Evolutionary Programming with Ensemble of Explicit Memories for Dynamic Optimization

E. L. Yu, and P. N. Suganthan, *Senior Member, IEEE*

***Abstract*—This paper presents the evolutionary programming with an ensemble of memories to deal with optimization problems in dynamic environments. The proposed algorithm modifies a recent version of evolutionary programming by introducing a simulated-annealing-like dynamic strategy parameter as well as applying local search towards the most improving directions. Diversity of the population is enhanced by an ensemble of external archives that serve as short-term and long-term memories. The archive members also act as the basic solutions when environmental changes occur. The algorithm is tested on a set of 6 multimodal problems with a total 49 change instances provided by CEC 2009 Competition on Evolutionary Computation in Dynamic and Uncertain Environments and the results are presented.**

I. Introduction

In recent years, evolutionary optimization in dynamic environments has attracted much interest among researchers. A successful dynamic optimization algorithm should not only be able to locate the optimum, as it does in the static sense, but also be capable of detecting when the environment changes and tracking the new optimum.

This can be a challenging problem, as in most cases, especially when the environment has been static for some time, the population tends to converge to the best solution, and thereby loses its adaptability. In order to be flexible enough to respond to environmental changes, the algorithm needs to explore several potentially good regions in the search space, and at the same time, maintain the pace of fine search for the optimal solution.

We have chosen evolutionary programming (EP) as the main algorithm to perform the task of optimization in dynamic environments on account of its fast convergence ability. Therefore our essential task is to increase the diversity of the population and ensure it does not get trapped into a single optimal solution so that it cannot evolve further. We use an ensemble of memory-based archives to introduce diverse individuals into the population. Moreover, we examine the strategy parameter in EP and propose a dynamic scheme that is designed to make the EP more suitable for optimization in dynamic environments.

We will follow with a literature review of evolutionary programming and introduce our dynamic strategy parameter and structure of the ensemble of external memories. Experimental results and a conclusion will be presented in the last sections.

Manuscript received November 10, 2008. This work was supported by the A*Star (Agency for Science, Technology and Research) under the grant #052 101 0020.

The authors are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798 (phone: 65-67905404; fax: 65-67933318; e-mails: yuling@ntu.edu.sg, epnsugan@ntu.edu.sg).

II. Evolutionary Programming

Evolutionary programming was first proposed in 1960s as an alternative method for generating artificial intelligence [1]. Later, since the middle of 1980s, it has been developed to solve more general tasks including prediction problems, numerical and combinatorial optimizations, and machine learning [2], [3]. As a member of evolutionary algorithms, the basic idea of EP is also the imitation of the natural evolution. Since this approach models organic evolution at the level of species, the original EP does not rely on any kind of recombination or crossover. The main difference between EP and other evolutionary algorithms such as genetic algorithms and differential evolution is that there is no information exchange among individuals in EP. Mutation is the only way to generate offspring.

Current studies involving evolutionary programming usually make use of self-adaptation of the strategy parameter in EP [4]–[6]. According to the description in [4], the self-adaptive EP is implemented as follows:

1) Generate the initial population of μ individuals, and set k=1. Each individual is taken to be a pair of real-valued vectors (x_i, η_i), i=1,2,…,μ, where x_i is the vector of objective variables and η_i is a vector of standard deviations (also known as strategy parameters) corresponding to Gaussian mutations.
2) Evaluate the fitness value for each individual (x_i, η_i) of the population based on the objective function $f(x_i)$.
3) Each parent (x_i, η_i), creates an offspring (x_i', η_i'), by:
$$x_i'(j) = x_i(j) + \eta_i(j) N_j(0,1) \quad (1)$$
$$\eta_i'(j) = \eta_i(j) \exp(\tau' N(0,1) + \tau N_j(0,1)) \quad (2)$$
for j=1,2,…,n, where n is the number of dimensions, $N(0,1)$ denotes a normally distributed one-dimensional random number with mean zero and standard deviation one. $N_j(0,1)$ indicates that the random number is resampled anew for each value of j. The factors τ and τ' are commonly set to $(\sqrt{2\sqrt{n}})^{-1}$ and $(\sqrt{2n})^{-1}$. The order of (1) and (2) may be swapped to improve the performance. Note that the offspring generated by such rules can be out of the variable domain of the problem. Hence, boundary checks should be performed.
4) Calculate the fitness of each offspring (x_i', η_i').

978-1-4244-2959-2/09/$25.00 ©

5) Conduct pair-wise comparison over the union of parents (x_i, η_i) and offspring (x_i', η_i'), $\forall i$=1,2,…,μ. For each individual, q opponents are chosen uniformly at random from the combined parents and offspring popluation. For each comparison, if the individual's fitness is no smaller than the opponent's, it receives a "win."
6) Select the μ individuals out of (x_i, η_i) and (x_i', η_i'), $\forall i$=1,2,…,μ, that have the most wins to be parents of the next generation.
7) Stop if the halting criterion is satisfied; otherwise, k=k+1 and go to Step 3.

In a recent study [7], a new version of EP called unbiased evolutionary programming (UEP) was introduced and was shown to have markedly improved performance for high-dimensional optimization. We will take this version to solve the dynamic optimization problems. Other than generating Gaussian deviates in axial directions, a set of angles is randomly generated in UEP. These are resolved into a direction vector in the co-ordinate system, which is then multiplied by a Gaussian deviate.

Several steps are added to the self-adaptive EP mentioned above. Firstly, an additional parameter ϕ_i is assigned to each of the population member. ϕ represents an (n−1)-dimensional vector of angles, initialized randomly from a uniform distribution between 0 and 2π. After that, create Δx_i, i=1,2,…,μ, as follows:

a) For each value of i, set *prevSin*=1. Then for each value of j, j=1,2,…,n−1, set:

$$\Delta x_i(j) = \cos(\phi_i(j)) \times prevSin \quad (3)$$

$$prevSin \leftarrow \sin(\phi_i(j)) \times prevSin \quad (4)$$

Finally set $\Delta x_i(n) = prevSin$.

b) Randomly permute the elements of Δx_i.

When generating the offspring (x_i', ϕ_i', η_i'), $\Delta x_i(j)$ is multiplied to the increment of $x_i(j)$, which makes (1) as:

$$x_i'(j) = x_i(j) + \eta_i(j) N_j(0,1) \Delta x_i(j) \quad (5)$$

$\phi_i'(j)$ is again randomly generalized from a uniform distribution between 0 and 2π, and $\eta_i'(j)$ is derived by (2). The other steps are as in the classical self-adaptive EP.

To make the UEP more efficient in searching for the optimum, we introduce a local search method to be applied to the algorithm. In every generation, the fitness of each offspring will be compared to that of its parent. From the largest increments of the fitness values we obtain the most-improving directions. These directions are used to perform the local search, which is described by the following equation:

$$x_i'' = x_i' + F(x_i' - x_i) \quad (6)$$

where $x_i' - x_i$ offers one of the most-improving direction and F is a constant (set to 0.85 in this paper).

III. The Proposed Dynamic Parameter Scheme

Observing the self-adaptive EP described in II, we find that the parameter η has a tendency to get near to either infinity or zero as generation number increases. The former situation is unfavorable, as solutions vibrate severely even at the end of the evolution. On the contrary, we may want to have a larger mutation deviation at the beginning, and, as population evolves, transfer the population from exploration to fine search near the good solutions with small mutation deviation. Annealing of the strategy parameter η will just serve the purpose.

Simulated annealing (SA) was proposed as a method for solving discrete optimization problems as well as single and multiple objective optimization problems in various fields [8]–[12]. Initial temperature is chosen such that it can capture the entire solution space. Cooling schedule determines functional form of the change in temperature required in SA. The earliest annealing schedules have been based on the analogy with physical annealing. They set initial temperature high enough to accept all transitions and used a proportional temperature. Three important cooling schedules are logarithmic, Cauchy and exponential. It was shown in [13] that the classical cooling schedules are all equivalent, with no clearly better annealing schedule than the logarithmic schedule to ensure convergence towards the set of optima with a probability of one.

We use a variant of the exponential form of cooling schedule to adjust the parameter in EP, which is explained by the following equation:

$$\eta = T_0 \exp\left(\sqrt{\frac{1}{n}} - \sqrt{\frac{t}{n}}\right) \cdot (r + 0.5) \quad (7)$$

where T_0 denotes the initial temperature which is set to 5 for our experiments, n is the dimension number, t stands for time or number of generations, and r is a uniformly distributed random variable in the interval of [0, 1]. This formula applies to the η value in every dimension of every individual in the population. A sample of the parameter values over 1000 generations in shown in Fig. 1.

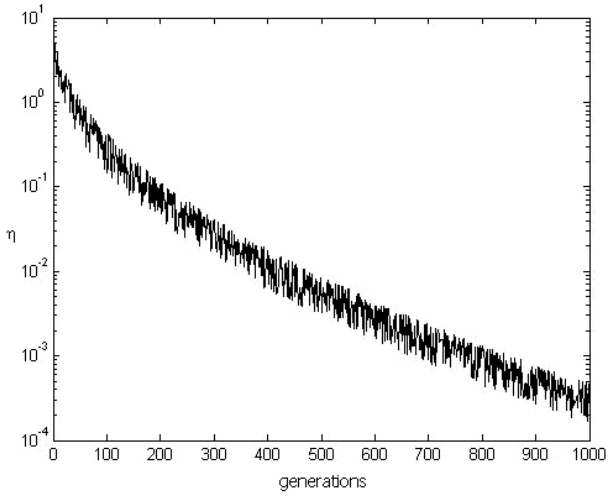

Fig. 1. Semi-log Plot of η.

The above scheme enables EP to have an exponentially decreasing deviation for mutations with certain perturbation. It is a simple and explicit form, which is very useful in

dynamic environments. The strategy parameter at any time depends only on the initial temperature, but not on the previous values. This makes it easy for the algorithm to adapt to environmental changes. On detecting a change, the strategy parameter will be re-initialized.

IV. Handling Environmental Changes

In order to make the algorithm be suitable for optimizing dynamic problems, we will make some further modifications. Although we can always regard each change as the arrival of a new problem, it is more efficient if we can make use of the previous knowledge about the search space, assuming that the changes are not too radical. There are many ways to deal with environmental changes [14], [15]. For example, after a change is detected, the mutation rate can be raised to increase the diversity of the population [16], or else, niching methods can be used to spread out the population so that it may adapt to changes more easily [17]. Memory-based approaches [18], [19] have also been proposed to recall information from past generations, which is especially useful when the new optimum is not far from previous locations. In addition, we may apply multi-population techniques to track multiple peaks in the fitness landscape [20], [21].

Here we propose a diversity enhancement method which is implemented not only after the environmental changes but also during the whole run. It is realized using an ensemble of two memory archives that store solutions found by the algorithm in different times before the current generation. The first archive, which represents the short-tem memory, serves for the regular updating, and the second, which represents the long-term memory, is used when premature convergence is identified. When both cannot help and the population totally loses diversity, we restart the search.

A. Archive Operations

The first archive is renewed every 10 generations, i.e. it stores all the offspring created in the last 10 generations. A niching method, clearing [22], is applied to the archive when it has gathered all the members. This is to ensure the individuals with high fitness in the archive are distributed evenly in the search space instead of crowding in one area. After the clearing procedure, we may assume the archive to have a certain level of diversity and we can directly make use of the members to replace the population in order to allow further exploration.

Different from the standard clearing procedure, we number all the individuals sequentially within a niche according to their fitness values. The individual with the highest fitness is assigned the number "1". After the numbering, we sort the archive from number "1" to the largest number so that the individuals with the highest fitness in every niche are on the top of the list and the individuals with the second highest fitness in every niche are next and so on.

Although we are going to replace the whole population with the top individuals from the sorted archive, we may still find some consistency of evolution as all the archive members are from the last 10 generations and there should surely be a certain number of them from the last generation. Our experiments have revealed that if we keep part of the population and replace the other part with the archive members, there will be some unnecessary redundancy. When a change is detected, we also count on the first archive to obtain the starting points for the next round.

The second archive relates to a higher level of diversity, of which we use the standard deviation of fitness values as the measure. It is generated in the same way as the first archive but only in the very beginning of the search and right after every detected change when the population is most diverse. In other words, we just maintain this archive until the next change occurs.

When the standard deviation of fitness in the population is successively very small, say, less than a predefined threshold for the last 50 generations, we shall replace 95 percent of the population with the members from the second archive, so that reasonably good and diverse members are stored. This means that for the 95 percent of the population the evolution is dated back to the very beginning of the current environmental state. We still keep 5 percent of the elites so that they can continue fine search without any impact.

In an effort to avoid repeatedly using the same individuals in the second archive, we use a probabilistic scheme when picking the archive members. As we have mentioned earlier, the archive is already properly sorted, so we pick the individuals from the archive sequentially with a fixed probability of 0.8. The individuals that are chosen out will be put to the bottom of the archive.

B. Detection of Changes

The detection of changes plays an important part in our algorithm. We must firstly detect the changes effectively so that we can take relevant measures. As is said in [14], we may use the deterioration of the population performance or the time-averaged best performance as an indicator; or we may re-evaluate several individuals and if the fitness of at least one individual has changed, we recognize it as a change.

As far as our experiments are concerned, we adopt the second strategy. The experimental results show that it is very efficient provided that the environmental change has an influence on every individual. The "interesting points" we focus on are the top three solutions in first archive (after clearing) as they are going to get into the population as soon as a change is detected. We do so with the purpose to decrease the wastage of function evaluations.

C. When Population Totally Loses Diversity

There are times that the population will lose diversity totally. All the individuals are converging to a single point. This can be disadvantageous as in a dynamic environment the optimum will probably move to another location sooner or later.

In our experiments, we found that partial restart is not enough to stop this situation. Reference [17] has also suggested that the main problem of partial restart is that it is

very hard for the newly introduced solutions to establish themselves when the population contains highly fit individuals. Therefore when we find that the standard deviation of fitness is extremely small, say, less than another predefined threshold, we are going to reinitialize all the population except the best one. Moreover, we set the strategy parameter η to the initial value and restart the "time" according to it.

V. Experimentation

We use the test suite provided by CEC 2009 Competition on Evolutionary Computation in Dynamic and Uncertain Environments. There are 6 multimodal test functions, the mathematical definitions of which can be found in [23]. The first test problem F_1 is the dynamic peak generator with 10 and 50 peaks. The other problems F_2–F_6 are dynamic composition functions. Seven change instances are examined. They are small-step changes (T_1), large-step changes (T_2), random changes (T_3), chaotic changes (T_4), recurrent changes (T_5), recurrent changes with noise (T_6), and random changes with changed dimensions (T_7). They make a total 49 test instances. For every test instance, 60 changes are involved in each run and a problem will be tested on 20 independent runs.

We use the population size of 100 for all the test problems. 20% of the function evaluations are assigned to local search with 5 directions applied to 4 individuals, and the rest of function evaluations are for generating offspring by the UEP algorithm. The tournament size q for the selection phase of EP is 10. Other parameters of our algorithm include the interval of archive updating, the initial temperature of η, the clearing radii in the archives, and the thresholds with regard to premature convergence and complete loss of diversity. The first three are set to be 10 generations, 6, and 5 respectively. The thresholds can be set to approximately 1/100 and 1/1000 of the initial fitness deviation for the cases of premature convergence and loss of diversity. The computers we use for simulation have the following configuration.

TABLE I
Computer Configuration

CPU	P4, 3000 MHz
Memory Total	4GB
Operating System	Linux
Programming Language	MATLAB 7.4 (R2007a)

VI. Results

All the environmental changes are successfully detected in our experiments. The absolute function errors [23] after reaching maximum number of fitness evaluations for all the changes are recorded. Average best, average mean, average worst values and standard deviations of the error values are listed in Tables II–VII. Fig. 2–8 show the convergence graphs for each problem (median run) for dimension n=10. The overall performance of the algorithm is summarized in Table VIII.

TABLE II
Error Values Achieved for F_1

Peaks (m)	Errors	T_1	T_2	T_3	T_4	T_5	T_6	T_7
10	Avg_best	0.0054	0.00445	0.00435	0.0057	0.01105	0.0104	0.0032
	Avg_worst	35.009	51.032	47.041	13.96	47.763	54.099	52.805
	Avg_mean	5.7109	10.658	10.87	1.5033	8.2954	8.232	13.123
	STD	9.6761	13.851	13.499	3.0008	13.227	13.102	14.96
50	Avg_best	0.0063	0.00535	0.00505	0.00585	0.0197	0.0164	0.00375
	Avg_worst	26.538	50.227	44.899	13.497	21.09	27.041	43.844
	Avg_mean	5.7391	13.285	15.896	1.4109	2.2653	3.1577	12.703
	STD	6.8424	12.944	13.365	2.4466	4.239	5.6002	11.917

TABLE III
Error Values Achieved for F_2

Errors	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Avg_best	0.1266	0.1383	0.13615	0.132	0.12985	0.1195	0.0896
Avg_worst	38.758	45.346	29.778	32.751	34.247	35.26	38.11
Avg_mean	6.2147	7.2236	4.9885	4.2067	3.5058	3.478	6.7124
STD	9.6292	11.024	8.245	7.5828	7.3318	7.5956	10.274

TABLE IV
ERROR VALUES ACHIEVED FOR F_3

Errors	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Avg_best	0.1996	0.17025	0.17975	0.20085	0.16635	0.1431	0.1243
Avg_worst	512.53	504.83	501.49	555.05	507.77	506.33	492.96
Avg_mean	151.98	140.47	136.67	164.96	95.123	107.54	107.69
STD	190.71	182.71	183.86	216.4	151.82	158.36	160.33

TABLE V
ERROR VALUES ACHIEVED FOR F_4

Errors	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Avg_best	0.13325	0.1386	0.13335	0.13045	0.13	0.1118	0.0814
Avg_worst	37.581	47.009	36.414	34.924	31.496	35.28	46.404
Avg_mean	6.601	8.1906	7.1991	5.0355	3.121	3.5162	8.3141
STD	10.032	11.923	10.145	8.3325	6.6867	7.3484	11.558

TABLE VI
ERROR VALUES ACHIEVED FOR F_5

Errors	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Avg_best	0.20075	0.18235	0.19615	0.2484	0.2035	0.184	0.17445
Avg_worst	44.887	54.133	36.438	39.928	55.669	56.092	56.026
Avg_mean	7.9042	10.091	7.2867	6.2507	8.2195	7.9011	10.779
STD	11.287	13.28	10.201	10.116	13.016	12.911	13.533

TABLE VII
ERROR VALUES ACHIEVED FOR F_6

Errors	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Avg_best	0.1493	0.14815	0.15235	0.15565	0.1404	0.12375	0.13875
Avg_worst	94.921	73.431	58.111	50.242	100.68	84.709	237.13
Avg_mean	17.303	18.732	16.005	11.753	26.311	24.558	25.231
STD	22.801	19.006	17.397	13.594	29.318	26.794	48.063

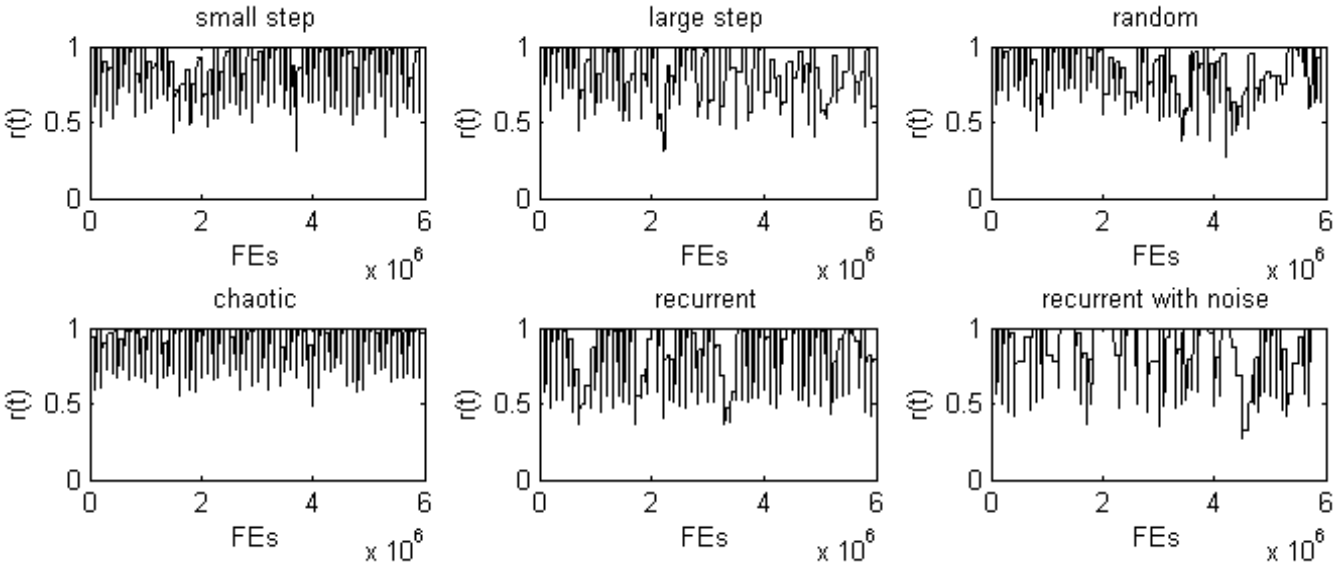

Fig. 2. Convergence Graph for F_1 (10 peaks)

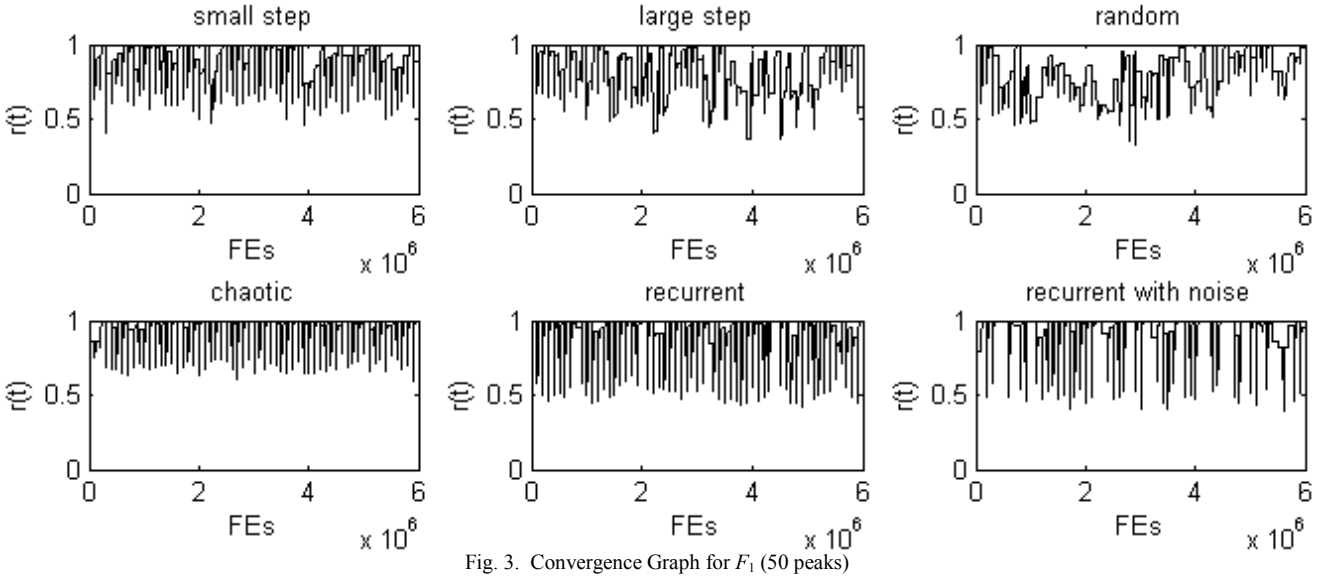

Fig. 3. Convergence Graph for F_1 (50 peaks)

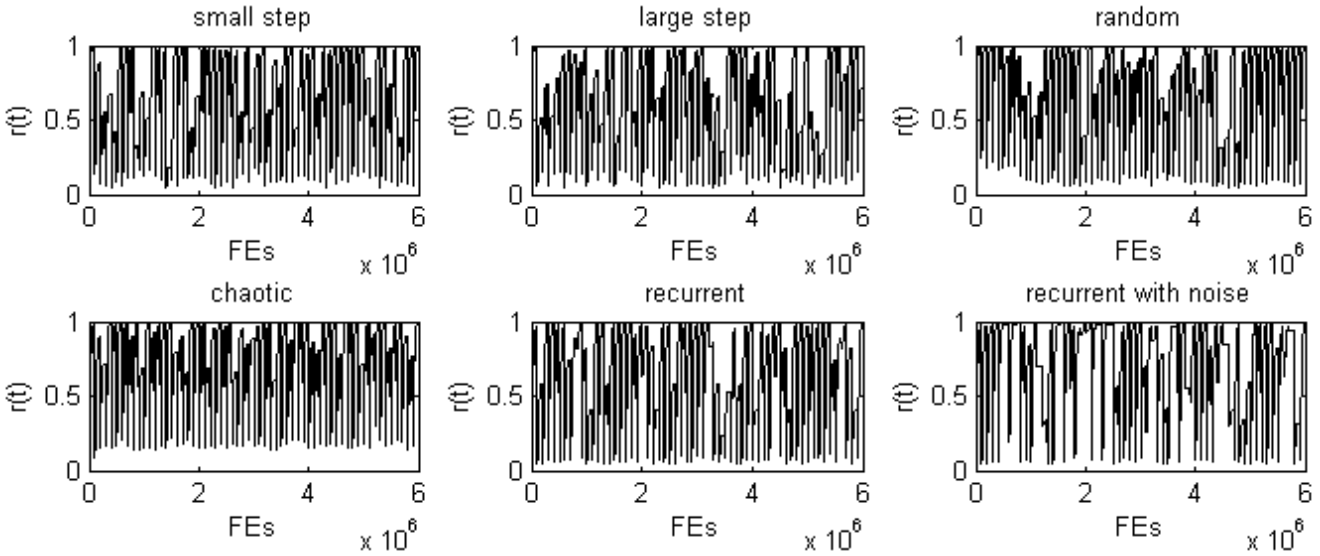

Fig. 4. Convergence Graph for F_2

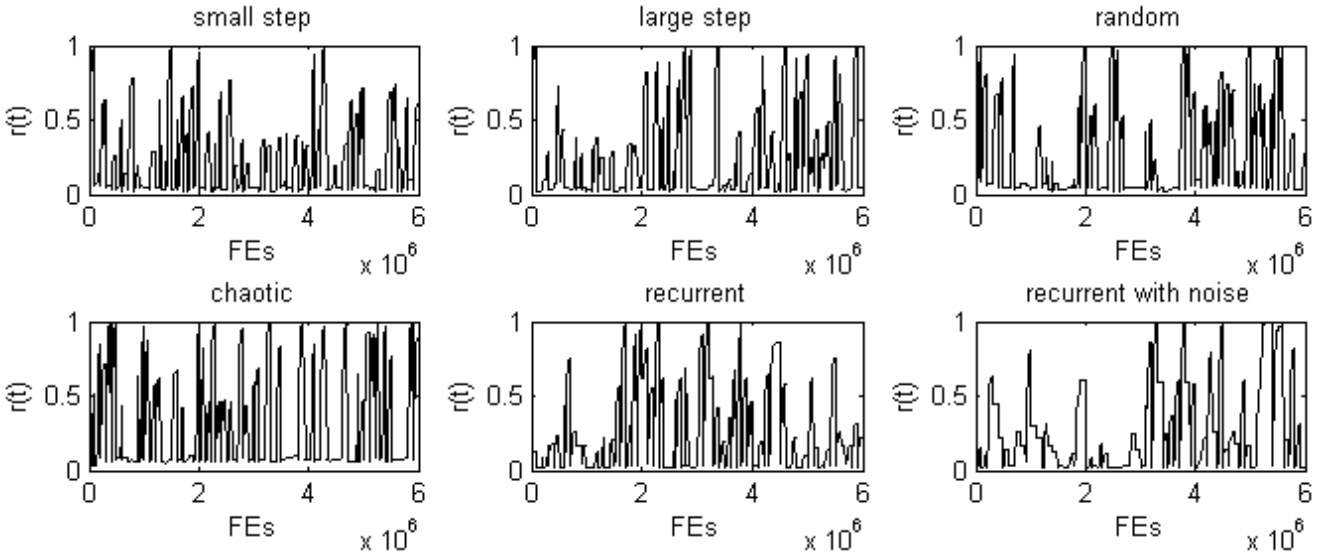

Fig. 5. Convergence Graph for F_3

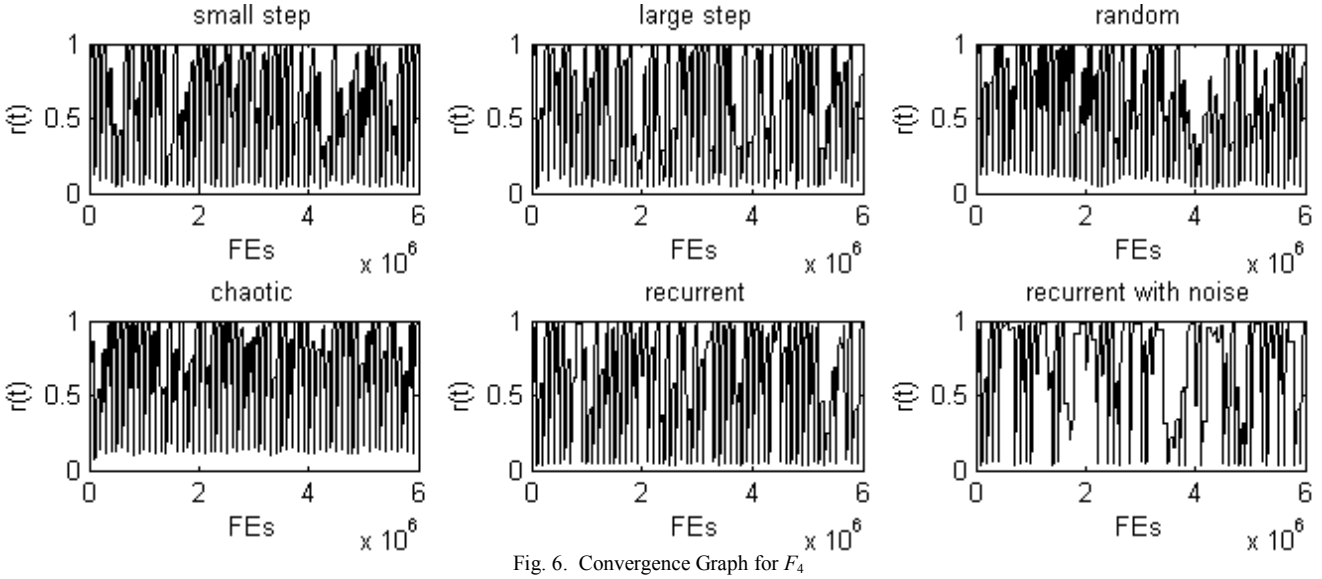

Fig. 6. Convergence Graph for F_4

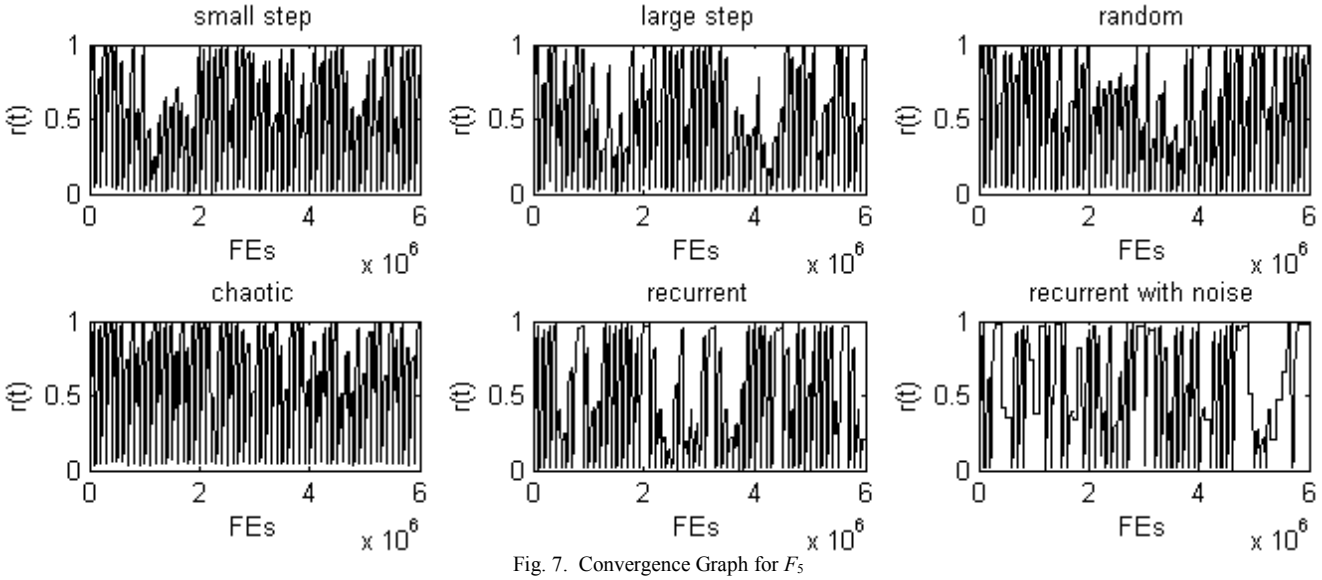

Fig. 7. Convergence Graph for F_5

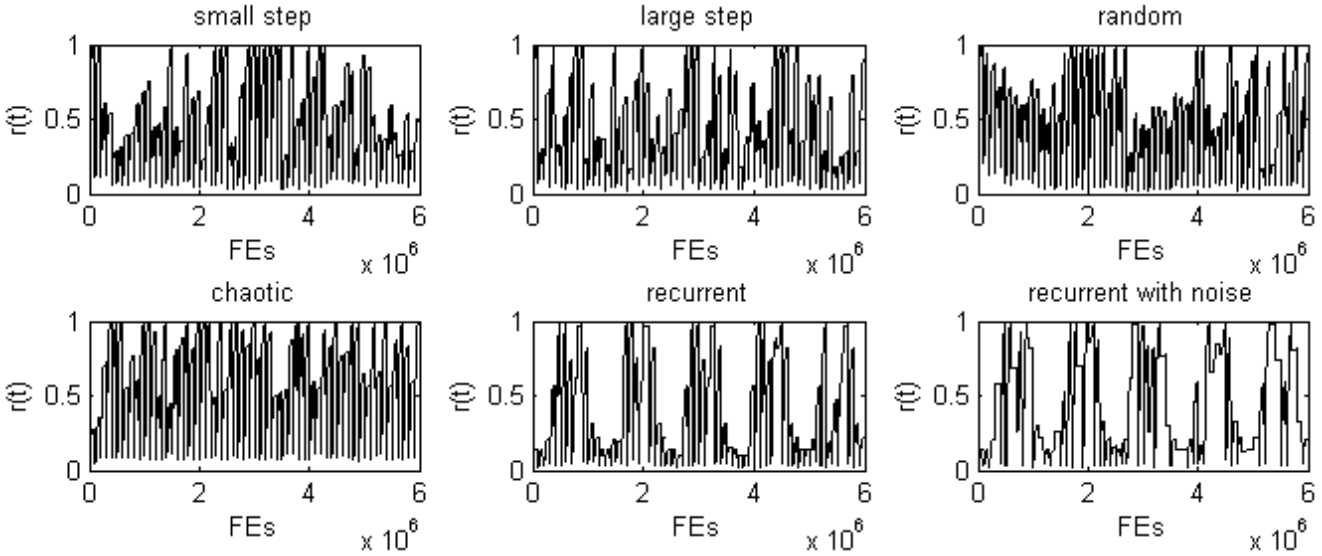

Fig. 8. Convergence Graph for F_6

TABLE VIII
ALGORITHM OVERALL PERFORMANCE

	$F_1(10)$	$F_1(50)$	F_2	F_3	F_4	F_5	F_6
T_1	(0.85365)*0.015	(0.85275)*0.015	(0.62892)*0.024	(0.2391)*0.024	(0.6198)*0.024	(0.55021)*0.024	(0.46945)*0.024
T_2	(0.78253)*0.015	(0.74912)*0.015	(0.61966)*0.024	(0.24515)*0.024	(0.59625)*0.024	(0.52358)*0.024	(0.43583)*0.024
T_3	(0.76886)*0.015	(0.70595)*0.015	(0.66252)*0.024	(0.27313)*0.024	(0.62127)*0.024	(0.56062)*0.024	(0.46881)*0.024
T_4	(0.92328)*0.015	(0.92981)*0.015	(0.74413)*0.024	(0.33056)*0.024	(0.72009)*0.024	(0.65535)*0.024	(0.5996)*0.024
T_5	(0.82083)*0.015	(0.89972)*0.015	(0.65367)*0.024	(0.26922)*0.024	(0.65597)*0.024	(0.52499)*0.024	(0.37705)*0.024
T_6	(0.83403)*0.015	(0.9015)*0.015	(0.69926)*0.024	(0.26169)*0.024	(0.6832)*0.024	(0.56829)*0.024	(0.40828)*0.024
T_7	(0.72982)*0.01	(0.75187)*0.01	(0.63527)*0.016	(0.30215)*0.016	(0.59808)*0.016	(0.5208)*0.016	(0.43222)*0.016
Mark	0.0820	0.0831	0.1064	0.0437	0.1031	0.0895	0.0731
Performance(sum the mark obtained for each case and multiply by 100):							58.0939

From the above we can see that the proposed algorithm performs well in all the change instances of test functions F_2, F_4, and F_5. It is also very effective in change instances T_1, T_4, T_5, and T_6 of function F_1, but not equally advantaged in change instances T_2, T_3, and T_7. It has some difficulties in solving problem F_6, and it is not able to trace the changing optimum in F_3. While it is quite natural that the problems with small-step changes can be solved smoothly, it is interesting to see that the algorithm can handle chaotic changes as well. And we would also like to note that the recurrent changes are not meant to be easy for the algorithm because the optimal solution is definitely moving to another peak (in the sense of maximization problems) after every change. Nonetheless, the performance of the algorithm is not obviously flawed in such circumstances.

VII. CONCLUSION

We have proposed the evolutionary programming with an ensemble of explicit memories for dynamic multimodal optimization. Though we use EP as the main algorithm, our implementation of the memory archives is general enough to be applied to other algorithms such as genetic algorithms and particle swarm optimizers. We test the algorithm on the benchmarks of CEC 2009 Competition on Dynamic Optimization. It has showed suitability for the majority of the test problems, but further analysis is still needed before we can claim its success in dynamic optimization.

REFERENCES

[1] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, 1966.

[2] D. B. Fogel, *System Identification Through Simulated Evolution: A Machine Learning Approach to Modeling*. Ginn, 1991.

[3] D. B. Fogel, *Evolutionary Computations: Toward a New Philosophy of Machine Intelligence*. Wiley-IEEE Press, 1995.

[4] H.-P. Schwefel, *Numerical Optimization of Computer Models*. John Wiley & Sons, 1981.

[5] D. B. Fogel, L.J. Fogel and J.W. Atmar, "Meta-Evolutionary Programming," in *Proc of the 25th Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, CA, 1991, pp. 540–545.

[6] K.-H. Liang, X. Yao, and C. S. Newton, "Adapting Self-Adaptive Parameters in Evolutionary Algorithms," *Applied Intelligence*, vol. 15, No. 3, November 2001, pp. 171–180.

[7] C. MacNish and X. Yao, "Direction Matters in High-Dimensional Optimisation," in *Proc. Congress on Evolutionary Computation*, Hong Kong, 2008, pp. 2372–2379.

[8] B. Suman and P. Kumar, "A Survey of Simulated Annealing as a Tool for Single and Multiobjective Optimization," *J. of the Operational Research Society*, vol. 57, 2006, pp. 1143–1160.

[9] S. E. Carlson and R. Shonkwiler, "Annealing a Genetic Algorithm over Constraints," in *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics*, vol. 4, 1998, pp. 3931–3936.

[10] J. M. Górriz, C. G. Puntonet, J. D. Morales and J. J. delaRosa, "Simulated Annealing Based-GA Using Injective Contrast Functions for BSS," *Lecture Notes in Computer Science*, Springer, vol. 3514, 2005, pp. 585–592.

[11] L. Fang, P. Chen, and S. Liu, "Particle Swarm Optimization with Simulated Annealing for TSP," *Proc of the 6th Conference on 6th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases*, vol. 6, Corfu Island, Greece, 2007, pp. 206–210.

[12] S. Bandyopadhyay, S. Saha, U. Maulik, and K. Deb, "A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA," *IEEE Trans. Evolutionary Computation*, vol. 12, No. 3, June 2008, pp. 269–283.

[13] E. Triki, Y. Collette, and P. Siarry, "A Theoretical Study on the Behavior of Simulated Annealing Leading to a New Cooling Schedule," *European J. of Operational Research*, vol. 166, 2005, pp. 77–92.

[14] J. Branke, *Evolutionary Optimization in Dynamic Environments*, Springer, 2002.

[15] Y. Jin and J. Branke, "Evolutionary Optimization in Uncertain Environments—A Survey," *IEEE Trans. Evolutionary Computation*, vol. 9, No. 3, June 2005, pp. 303–317.

[16] H. G. Cobb, "An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-dependent Nonstationary Environments," Naval Res. Lab., Washington, DC, Tech. Rep. AIC-90-001, 1990.

[17] W. Cedeño and V. R. Vemuri, "On the Use of Niching for Dynamic Landscapes," in *Proc. Int. Conf. Evol. Comput.*, 1997, pp. 361–366.

[18] J. Branke, "Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems," in *Proc of the 1999 Congress on Evolutionary Computation*, vol. 3, 1999, pp. 1875–1882.

[19] S. Yang, "Explicit Memory Schemes for Evolutionary Algorithms in Dynamic Environments," *Evolutionary Computation in Dynamic and Uncertain Environments*, vol. 51, 2007.

[20] R. K. Ursem, "Multinational GAs: Multimodal Optimization Techniques in Dynamic Environments," in *Proc. Genetic and Evolutionary Computation Conf.*, 2000, pp. 19–26.

[21] M. Wineberg and F. Oppacher, "Enhancing the GA's Ability to Cope with Dynamic Environments," in *Proc. Genetic Evol. Comput. Conf.*, D. Whitley et al., Eds., 2000, pp. 3–10.

[22] A. Pétrowski, "A Clearing Procedure as a Niching Method for Genetic Algorithms," in *Proceedings of the 1996 IEEE Int. Conf. on Evolutionary Computation*, New York, USA, 1996, pp. 798–803.

[23] C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H.-G. Beyer, and P. N. Suganthan, "Benchmark Generator for CEC'2009 Competition on Dynamic Optimization," Tech. Rep., Univ. of Leicester, Univ. of Birmingham, Nanyang Technological Univ., 2008.