Handling Uncertainty with a Real-coded EA

Maumita Bhattacharya
SOBIT, Charles Sturt University
Australia - 2640
maumita.bhattacharya@ieee.org

ABSTRACT

Presence of uncertainty in the search environment of Evolutionary algorithms (EA) interferes with the evaluation and the selection process of EA and adversely affects the performance of the algorithm. Presence of noise also means fitness function can not be evaluated and it has to be estimated instead. Of the various approaches which been tried to handle uncertainty in EA search environment, the more familiar approaches are: introduction of diversity (hyper mutation, random immigrants, special operators); and incorporation of memory of the past (diploidy, case based memory) [6]. In [2], we proposed a method, DPGA (distributed population evolutionary algorithm) that uses a *distributed* population architecture to simulate a distributed, self-adaptive memory of the solution space. Local regression is used in each sub-population to estimate the fitness. In the current research, we further investigate performance of DPGA for noisy fitness function i.e. fitness of any solution is altered by the addition of a noise term $N(\mu, \sigma^2)$. 'Noisy' versions of few standard benchmark problems have been considered in the simulation runs of the DPGA algorithm.

Categories and Subject Descriptors

Computing Methodologies [I.2 **Artificial Intelligence**]: I.2.8 Problem Solving, Control Methods, and Search.

General Terms

Algorithms, Design, Performance.

Keywords

Premature convergence, evolutionary algorithm, uncertainty.

1. BACKGROUND

Many real world problems have to deal with certain degree of randomness or noise. Noise arises from different sources such as:

- Measurement errors in experiments. For example, in case of online adaptation of real world systems, some of the design parameters can be measured only real time.
- Stochastic nature of simulation processes. For example, simulation based optimization of large and complex systems that uses random numbers in the simulation. Use of random numbers causes fluctuation of the fitness value obtained by simulation.
- Use of small sample collected from a very large search space.

Evolutionary algorithms are known to be robust in presence of noise compared to other conventional search methods. This notion is also supported by the fact that evolutionary algorithms and genetic algorithm in particular have metaphor in natural evolution and that nature does not necessarily deal with perfect fitness. Population based search techniques such as evolutionary algorithms are robust in single objective problem domain in presence of noise as an EA possesses a high amount of *knowledge* of the search space and in turn the average performance of the population acts as a filter for the noise. Despite these positive traits, evolutionary algorithm's applications to optimization problems involving noisy or uncertain fitness functions pose problems that require special attention. Presence of noise in the fitness function could mean 'overestimation' of fitness of inferior candidates and similarly 'underestimation' of fitness of superior candidates at the time of selection. This is likely to result in reduced learning rate, inability to retain learnt information, limited exploitation and absence of gradual monotonous improvement of fitness with generations.

Considering EA's *natural* noise-resistant properties, the effects of noise can be minimized by simple methods such as re-sampling and adaptation of population size. More recent approaches use efficient averaging techniques, based on statistical tests, or local regression methods for fitness estimation. The major techniques used to curb the effect of uncertainty in EA can be summarized as: *Re-sampling* [7, 5, 6], *Conventional EA with increased population size* [5, 7], *Rescaled mutation* [1], *Thresholding* [7, 6], *Fitness value based on neighbouring individuals* [7, 10, 4, 9] and *Reduced resampling* [8, 7]. A comprehensive survey of various techniques to handle noisy environment with EA, including distributed population approach similar to our current work can be found in [6].

Despite being one of the most common approaches to cope with noise, re-sampling is computationally expensive and as Darwen points out in [5] improvement through re-sampling may reach a point of stagnation and may even be harmful in case of small population. Adaptive or large population size is computationally expensive due to additional solution evaluations. Techniques such as Thresholding require estimation of additional parameters such as the *threshold*, τ in case of *Thresholding*, which is not feasible in all cases.

Our proposed DPGA algorithm [2] focuses on uncertain problem domains where accuracy and robustness of solution are the major concerns. The DPGA framework involves a distributed population architecture with local regression employed in the sub-populations.

The organization of the paper is as follows. To make this paper self-sufficient, Section 2 briefly outlines the DPGA framework. Simulation details and discussions of results are given in Section 3. Finally, Section 4 concludes the paper along with some future research directions.

2. THE DPGA ALGORITHM

Algorithm 1: *Procedure DISTRIBUTED_POPULATION_GA*

```
1: begin
2: $t = 0$
3: Initialize population $P(t)$
4: Evaluate population $P(t)$
5: while (not<termination condition>)
6: begin
7:     $t = t + 1$
8: if (reorganizing generation)
9: begin
10: Evaluate population $P(t)$ using re-sampling
11: Evolve population $P(t)$ to create new generation with canonical GA mechanism
12: Self-organize new population to create pseudo-populations
13: end if
14: for (each eligible one of $n$ pseudo-populations $P(t)_{N=1,2,\ldots,n}$)
15: begin
16: Evaluate population $P(t)_N$ partly by local regression
17: Evolve canonical as per population GA
18: end for
19: end main
```

The DPGA framework uses distributed population architecture with local regression in the pseudo-populations. Distribution of

Figure 1. The DPGA algorithm.

the population allows tracking multiple peaks in the search space. Each pseudo-population maintains information about a separate region in the search space, thus acting as a distributed self adaptive memory. We use the term pseudo-population instead of subpopulation to emphasize on the fact that each one of these sub groups may not actually act as self-sufficient evolving populations. Two basic ideas guide the functioning of the proposed model:

- Retention of memory with distributed pseudo-populations in the search space i.e., the pseodo-populations should be able to track their moving peaks through time.
- Estimation of time variant noisy fitness by local regression in each subpopulation. Quadratic regression is used in the present work. However, linear regression could be used to reduce computation time.

The basic algorithm structure of DPGA is as described in Fig. 1.

Specific features of the proposed distributed population GA are as below:

- Unlike conventional multipopulation EAs [5] the distributed population GA does not maintain a main population alongside the subpopulations. Instead, DPGA switches from single population to multipopulation periodically.
- At *reorganizing generation* the pseudo-populations are merged together to regain the main population and evolution is carried out as per canonical GA mechanism. To correct the adverse effects of noise, actual fitness evaluation along with resampling is used at this stage.
- Next the regenerated main population dissolves into pseudo-populations by self-organization. This is essentially distribution of the candidate solutions into pseudo-populations based on specific criteria. A factor of *fitness* and *population size* decides the eligibility of a pseudo-population to obtain evolution right. An eligible pseudo-population then evolves by the canonical GA mechanism. Local regression is used to estimate the fitness values. Mutation rate depends on a factor of *fitness* and *population size* as well.
- Members of the non-eligible pseudo-populations either survive or eventually disappear after *reorganizing generations*.
- Retention of memory about the *moving peaks* in case of non-static functions is achieved through the pseudo-populations over specific durations until *reorganizing generations*. This is logical considering the dynamic nature of the phenotypic solution space.

Simulation details for DPGA are presented in Section 3 and Section 4.

3. SIMULATION DETAILS

This research investigates the performance of the DPGA model as against standard GA, differential evolution, particle swarm optimization for a set of benchmark functions with noise incorporated. For analysis purpose, experiments have been conducted on the non-noisy versions of the same set of benchmark functions as well.

3.1 Test Functions

We have used the following benchmark functions to test the algorithms: Sphere function (5D), Griewank's function (50D), Rastrigin's function 1 (50D) and the Rosenbrock's function (50d). These benchmark functions have previously been used by researchers to test performance of evolutionary algorithm for noisy optimization problems [4]. All the benchmarks used are minimization problems. The descriptions of the functions are as given in [2].

The noisy versions of the above set of functions are defined as:

$$f_{Noisy}(\vec{x}) = f(\vec{x}) + N(\mu, \sigma^2) \tag{1}$$

where, $N(\mu, \sigma^2)$= Standard Normal (or Gaussian) distribution with mean, μ = 0 and variance, σ^2 = 1 in Stage 1 of the experiments. The probability density function $f\left(x;\mu,\sigma^2\right)$ is given as below,

$$f\left(x;\mu,\sigma^2\right)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2}$$

In Stage 2 of the experiments, we have expanded the simulation with various values of the noise's standard deviation σ .

3.2 Experiment Settings

We have conducted two sets of experiments. For the *first set of experiments*, the parameter settings used for the algorithms are as shown in Table 1. For comparison purpose we have used the results reported in [7] for conventional GA, differential evolution (DE) and particle swarm optimization (PSO). Hence similar parameter settings have been used. As Krink et al. in [7] have reported, the parameters of the heuristic algorithms were not tuned for each test problem. The reason stated is that in case of real life problems such tuning can be infeasible due to time constraints.

Let $totalEval$= total number of function evaluations, $popSize$= population size, $totalIT$ = total number of iterations, rs= total number of candidate solution resampling, and $totalIUnchanged$=total number of individuals that remained unchanged (such as the elites) and were not reevaluated in the various generations. Then,

$$totalEval = popSize * totalIT * rs - totalIUnchanged \tag{3}$$

The total number of function evaluations in [7] in case of conventional GA, differential evolution and PSO is kept constant by keeping $totalIT \propto \frac{1}{rs}$. However, this has not been followed for the proposed method. We have used a fixed number of iteration in these experiments. The experiments were run with different values of rs such as rs =1, 5, 20, 50 and 100, to eradicate the effects of noise and find the 'true' fitness. The 'true' fitness here refers to the fitness value obtained by evaluation of the non-noisy versions of the same functions.

In the *second set of experiments*, the population size has been increased as follows: 200 for the five dimensional problems and 600 for the fifty dimensional problems respectively, with rs =1. As before, the experiments were run for a fixed number of iterations.

3.3 Discussions

Table 2 presents the final results obtained for the chosen benchmark functions (both noisy and non-noisy versions) with the *first set of experiments*. The results for the non-noisy versions of the functions have been reported here mainly as 'standards' to judge the impact of noise. For comparison purpose we have used the results reported in [7] for the following methods: differential evolution, particle swarm optimization (PSO) and conventional EA (CGA). As mentioned earlier, the total number of function evaluations has been kept fixed for the experiments with differential evolution, PSO and conventional EA [7]. The proposed method used variable number of function evaluations.

Not surprisingly all the heuristics have performed considerably better on the non-noisy benchmark functions compared to the noisy versions of the same functions. As can be observed from the results (Table 2), while the proposed method has performed better for majority of the test cases, the difference is not necessarily significant in all cases of the low dimensional Sphere function.

Table 1. Parameter settings[**] of the algorithms.

Proposed Method	Canonical GA	Differential Evolution	PSO
popSize=100	popSize=100	popSize=50	popSize=20
p_c **=1.0**	p_c =1.0	CF =0.8	w = 1.0 $\rightarrow$ 0.7
p_m **=0.3**	p_m =0.3	f =0.5	$\varphi_{\min}$ =0.0
n **=10**	n =10		$\varphi_{\max}$ =2.0
σ_m **=0.01**	σ_m =0.01		

**popSize=Population Size, p_c = Crossover rate (EA), p_m = Mutation Rate (EA), n = Number of Elites, σ_m = Mutation Variance, CF = Crossover Factor (in differential Evolution), f =Scaling Factor, w =Inertia Weight (the value is linearly decreased from 1.0 to 0.7 during the run), $\varphi_{\min}$, $\varphi_{\max}$ = Lower and Upper Bounds of the Random Velocity Rule Weights.

However, for the high dimensional (50D) Griewank, Rastrigin and Rosenbrock functions the differences are clearly significant. In case of the noisy versions, similar to the observation reported in [7], resampling has prevented stagnation in all the four test functions, regardless of whether the mean final result has been improved. However, the effect of rate of resampling is rather inconclusive as it varied from case to case rather randomly. Only conclusion that can be drawn is that the effect of rate of resampling may be problem dependent. From the results reported in [7] it is obvious that in case of conventional EA, increased number of resampling can not improve the performance when number of iterations is inversely proportional to number of resampling to keep the total number of function evaluations constant. However, interestingly enough, for most of the test cases moderate rate of resampling has helped to improve the solution, while high to very high rate of resampling has rather a deteriorating effect on the solutions.

In case of the *second set of experiments*, the success rates of DPGA algorithm for simulations with noisy functions for various values of noise's standard deviation are summarized in Table 3.

Table 2. Performance (Average Best Fitness) Comparison for DPGA, CGA, Differential Evolution and PSO. rs =number of candidate solution resampling.

		DPEA	CGA	Differential Evolution	PSO
f_1 =Sphere function (5D non-noisy) And f^*_1=Sphere function (5D noisy)	f_1	4.21334E-75±0	6.71654E-20±0	4.12744E-152±0	2.51130E-8±0
	$f_1^*, rs=1$	0.00103E-2±0.003	0.04078±0.00543	0.25249±0.02603	0.36484±0.05182
	$f_1^*, rs=5$	0.00023E-2±0.001	0.02690±0.00363	0.13315±0.01266	0.16702±0.03072
	$f_1^*, rs=20$	0.00018E-2±0.023	0.02205±0.00290	0.07364±0.00811	0.11501±0.01649
	$f_1^*, rs=50$	0.00001E-2±0.0031	0.01765±0.00233	0.07004±0.00686	0.06478±0.00739
	$f_1^*, rs=100$	0.00011E-2±0.0009	0.03929±0.00396	0.08165±0.00800	0.07135±0.00938
f_2 =Griewank function (50D non-noisy) And f_2^*=Griewank function (50D noisy)	f_2	4.00E-7±0.001	0.00624±0.00138	0±0	1.54900±0.06695
	$f_2^*, rs=1$	0.00211E-1±0.001	1.14598±0.00307	3.31514±0.07388	11.2462±0.50951
	$f_2^*, rs=5$	0.00211E-1±0.001	1.10223±0.00342	2.42183±0.03616	16.6429±0.70800
	$f_2^*, rs=20$	0.00011E-1±0.021	1.44349±0.01381	2.67093±0.03895	85.4865±2.13148
	$f_2^*, rs=50$	0.00200E-1±0.323	3.69626±0.13127	46.8197±0.96449	143.021±2.33228
	$f_2^*, rs=100$	0.00301E-1±0.481	18.0858±0.99646	233.802±6.25840	194.188±4.90959
f_3 =Rastrigin's function 1 (50D non-noisy) And f_3^*=Rastrigin's function 1 (50D noisy)	f_3	1.219E-7±0.0013	32.6679±1.94017	0±0	13.1162±1.44815
	$f_3^*, rs=1$	0.03011E-1±0.031	30.7511±1.32780	2.35249±0.06062	55.9704±2.19902
	$f_3^*, rs=5$	0.01071E-1±0.011	31.4725±2.02356	14.0355±0.47935	160.500±2.67500
	$f_3^*, rs=20$	0.01171±0.111	39.1777±2.11529	167.628±2.12569	313.184±3.93659
	$f_3^*, rs=50$	0.01092±0.100	74.8577±2.69437	314.762±2.88650	380.178±4.88706
	$f_3^*, rs=100$	0.01232±0.190	147.800±2.93208	438.036±3.67504	418.265±5.35434
f_4 =Rosenbrock's function (50D non-noisy) And f_4^*=Rosenbrock's function (50D noisy)	f_4	1.4011E-14±0.0011	79.8180±10.4477	35.3176±0.27444	5142.45±2929.47
	$f_4^*, rs=1$	1.4201E-4±0.0021	118.940±13.2322	47.6188±0.15811	4884.68±886.599
	$f_4^*, rs=5$	1.1201E-4±0.0091	341.788±49.6738	47.0404±0.13932	368512±39755.5
	$f_4^*, rs=20$	1.0001±0.2191	1859.06±261.844	7917.46±352.851	1.61E+7±1.18E+6
	$f_4^*, rs=50$	1.0091±0.1393	35477.7±4656.17	1.65E+7±903677	5.57E+7±2.38E+6
	$f_4^*, rs=100$	2.0028±2.1999	257488±19371.2	2.98E+8±1.04E+7	1.17E+8±7.38E+6

Table 3. Analysis of Success Rates of the DPGA Algorithm for the Noisy Versions of the Test Functions with Various Values of Noise's Standard Deviation σ .

σ	f_1^* $rs=1$	f_2^* $rs=1$	f_3^* $rs=1$	f_4^* $rs=1$
0	100%	100%	100%	100%
0.1	75%	70%	65%	100%
0.2	88%	85%	75%	100%
0.3	88%	75%	75%	95%
0.4	76%	70%	65%	95%
0.5	58%	55%	58%	95%
0.7	40%	50%	53%	90%
0.9	75%	70%	69%	95%

Simulations were repeated 100 times for each noise standard deviation σ value. The reported results are for the $rs=1$ simulation cases. Here '*success*' has been defined slightly differently for the different test functions. The performance (Average best Fitness) obtained with DPGA, in the *second set of experiments* is as summarized in Table 4. As can be observed, DPGA has shown a significant improvement in performance in all the four test cases with the increase in population size. However, the rate of improvement in performance is different for the various test cases.

4. CONCLUSIONS

Various methods have been proposed in literature to deal with problem of uncertainty in search environment. One such approach is that of the multi-population approach where the subpopulations are used to retain information about the changes in the search environment. The DPGA framework [2] investigated in this paper is similar to the multi-population approach in that it divides the search or solution space into multiple pseudo-populations and retains information about the *changes* in them. However, the DPGA algorithm applies superior mechanism in that this reduces the computational expense of maintaining the main population along with the subpopulations by switching between main and subpopulations at regular intervals. Also just enough retention of memory is allowed by dissolving the subpopulations periodically. Performance of the framework has been found to be satisfactory. However, further investigation is needed to resolve issues such as:

- How to ensure the optimal number and size of the pseudo-populations?
- The interval for the reorganizing generation is currently based on statistical information about the benchmark test functions. How to base this on the characteristics of the evolving phenotypic solution space keeping the computational overhead within acceptable limits.

Further investigation will also be conducted to reach some definitive conclusion on the optimal rate of re-sampling.

5. REFERENCES

[1] A. Ratle.., "*Accelerating the convergence of evolutionary algorithms by fitness landscape approximation*", Parallel Problem Solving from Nature-PPSN V, Springer-Verlag, pp. 87-96, 1998.

[2] M. Bhattacharya, "*DPGA: a Simple Distributed Population Approach to Tackle Uncertainty*", accepted to be published in Proceedings of the 2008 IEEE Congress on Evolutionary Computation (CEC 2008), Hong Kong.

[3] M. Bhattacharya, *"Exploiting Landscape Information to Avoid Premature Convergence in Evolutionary Search"*, Proceedings of the 2006 IEEE Congress on Evolutionary Computation, Vancouver, Canada, 0-7803-9487-9/06, IEEE Press, 2006 , pp. 2575-2579.

[4] J. Branke, C. Schmidt, and H. Schmeck, *"Efficient fitness estimation in noisy environments"*, Proceedings of the Genetic and Evolutionary Computation Conference 2001, GECCO2001, Morgan Kaufman, 2001, pp. 243 – 250.

[5] P. J. Darwen, *"Computationally intensive and noisy tasks: co-evolutionary learning and temporal difference learning on Backgammon"*, Proceedings of the 2000 Congress On Evolutionary Computation, CEC2000, Volume 1, 2000, pp. 872 – 879.

[6] Y. Jin, and J. Branke, *"Evolutionary optimization in uncertain environments - A survey"*, IEEE Transactions on Evolutionary Computation, 9(3), 2005, pp. 303 – 317.

[7] T. Krink, B. Filipic, and G. B. Fogel, *"Noisy optimization problems - a particular challenge for differential evolution?"*, Proceedings of IEEE Congress on Evolutionary Computation, CEC2004, Volume 1, 2004, pp.332 – 339.

[8] G. Rudolph, *"A partial order approach to noisy fitness functions"*, Proceedings of the Congress on Evolutionary Computation, CEC2001, IEEE Press, USA, 2001, pp. 318 – 325.

[9] Y. Sano, and H. Kita, *"Optimization of noisy fitness functions by means of genetic algorithms using history of search with test of estimation"*, Proceedings of the 2002 Congress on Evolutionary Computation, CEC2002, IEEE Press, USA, pp. 360 – 365.

[10] H. Tamaki, and T. A. Arai, *"A genetic algorithm approach to optimization problems in an uncertain environment"*, Proceedings of International Conference on Neural Information Processing and Intelligent Information Systems, Volume 1, 1997, pp. 436-439.

Table 4. Performance (Average Best Fitness) Obtained with DPGA. In Case of the experiments with Noisy Versions of the Problems rs =1 has been used. Here, rs is the number of candidate solution resampling.

	Non-noisy	**Noisy**
Sphere function (5D)	4.0019E-78±0	0.00115E-36±0.001
Griewank function (50D)	4.00E-39±0.001	0.00101E-21±0.001
Rastrigin's function 1 (50D)	3.115E-39±0.0011	0.01501E-21±0.011
Rosenbrock's function (50D)	1.911E-30±0.0011	1.5101E-18±0.0011