factors can invalidate the assumptions of the original optimization model, rendering the optimization results ineffective or unreliable. Therefore, it is necessary for the algorithm to quickly converge to a global optimum after each environmental change, without being trapped by previous suboptimal solutions. However, this is often unrealistic in many real-world optimization problems. Hence, it is worth considering finding robust solutions that are relatively insensitive to changes over time, rather than pursuing constantly changing optimal values. When the quality of a solution remains acceptable within a certain time interval and is relatively insensitive to environmental changes during this interval, it is referred to as a robust solution.

Most of the existing algorithms use particle swarm optimization combined with another method which predicts future solutions to the optimization problem. We argue that this approach may perform subpar and suggest instead a method based on a random sampling of the search space. We prove its theoretical guarantees and show that it significantly outperforms the state-of-the-art methods for ROOT.

Keywords: Dynamic optimization; Robust optimization; Robust optimization over time; Uniform sampling; Particle swarm optimization

1 Introduction

The classical optimization problem involves minimizing or maximizing a function f over a region X. Typically, these problems depend on time t and random variables (also called environment variables)$\alpha(t)$. These questions can be written as:

$$\underset{x \in X}{\text{maximize}}\, f(\boldsymbol{x}; \boldsymbol{\alpha}(t)). \qquad (1)$$

Consider the scenario where only the history of $\alpha(t)$ up to time t is known, and there is no information available about its future distribution. Furthermore, in the absence of exact knowledge of $\alpha(t)$, the objective can only be accessed through black-box evaluations. My goal is to find the optimal solution for (1). Due to limited computational budget, it is necessary to leverage previous function evaluations to guide the search for the current moment. These settings describe "solution tracking," where the solution can be recomputed and changed at each time step. However, this is often impractical or even impossible as re-implementing the solution set may be physically unfeasible or result in additional costs or inconvenience for the user.

Thus, another approach has been proposed that focuses not on the current time performance of the solution, but rather on its performance over a future period of time. Therefore, the solution does not necessarily have to perform exceptionally well at the current moment but must deliver satisfactory results as time progresses. This problem is known as Robust Optimization Over Time (ROOT).

A good ROOT solution should display good performance in at least one of two main performance criteria: the first being the average performance over the future time interval, and the second being a time where the solution performs better than a given threshold.

In this paper, we first propose a new method. While the most advanced methods currently employ improvements using Particle Swarm Optimization, this paper uniformly samples the search space and then improves the best point through local search. The advantage of uniform search is that it provides a theoretical boundary on the quality of the solution. Additionally, if the problem dimensionality is low, the sampled points may be the same for all times, allowing for the use of prediction algorithms without the need to re-evaluate the function for earlier times.

2 A SIMPLE APPROACH TO ROOT

There are many alternatives to solve (1). Stochastic optimization maximizes *f* in expectation, while robust optimization maximizes it in the worst case. Dynamic optimization models evolution via ordinary differential equations, while multistage programming generalizes stochastic optimization by considering a longer horizon. All of these fields assume knowledge of the distribution of *α(t)*, and they are computationally quite expensive.

2.1 Problem formulation

We are considering a time-discrete ROOT problem and seeking to solve equation (1) for all t ∈ {1,.....,T}. We are considering a rather general case where, at time *t*, we can compute the target value f(x, α(t)) for any query point *x*. We do not know the exact value of *α(t)* or its future distribution, but we can utilize all previous function values at time instances (1, ..., t-1).

To assess the quality of the solution x(t) at time t, we consider two metrics:

$$F_{\text{aver}}(\boldsymbol{x}(t);t) = \frac{1}{S}\sum_{s=0}^{S-1} f(\boldsymbol{x}(t);\boldsymbol{\alpha}(t+s)), \qquad (2)$$

$$F_{\text{surv}}(\boldsymbol{x}(t);t) = \min\{s \geq 0 \mid f(\boldsymbol{x}(t);\boldsymbol{\alpha}(t+s)) \leq f^*\}.$$

The average objective metric *Faver* measures the average performance over a future time interval, while the survival metric *Fsurv* measures the time that the objective stays above a threshold *f**. Both metrics use a target function that evaluates both the current time, which can be assessed, and the future time, which can only be predicted.

The future value α(t+s) in (2) is considered fixed but unknown. In the field of stochastic optimization, this corresponds to adding the expectation with respect to α to (2). Since in the numerical part we average the results over different realizations of α, we should technically also add this expectation to (2). The key difference is that stochastic optimization assumes that the future distribution is known, while we assume that it is unknown.

2.2 Proposed methods

Most existing ROOT methods are based on particle swarm optimization. These papers do not provide any convergence proof and require tuning of hyperparameters. In this section, we propose two very simple methods that are not affected by these issues. The first method solves equation (1) at the current time *t* without considering the past or future, while the second method aims to obtain a stable solution. At each time instance *t*, we have a computational budget for estimating the function f(·; α(t)) denoted as Nevale.

The first method spends N evaluations on global search and Nloc = Nevale - N evaluations on local search. The global search is performed by uniformly discretizing the search space into {x1, ..., xN} and evaluating fn(t) for all n=1....N. Then, we find the index nmax for which fn(t) has the maximum value and improve Xnmax using any local search method in Nloc function evaluations. We provide a summary of this in Algorithm II.1.

Algorithm II.1: Hybrid Method of Uniform Sampling and Local Search for Solving ROOT

The total number of function evaluations is Neval, with Nloc function evaluations dedicated to local search and N = Neval - Nloc evaluations for global search. The search space is uniformly discretized into {x1, ..., xN} for global search.

For all n = 1...N, fn(t) = f(Xn; α(t)) is evaluated at each time step, and we find the index nmax with the maximum value of fn(t). xnmax is then improved by Nloc local searches, resulting in the optimal solution xopt(t).

The second method dedicates all Neval evaluations to global search. Once again, the search space is uniformly discretized into {x1, ..., xNeval}, and robust solutions are selected by considering neighborhood or previous time-step function values. As the space discretization remains the same each time, except for fn(t), we also know fn(t-1) and f1...fn(1). A summary is provided in Algorithm II.2.

Algorithm II.2: Uniform Sampling Method for Solving ROOT

All Neval evaluations are used for global search. Similarly, the search space is uniformly discretized into {x1, ..., xNeval}.

For all n = 1, ..., Neval, compute fn(t) = f(xn; α(t)) at each time step. The method for selecting robust solutions considers neighborhood or previous time-step function values.

If the search space is given by $X = [x_{\min}, x_{\max}]^D$, the first method provides a solution that is optimal within the following bounds:

$$f(\boldsymbol{x}_{\text{opt}}(t);\boldsymbol{\alpha}(t)) \geq f^*(t) - \frac{L\sqrt{D}(x_{\max}-x_{\min})}{2(N^{\frac{1}{D}}-1)}, \quad (3)$$

Where f* (t) is the optimal solution at time *t* and L is the Lipschitz constant of f(x) on D. Since most ROOT methods are tested for the two-dimensional case D=2, the previous bound is quite tight. The quality of the solution is further improved by local search.

To summarize the benefits of the proposed two methods: (1) The advantage of uniform search is that it gives a theoretical bound on the quality of the solution. (2) Since the same point is evaluated at all moments, no additional function evaluation is required to use any tracking or prediction mechanisms from other ROOT papers.

3 NUMERICAL BENCHMARKS

Non-dominated sorting genetic algorithm was proposed by Srinivas and Deb in 1994. It has been criticized because of some problems mentioned earlier. In this section, the NSGA-II algorithm is proposed, which alleviates these difficulties. The NSGA-II algorithm will be presented next in several sections.

3.1 Moving peaks benchmark 1

The benchmark considers M peaks of a cone shape in RD. The peak M has a center, height, and width. The random vector is defined, and the objective function measures the height of the largest peak at *x*:

$$f_t^1(\boldsymbol{x};\boldsymbol{\alpha}(t)) = \max_{m=1,\ldots,M}(h_t^m - w_t^m\|\boldsymbol{x}-\boldsymbol{c}_t^m\|_{l_2}),$$

The dynamics of the random vector is given by

$$\begin{aligned} h_{t+1}^m &= h_t^m + \sigma_h^m \cdot N(0,1), \\ w_{t+1}^m &= w_t^m + \sigma_w^m \cdot N(0,1), \\ \boldsymbol{c}_{t+1}^m &= \boldsymbol{c}_t^m + \boldsymbol{v}_{t+1}^m, \\ \boldsymbol{v}_{t+1}^m &= s^m \frac{(1-\lambda)\boldsymbol{r}_{t+1}^m + \lambda \boldsymbol{v}_t^m}{\|(1-\lambda)\boldsymbol{r}_{t+1}^m + \lambda \boldsymbol{v}_t^m\|}. \end{aligned} \quad (4)$$

Here N (0,1) represents a normal distribution with zero mean and unit variance, $\boldsymbol{r}_t^m$ follows a uniform distribution on a d-dimensional sphere with radius s^m ,and λ∈[0, 1] are fixed parameters. The difference between the peak height h_{t+1}^m and the previous height h_t^m is a random number drawn from a normal distribution with zero mean and standard deviation σ_h^m.

The center $\boldsymbol{c}_{t+1}^m$ is moved from $\boldsymbol{c}_t^m$ by vector $\boldsymbol{v}_{t+1}^m$. If $\boldsymbol{v}_1^m$ has norm s^m, then we have:

$$\begin{aligned} \lambda = 0 &\implies \boldsymbol{v}_{t+1}^m = \boldsymbol{r}_{t+1}^m, \\ \lambda = 1 &\implies \boldsymbol{v}_{t+1}^m = \boldsymbol{v}_t^m. \end{aligned}$$

Thus, λ = 0 means that the motion at the center of the peak is random, while λ = 1 means that the motion is constant in the direction $\boldsymbol{v}_1^m$. In both cases, the distance between the previous center and the new center is s^m .

Random variables have certain bounds. The boundary of the center $\boldsymbol{c}_t^m \in [x_{\min}, x_{\max}]^D$ is the same as the boundary of the search space. If dynamic (4) pushes some variable out of its corresponding boundary, it is projected back.

Finally, for the initialization of (4), we need to know the initial center, height, width, and initial velocity. Following the previous paper, we randomly initialize the center in the search space [xmin, xmax]D with height and width to some known values hinit and winit, respectively, and the initial velocity is randomly generated on a d-dimensional sphere of radius s^m .

3.2 Moving peaks benchmark 2

Second benchmark question:

$$f_t^2(\boldsymbol{x};\boldsymbol{\alpha}(t)) = \frac{1}{D}\sum_{d=1}^{D}\max_{m=1,\ldots,M}\left(h_t^{m,d} - w_t^{m,d}|x^d - c_t^{m,d}|\right),$$

The index d represents the d-th component of a vector. The D-dimensional problem is then decomposed into D one-dimensional problems. Additionally, since the heights in each dimension can vary, the problem is no longer specifically dealing with moving peaks.

The dynamics of the parameters are similar to those in benchmark 1:

$$\begin{aligned} h_{t+1}^{m,d} &= h_t^{m,d} + \sigma_h^m \cdot N(0,1), \\ w_{t+1}^{m,d} &= w_t^{m,d} + \sigma_w^m \cdot N(0,1), \\ \boldsymbol{c}_{t+1}^m &= R(\theta_t^{D-1},\ldots,\theta_t^1)\boldsymbol{c}_t^m, \\ \theta_{t+1}^d &= \theta_t^d + \sigma_\theta \cdot N(0,1). \end{aligned} \quad (5)$$

The dynamics of height and width are the same as in the first benchmark (4). The center is rotated according to the matrix $R(\theta_t^{D-1}, \ldots, \theta_t^1)$. Where each rotation matrix $R^d(\theta_t^d)$ performs a rotation in the d-(d + 1) plane by angle θ_t^d. Similar to the first benchmark, if the variables are out of bounds, we project them back. We randomly initialize the centers in the search space [xmin, xmax]D. Based on the initial height and width of [2] and randomly generated from their boundaries. The initial θd1 is set to θinit.

4 EXPERIMENTAL RESULTS

All the results shown are the average of 5000 independent simulations of α.

4.1 Parameter setting

In Table I, we show the parameters used. We first generate a random evolution of α and then uniformly discretize the search space [xmin, xmax]D into Neval=2500 points. Algorithm II.1 randomly selects 2300 points from the 2500, evaluates f(·, α(t)) and selects the best value at each t, while the remaining 200 function evaluations are put into a local search using the built-in Matlab function fmincon. Algorithm II.2 evaluates all 2500 points, and replaces the function value at one point with the average of all neighboring values within a maximum distance of 3 (points outside the search space are ignored). The solution with the highest average value is considered a robust solution.

Table I
PARAMETER VALUES FOR BENCHMARK PROBLEMS

Parameter	Benchmark 1	Benchmark 2
N_{eval}	2500	2500
M	5	25
D	2	2
λ	{0,1}	-
$[x_{\min}, x_{\max}]$	$[0, 50]$	$[-25, -25]$
$[h_{\min}, h_{\max}]$	$[30, 70]$	$[30, 70]$
$[w_{\min}, w_{\max}]$	$[1, 12]$	$[1, 13]$
$[\theta_{\min}, \theta_{\max}]$	-	$[-\pi, \pi]$
σ_h	$U(1, 10)$	5
σ_w	$U(0.1, 1)$	0.5
σ_θ	-	1
h_{init}	50	$U(h_{\min}, h_{\max})$
w_{init}	6	$U(w_{\min}, w_{\max})$
θ_{init}	-	0

Although predicting future values could be achieved by using function evaluations at previous time points, we decided not to do so. The reason is that even this basic approach significantly outperforms the state of the art algorithms, and the addition of predictions can obscure the basic idea.

4.2 Numerical results

We compared three methods. Mesh and Time-optimal are based on Algorithm II.1, with the difference that Mesh does not perform a local search. Robust is based on Algorithm II.2.

We compare the Time-optimal method with the known results in Table III. On benchmark 1 and benchmark 2 with λ∈{0,1}, we show the average objective Faver for the time window S∈{2,6} and the survival function Fsurv for δ∈{40,50}. Both are defined in (2). We used the range T=100, and the results shown are averaged over all moments of T∈[20100]. Our results significantly outperform the best known results for all benchmarks and evaluation criteria.

Table III
COMPARISON OF BEST KNOWN AND OUR RESULTS. ALL METHODS USE 2500 FUNCTION EVALUATIONS AT EACH TIME INSTANT. THE PROCESS OF COLLECTING THE BEST KNOWN RESULTS IS DESCRIBED IN APPENDIX C. ALL EXPERIMENTS WERE REPEATED 5000 TIMES.

Setting	From	Best known result				Our result			
		F_{aver}		F_{surv}		F_{aver}		F_{surv}	
		$S=2$	$S=6$	$\delta=40$	$\delta=50$	$S=2$	$S=6$	$\delta=40$	$\delta=50$
Benchmark 1 with $\lambda=1$	[8]	53.48	8.82	3.02	1.69	63.32	58.76	13.72	10.11
Benchmark 1 with $\lambda=0$	[9] [11]	-	-	8.35	4.25	61.13	54.77	10.42	5.91
Benchmark 2	[2] [12]	48.88	40.58	1.35	1.02	62.21	57.58	16.54	6.38

It can be shown that our result is almost optimal. For example, base one with λ=0. The expected value of the optimal solution is about 65. Since the peak moves with a step size sm=1 with an average width of 6.5, the target drops to 65−6.5=58.5 at the next instant. So the expected target of 1/2 (65+58.5) =61.75 for S=2, which is very close to the value 61.13 in Table III.

We show the gap between the optimal objective and the discovery objective. Mesh shows about half of the theoretical gap, 3, and this gap is almost zero when we improve it by Time-optimal local search. This means that Time-optimal finds the center of the highest peak. We would like to emphasize that no information about the highest peak is used in the optimization process, we just use it as a posteriori value to evaluate the performance.

Table II
GAP BETWEEN THE BEST POSSIBLE OBJECTIVE f_t^* AND THE OBJECTIVE FOUND BY OUR METHODS

	Maximal gap (3)	*Mesh*	*Time-optimal*	*Robust*
Benchmark 1	4.69	2.18	0.09	5.38
Benchmark 2	5.05	0.99	0.15	4.38

Tables II and III also illustrate why the other methods perform below the norm: (1) Since the time-optimal solution is at the center of the peak, it is also a natural candidate for the robust solution. We believe that the commonly used PSO algorithm is far from the center of the peak. (2) While incorporating object tracking in previous papers, points at the previous moment need to be re-evaluated. This reduces the number of survey points. Note that, as explained at the end of Section II, our approach does not suffer from these problems.

We show additional results for Benchmark 1 and Benchmark 2 in Figures 3 and 4, respectively. For benchmark 1, the columns show the results for $\lambda = 0$(left) and $\lambda = 1$(right), while for benchmark 2, the columns show the random generation (left) and grid generation (right) of the initial centers. We can observe the following phenomena:

- In all cases, the Timeoptimal method with local search outperforms the Mesh method without local search.
- Robust outperforms Timeoptimal for one benchmark.
- The survival time of benchmark 2 is stable, while the survival time of benchmark 1 increases with time. The reason is that benchmark 1 initializes the peak heights to 50, while benchmark 2 initializes them randomly in [30,70]. Thus, for the former case, the maximum peak height at the initial time is much smaller.
- Initialization or parameters have a big impact on the solution (left vs. right column comparison).
- Benchmark 2 is not affected by variable boundary conditions. This does not hold for benchmark 1, where the survival time increases as the center reaches the boundary and stays there.

Overall, the Time-optimal method, which does not use any tracking or future prediction, performs very well in both benchmarks.

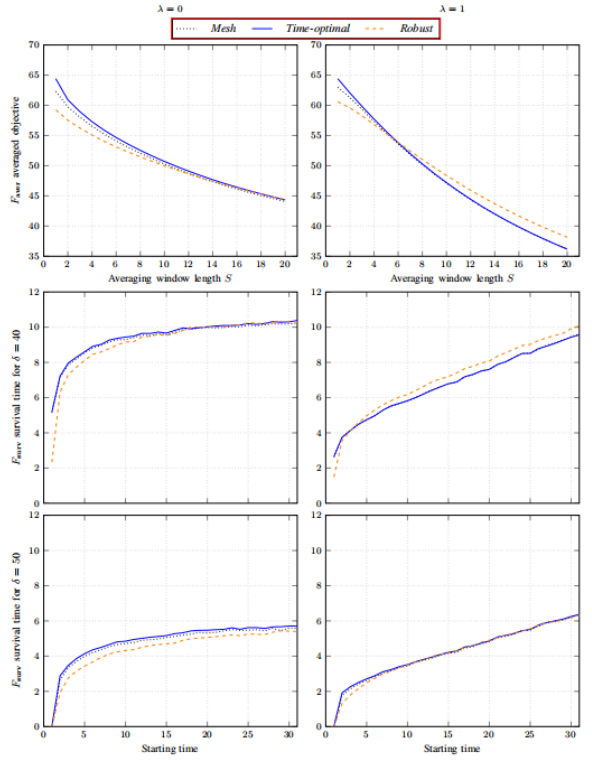

Figure 3. Results for Benchmark 1 with $\lambda = 0$ (left) and $\lambda = 1$ (right). We show the averaged objective F_{avve} as a function of the averaging time window S (top) and the survival function F_{surv} for thresholds $\delta = 40$ (middle) and $\delta = 50$ (bottom). Note that the metrics are defined in (2).

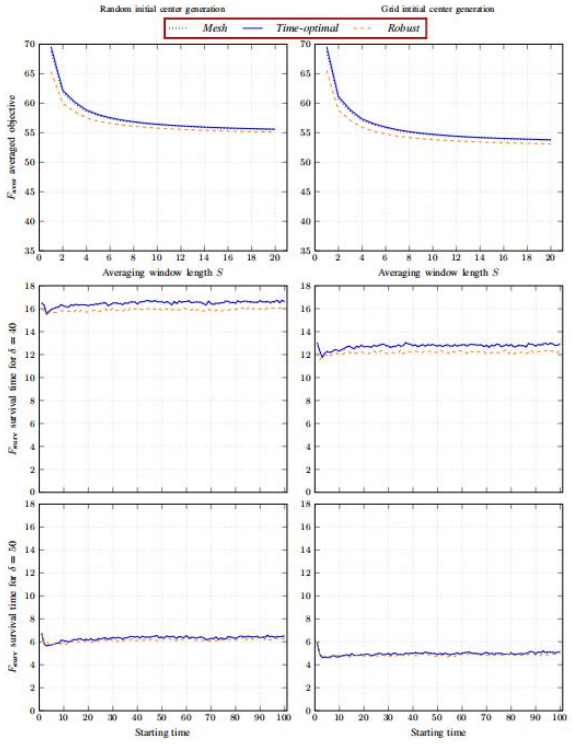

Figure 4. Results for Benchmark 2 with random center generation (left) and the grid center generation described in Appendix C (right). We show the averaged objective F_{avve} as a function of the averaging time window S (top) and the survival function F_{surv} for thresholds $\delta = 40$ (middle) and $\delta = 50$ (bottom). Note that the metrics are defined in (2).

5 Conclusion

In this paper, the mobility benchmark problem of ROOT is properly described and a simple solution is proposed. Most of the existing algorithms use PSO combined with another method to predict the future solution of the optimization problem. This paper argues that the performance of this approach may be substandard and suggests an approach based on random sampling of the search space. We prove its theoretical guarantees and show that it significantly outperforms the state-of-the-art ROOT method.

References

[1]X. Yu, Y. Jin, K. Tang, and X. Yao, "Robust optimization over time—a new perspective on dynamic optimization problems," in IEEE Congress on evolutionary computation. IEEE, 2010, pp. 1 – 6.
[2] H. Fu, B. Sendhoff, K. Tang, and X. Yao, "Robust optimization over time: Problem difficulties and benchmark problems," IEEE Transactions on Evolutionary Computation, vol. 19, no. 5, pp. 731 – 745, 2015.
[3] J. R. Birge and F. Louveaux, Introduction to stochastic programming. Springer Science & Business Media, 2011.
[4] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, Robust optimization. Princeton University Press, 2009, vol. 28.
[5] D. P. Bertsekas, Dynamic programming and optimal control. Athena scientific Belmont, MA, 1995, vol. 1, no. 2.
[6] M. V. Pereira and L. M. Pinto, "Multi-stage stochastic optimization applied to energy planning," Mathematical programming, vol. 52, no. 1-3, pp. 359 – 375, 1991.
[7] H. Fu, B. Sendhoff, K. Tang, and X. Yao, "Characterizing environmental changes in robust optimization over time," in 2012 IEEE Congress on Evolutionary Computation. IEEE, 2012, pp. 1 – 8.
[8] ——, "Finding robust solutions to dynamic optimization problems," inEuropean Conference on the Applications of Evolutionary Computation. Springer, 2013, pp. 616 – 625.
[9] Y. Jin, K. Tang, X. Yu, B. Sendhoff, and X. Yao, "A framework for finding robust optimal solutions over time," Memetic Computing, vol. 5, no. 1, pp. 3 – 18, 2013.
[10] Y. Guo, M. Chen, H. Fu, and Y. Liu, "Find robust solutions over time by two-layer multi-objective optimization method," in 2014 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2014, pp. 1528 – 1535.
[11] D. Yazdani, T. T. Nguyen, J. Branke, and J. Wang, "A new multiswarm particle swarm optimization for robust optimization over time," inEuropean Conference on the Applications of Evolutionary Computation. Springer, 2017, pp. 99 – 109.
[12] P. Novoa-Hernández, D. A. Pelta, and C. C. Corona, "Approximation models in robust optimization over time-an experimental study," in 2018 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2018, pp. 1 – 6.
[13] D. Yazdani, T. T. Nguyen, and J. Branke, "Robust optimization over time by learning problem space characteristics," IEEE Transactions on Evolutionary Computation, vol. 23, no. 1, pp. 143 – 155, 2018.
[14] M. Chen, Y. Guo, H. Liu, and C. Wang, "The evolutionary algorithm to find robust Pareto-optimal solutions over time," Mathematical Problems in Engineering, vol. 2015, 2015.
[15] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), vol. 3. IEEE, 1999, pp. 1875 – 1882.