# Evolutionary Optimization guided by Entropy-based Discretization

Guleng Sheri and David W. Corne

Department of Computer Science, Heriot-Watt University, Ediburgh, UK
gls3@macs.hw.ac.uk, dwcorne@gmail.com

**Abstract.** The Learnable Evolution Model (LEM) involves alternating periods of optimization and learning, performa extremely well on a range of problems, a specialises in achieveing good results in relatively few function evaluations. LEM implementations tend to use sophisticated learning strategies. Here we continue an exploration of alternative and simpler learning strategies, and try Entropy-based Discretization (ED), whereby, for each parameter in the search space, we infer from recent evaluated samples what seems to be a 'good' interval. We find that LEM(ED) provides significant advantages in both solution speed and quality over the unadorned evolutionary algorithm, and is usually superior to CMA-ES when the number of evaluations is limited. It is interesting to see such improvement gained from an easily-implemented approach. LEM(ED) can be tentatively recommended for trial on problems where good results are needed in relatively few fitness evaluations, while it is open to several routes of extension and further sophistication. Finally, results reported here are not based on a modern function optimization suite, but ongoing work confirms that our findings remain valid for non-separable functions.

**Key words:** function optimization, hybridization, learning

## 1 Introduction

The Learnable Evolution Model (LEM) [13] is a general hybrid approach, in which the overall idea is to combine evolutionary search and learning, where 'evolution' periods are informed by previous 'learning' periods, E.g., following $n$ generations of an evolutionary algorithm (EA), we might then apply learning (perhaps a neural network, or a decision tree learner) which will exploit the information gained so far by trying to learn to predict fitness (or categories of fitness, such as 'good' and 'bad') from vectors of gene values. The result of the learning phase is then used somehow to inform the next period of evolution. The way in which learning influences evolution is not restricted. For example, the learned mapping from gene values to fitness could be used to predict the fitness of children before they are evaluated, and the evolution phase then discards, without evaluation, children that are predicted to be of poor fitness. Alternatively the learned model may be used to constrain the genetic operators in such a way that children are more likely to be fit. Or, the learned model may be used to 'repair' children, and so on.

In [13], the learning method was AQ15 [11, 16], and the results of this, along with the results emerging from continuing development of LEM from Michalski's group [17, 18], were highly promising. In particular LEM led to dramatic speedup on a range of function otpimization problems as well as on a real-world problem. In other work inspired by LEM, a multiobjective version (using C4.5 as the learner), significantly accelerated and improved solution quality for large-scale problems in water distribution networks [10].

Of course, the essential idea of combining learning and evolution is not restricted to LEM. While LEM emerged from the machine learning community, Estimation of Distribution Algorithms (EDAs) blossomed in the evolutionary computation community [12]. Both LEM and EDA now have several published variants (particularly EDA variants), and it is interesting to contrast the two. While EDAs focus on modelling as the key force behind search (i.e. search is guided closely by the modelling, with new sample points in the space generated directly from the model), in LEM the evolutionary component is most responsible for the search (i.e. new points are sampled mainly in the familiar way by using genetic operators on a population of chromosomes), with guidance from learning processes. Most interestingly, recent results compare EDAs and LEM3 directly [18], using a variety of EDA implementations [5]. Comparisons showed LEM3 (with AQ) from 15–230 times faster than EMNA_*global* , and always achieveing a superior final solution. Also, recent years have seen the appearnace of *hybrids* of EDAs and other search methods [19, 14, 20]. When contrasting LEM with EDA, it is sensible to say that LEM is similar to a hybrid of an EDA and an EA; this seems consonant with the success that has so far been reported for EDA/EA hybrids.

Clearly, more research is warranted towards understanding the design and application of algorithms within the LEM framework. In previous work [4], the authors explored how well a LEM algorithm performs when the learning technique is perhaps the simplest possible such mechanism: $k$-nearest neighbour. In LEM(KNN), the learning phase comprised only the process of identifying the best and worst 30% of the current population; subsequent evolution only allowed evaluation of children for whom the majority of their 5 nearest neighbours from these sets were in the higher-fitness group. Although not able to produce results that rivalled those of LEM(AQ) [13], this simple intervention of learning provided very significant speedup and solution quality improvements over the unchanged EA. In this paper the investigation of alternative learning methods is continued by the use of a quite different approach. We use the concept of *entropy-based discretization* (ED), whereby, for each parameter in the search space, we infer from recent evaluated samples what seems to be a 'good' interval for that parameter. Subsequent periods of evolutionary search are biassed towards exploring these 'good' intervals. Over a range of function optimization problems, we find that the resulting LEM(ED) methods again provide significant advantages in both solution speed and quality over the unadorned evolutionary algorithm, and on most problems it is also superior to CMA-ES when the number of evaluations

is limited. Though details are not included here, LEM(ED) is also significantly better than LEM(KNN).

In the remainder we continue as follows. Section 2 provides more detail on LEM and on our LEM(ED) variation(s). Section 3 covers experiments and results on some test problems, and we conclude and discuss in section 4.

## 2 LEM and LEM(ED)

### 2.1 The LEM Framework

In LEM(AQ) [13], an initial population is divided into high-performance (H-group) and low-performance (L-group) groups according to fitness, to be used as positive and negative training examples for AQ learning. The outcome of learning is a set of rules which predict a class label (i.e. H-group or L-group) for a chromosome. LEM(AQ) then proceeds with an otherwise normal EA, except that the operators generate new individuals only with gene values within the ranges sanctioned by the recently learned rules. LEM(AQ) then continues for a specified amount of generations, and then pauses for more learning based on the current population. This feeds into the next stage of evolution, and so on. LEM(AQ) has many additional details that mediate the transitions between learning and evolution, and we refer readers to [13] for more details.

LEM(AQ), thus overviewed, is simply an instantiation of the wider LEM framework, which allows for creativity in the choices of learning method, and the way in which learning and evolution interact. Here, we continue an investigation of simple instantiations of this framework. The key details in the instantiation we use in this paper are as follows: the learning method is entropy-based discretization (described next); we tightly interleave learning and exploration in the initial generations, and then we cease further learning entirely; finally, the way in which learning affects evolution, similarly to the LEM(AQ) case described above, is that we constrain the generation of new chromosomes according to the learned model.

### 2.2 Entropy-based discretization(ED)

We use entropy-based supervised binary discretization [1], taken from the machine learning literature. As is well-known, for example, ID3[2] and C4.5[3] use entropy measures to guide the development of a decision tree. In particular, in data mining there is often a need to discretize the range of a continuous parameter (into, for example, two intervals labelled 'low' and 'high'). This is done by considering all possible cutpoints for the boundary between the intervals, and calculating an entropy measure for those cutpoints. In the case of data mining, entropy is based on the way in which the target class values are distributed over the two intervals, and the cutpoint that provides the best entropy measure is chosen. In our usage of ED, we basically do the same, but considering 'good fitness' and 'bad fitness' to be the target class values. I.e. we choose a cutpoint that seems to best separate 'good' from 'bad' in the current population.

### 2.3 The LEM(ED) algorithm

As with LEM(AQ), LEM(ED) divides the *current population* into high-performance (H-group) and low-performance (L-group) groups according to their fitness values and a given *threshold* (say, 30% – that is, the fittest 30% form the H-group and the worst 30% form the L-group). This is then saved as the *learning population*. Individuals of the H-group and L-group in the *learning population* form the training examples used by ED. ED is applied to each parameter $i$ independently; the output of ED for parameter $i$ is the pair of intervals $[min_i, cut\_point_i]$ and $[cut\_point_i, max_i]$ where $cut\_point_i$ is the splitting point with minimum entropy, and $min_i$ and $max_i$ are the lower and upper limits specified for parameter $i$.

The two intervals in each such pair are then labelled *good* and *bad* (the good interval is the one with the highest proportion of points from the H-group). Figure 1 illustrates this. On the left, we show the cutpoint (and hence division into good and bad intervals) that might be learned from a population of fitnesses for a given parameter. In the case on the left, the good interval happens to contain the value of that parameter that appears in the global optimum; in the case on the right, the optimum is missed by the good interval. The idea of LEM(ED) is to guide evolution by biassing further search within the 'good' intervals, and this is, of course, appropriate if the case on the left occurs more often than the case on the right, or if not, we should allow enough exploration and adaption to 'recover' from mislabelled intervals. We have done various investigations around this issue and expect good adaptive schemes to emerge in future, but in the present work we consider only two basic variants of LEM(ED). In each one, learning of intervals ceases after a generation that has seen no improvement in the best solution. In LEM(ED)1, further generations continue with the underlying EA (unconstrainted), and in LEM(ED)2, further generations continue with the underlying EA, but biassed by the last learned intervals.

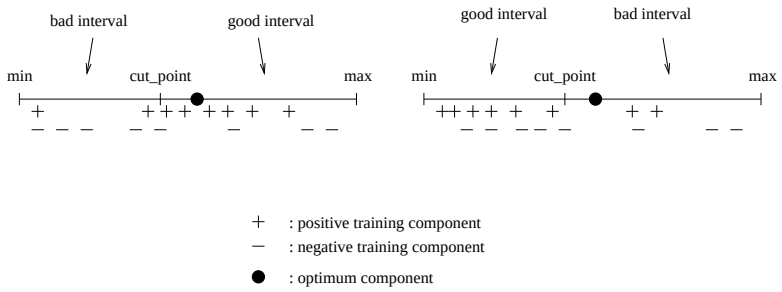


**Fig. 1.** Illustrating entropy-based discretization, showing correctly (left) and incorrectly (right) labelled interval pairs.

When the intervals for all parameters are labelled, LEM(ED) begins the instantiation procedure. New individuals for the next generation are now generated according the *good* intervals. In the current work, we achieve this by generating

new individuals uniformly at random from one of the intervals, using a high probability to choose the good interval.

When the new population is created, ED is applied again to the current population, and the process of ED and generation of new individuals until the latest instantiation step leads to no improvement in best fitness. When the learning mode is finished, we continue with an ordinary EA. In the case of LEM(ED)1, the EA benefits only from a good (hopefully) initial population emerging from the learning phase just described. In LEM(ED)2, the EA additionally uses the final set of learned intervals, biassing the generation of new individuals.

'Overview' pseudo-code for LEM(ED) is as follows:

1. Set values for *population size*, mutation probability, crossover probability, *learning threshold* and set elite-preserve operator option.
2. Generate initial population: Choose a method to create the initial population with *population size* and evaluate this population.
3. Begin learning mode :
   (a) Derive extrema: Copy the *current population* as the *learning population*, from which create the high fitness group (H-group) and low fitness group (L-group) according to fitness values and *threshold*. These two groups are stored as positive and negative training data for learning algorithm.
   (b) Apply ED: For each parameter, consider each value as the potential cut-point and calculate the information gain for each such point, choosing the best as the cut_point for this parameter. The output of this step on each dimension is two intervals having the form $< min, cut\_point >$, and $< cut\_point, max >$.
   (c) Label the learned intervals: For the output intervals on each parameter, label them as *good* and *bad* intervals by observing the relative proportions of gene values from the H-group and L-group.
   (d) Instantiate new individuals: After the discretization and labelling procedures, the new individuals for next generation are generated from the good intervals in each dimension.
   (e) evaluate each of the new individuals; if there has been no improvement in best fitness, terminate learning and switch to EA mode.
   (f) Update H-group and L-group: When the new population is generated, the H-group and L-group are recalculated. Return to step (b).
4. Begin evolution mode: The EA can of course be freely designed, and may or may not use the intervals generated at the end of the learning phase.

LEM(ED) will terminate if the optimum (if known) is reached, or the maximum allowed number of generations is reached. In LEM(ED)1, the evolution phase runs a straightforward EA, described next, whose initial population is the output of the learning phase. In LEM(ED)2, the EA is the same, except that mutation uses the final set of intervals learned in the learning phase; mutations to parameter $i$ will have a high chance of choosing a value from the 'good' interval for $i$.

## 3 Experiments and Results

We tested LEM(ED)1 and LEM(ED)2 by running them on a set of functions that were used in the original LEM paper [13], augmented to provide a more thorough test. We compared with the underlying EA (i.e. LEM(ED) without the learning phase), and, to achieve an understanding in relation to state of the art performance, we chose to compare also with CMA-ES[7, 8].

In all cases the encoding was a vector of real-valued genes within the specified interval. The EA used binary tournament selection [9], elitism, uniform crossover [15] with probability 0.5, and uniform mutation with probability $1/(\textit{length of chromosome})$.

In LEM(ED) the *learning threshold* is 0.3, the instantiation method is implemented according to a probability; 80% new individuals are generated from the *good* interval, 20% are from the *bad* interval. In LEM(ED)2's evolution mode, the mutation is implemented with a normal distribution whose mean is centred over the good interval for the parameter in question, and with variance 1.

The main parameters for CMAES($\mu$,$\lambda$) are $\mu$, number of parent individuals, $\lambda$, the number of offspring, and the initial standard deviations $\sigma$ Here, we implement CMAES(50, 100) and *sigma* is set to one third of the range of each variable. Finally, EA and LEM(ED) population size was 100.

For completeness, details of the test functions used are:

a) Rastrigin's function:

$$f(x_1, \ldots, x_n) = 3.0n + \sum_{i=1}^{n} (x_i^2 - 3.0\cos(2\pi x_i)) \tag{1}$$

where $n = 30$ and $-5.12 \leq x_i \leq 5.12$.

b) Griewangk's function:

$$f(x_1, \ldots, x_n) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{2}$$

where $n = 30$ and $-600 \leq x_i \leq 600$.

c) Rosenbrock's function:

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2) \tag{3}$$

where $n = 30$ and $-2.048 \leq x_i \leq 2.048$.

d) Ackley's function:

$$f(x_1, \ldots, x_n) = 20 + e - 20exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)\right) \tag{4}$$

where $n = 30$ and $-30 \leq x_i \leq 30$.

e) de Jong's function 3:

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} integer(x_i) \tag{5}$$

where $n = 30$ and $-5.12 \leq x_i \leq 5.12$.

f) de Jong's function 4:

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} ix_i^4 + Gauss(0, 1) \tag{6}$$

where $n = 30$ and $-1.28 \leq x_i \leq 1.28$.

g) Schwefel's function:

$$f(x_1, \ldots, x_n) = 418.9829n + \sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right) \tag{7}$$

where $n = 30$ and $-500 \leq x_i \leq 500$.

Table 1 summarises the results of 100 runs of each algorithm on each function, with means and standard deviations recorded at 10, 20, 50 and 100 generations (multiply by 100 for number of fitness evaluations). Table 1 also provides information about the mean generation number at which learning was automatically terminated in the LEM(ED) variants. In interpreting the results, note that de Jong function 3 is a maximisation problem, and all others are minimization.

In the below, statements of significance are based on randomisation tests ([6]), and made only when confidence was above 99%. Inspection of standard deviations also clearly supports the statements made. With only one exception (LEM(ED)2 on Schwefel at 50 and 100 generations) the LEM(ED) variants are always significantly superior to the underlying EA. To some extent, the underlying EA can be seen as a 'straw man', and it is used here only as a baseline, with improvement to be expected. However, since LEM(ED)2 biasses itself strongly to learned intervals, it is quite possible that some functions can lead to it being deceived and misled. It is interesting and promising that this usually does not seem to happen, however LEM(ED)2's early fast progress on the Schwefel function clearly leads it in the wrong direction. This situation is the same for CMAE-ES, which is always beaten by both the EA and LEM(ED)1 on Schwefel, and beaten also be LEM(ED)2 until the 100-generations point.

Notwithstanding the case of the Schwefel function, the CMA-ES comparisons provide a much more exacting test for LEM(ED).With the exception of de Jong's function 3, one of the LEM(ED) variants is always superior to CMA-ES (and the GA) at the 1,000 and 2,000 evaluation points. At the 5,000 evaluations point, a LEM(ED) variant outperforms CMA-ES on de Jong's function 4, Rastrigin, Griewangk and Schwefel, and at 10,000 evaluations this reduces to Rastrigin and Schwefel, with ties for the de Jong functions. In general, CMA-ES, which is itself a sophisticated hybrid of learning and evolution, overtakes LEM(ED) (and the vast majority of other algorithms) as we consume more function evaluations.

**Table 1.** Summarising 100 indpendent runs of each algorithm on each problem: means and standard deviations after 10, 20, 50 and 100 generations (1,000, 2,000, 5,000 and 10,000 evaluations); in the first column of the final (100 generations) section, the number in brackets after the function name indicates the mean generation number at which learning terminated in the two LEM(ED) variants.

| Functions | EA | LEM(ED)1 | LEM(ED)2 | CMAES |
|---|---|---|---|---|
| 10 generations | | | | |
| Dejong3 | 84.1(0.89) | 135.4(0.62) | 136.1(0.01) | **150**(0) |
| Dejong4 | 26.7(1.3) | **1.68** (0.18) | 1.96 (0.05) | 50.22 (8.2) |
| Rastrigin | 148.52(1.1) | 92.91(1.8) | **91.98** (0.16) | 196.1(0.92) |
| Griewangk | 243.5(0.25) | **64.21** (0.89) | 64.79 (1.4) | 256.8 (2.9) |
| Rosenbrock | 2105.4 (28) | **397.6**(5.4) | 398.3 (11) | 3663.7 (150) |
| Ackley | 18.46 (0.043) | **13.67**(0.081) | 13.67(0.082) | 20.14(0.021) |
| Schwefel | 6733.6 (36) | 5542.73 (82) | **5196.01** (24) | 8400.76 (510) |
| 20 generations | | | | |
| Dejong3 | 111.08 (1.2) | 139.76(0.73) | 142.72 (0.3) | **150** (0) |
| Dejong4 | 8.732 (0.26) | **-0.910** (0.052)) | -0.820 (0.030) | 3.657(0.061) |
| Rastrigin | 98.77(1) | 58.31(0.21) | **56.77** (1.9) | 107.45 (2.6) |
| Griewangk | 116.19(1.5) | **7.61** (0.058) | 7.93(0.25) | 187.45(6.5) |
| Rosenbrock | 942.47 (12) | 112.13(5.1) | **110.096**(1.8) | 664.91 (32) |
| Ackley | 16.068(0.04) | 6.94 (0.17) | **6.90** (0.062) | 16.48 (0.049) |
| Schwefel | 4607.46 (87) | **3686.41** (9.3) | 3693.12 (8.0) | 8016.14 (450) |
| 50 generations | | | | |
| Dejong3 | 134.76(0.13) | 146.5 (0.087) | 149.93 (0.012) | **150**(0) |
| Dejong4 | 0.298(0.19) | -2.176 (0.041) | **-2.192**(0.068) | 0.0012(0.00016) |
| Rastrigin | 50.9(1.1) | 18.13(0.53) | **14.53** (0.63) | 62.01 (0.5) |
| Griewangk | 36.70(3.1) | 2.31(0.15) | **2.58** (0.14) | 3.399 (0.16) |
| 4 Rosenbrock | 331.65 (8.6) | 81.79 (3.9) | 79.97 (0.95) | **40.1**(0.71) |
| Ackley | 11.63(0.13) | 3.57(0.11) | 3.49(0.14) | **3.054** (0.033) |
| Schwefel | 2205.78 (74) | **1795.15** (83) | 3258.36 (3.7) | 6637.8 (820) |
| 100 generations | | | | |
| Dejong3 (5) | 145.41 (0.42) | 149.46(0.08) | **150** (0) | **150** (0) |
| Dejong4 (11) | **-1.19 (0.054)** | -2.64(0.1) | **-2.71**(0.054) | 1.9e-5(2e-6) |
| Rastrigin (40) | 25.29 (0.84) | 9.13 (0.07) | **4.76** (0.09) | 36.09 (0.22) |
| Griewangk (58) | 11.94(0.3) | 1.39 (0.005) | 1.98(0.11) | **0.38** (0.079) |
| Rosenbrock (47) | 178.3 (6.4) | 55.39 (1.9) | 47.7 (0.25) | **28.01** (0.014) |
| Ackley (50) | 8.061 (0.05) | 2.43 (0.16) | 0.701(0.069) | **0.1694** (0.00074) |
| Schwefel (2) | 1008.87 (42) | **827.694** (12) | 2871.39 (3.2) | 2621.92 (1200) |

Regarding the LEM(ED) design tested here, this is not surprising since our LEM(ED) variants use a single learning phase followed by an EA, while CMA-ES is continually learning and adapting. LEM(ED)'s performance in the 1,000–5,000 evaluations regime is neverthess encouraging, and there may be considerable value in more sophisticated adaptive versions.

## 4 Concluding Discussion

Our choice of function suite follows those used in the original LEM publications. However, such suites are now uperseded by those described in the CEC 2005 Challenge [21], which emphasises non-separability and other measures that are likely to make functions difficult. Since most of the functions tested herein are, however, separable, the criticism can be made that the findings may well not generalise to nonseparable functions. However, following preliminary and ongoing work we can confirm that the LEM(ED) variants here show entirely similar relative (to basic GA and to CMA-ES) performance properties as found here, and this will be the topic of further dissemination.

So, on the functions studied herein, we find that LEM(ED), a method which incorporates a simple entropy-based discretization method in the initial part of a EA run, clearly outperforms its EA component alone, and also generally outperforms CMA-ES during the initial several-thousand fitness evaluations. Also, though we omit details here, LEM(ED) outperforms LEM(KNN) [**?**]. This adds to evidence that even straightforward learning mechanisms provide considerable benefit to an EA, especially for accelerating the search.

Lines of further work that seem warranted include testing on a more accepted set of optimization challenge functions, and investigating repeated phases of ED-based learning (rather than a single phase at the beginning). Our investigations so far have focussed on tightly coupling ED and instantiation, which (as we find in preliminary experiments) is best limited, rather than continued throughout the run, otherwise the learned intervals can be deceived and results suffer. However we are yet to investigate (which would be highly suited to the LEM framework) the interleaving of further ED/instantiation phases with phases of several generations of evolution. Meanwhile, the information inherent in the learned interals could be used more creatively in later phases, in various ways. Also, a more sophisticated termination criterion for the ED phase would be beneficial, since we note that the 'better' sets of intervals are often those learned a handful of generations before the cessation of fitness improvement.

More generally, more study seems warranted in the area of learning/evolution combinations (both in terms of LEM-framework instantiations, and also in terms of organising the knowledge in this important area that is currently spread widely in the literature).

## References

1. Hussain, F., Liu, H., Tan,C.L., Dash, M.: Discretization: An Enabling Technique. Data Mining and Knowledge Discovery, v.6 n.4, p.393-423, October (2002).

2. Quinlan, J.R. Induction of decision trees. Machine Learning, 1:81 -106, (1986)
3. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, California (1993)
4. Sheri, G and Corne, D. : The Simplest Evolution/Learning Hybrid: LEM with KNN. In Proc. IEEE CEC 2008, pp. 3244-3251,(2008)
5. Bengoextea, E., Miquelez, M., Larranga, P., Lozano, J.A.: Experimental results in function optimization with EDAs in Continuous Domain, in [12] (2002)
6. Edgington, E.S : Randomization tests, 3rd ed. New York: Marcel-Dekker (1995)
7. Auger, A., Hansen, N.: A Restart CMA Evolution Strategy With Increasing Population Size. In Proc. IEEE CEC 2005, pp.1769-1776 (2005)
8. Hansen, N.: The CMA Evolution Strategy: A Comparing Review. In Lozano, J.A. et al. (eds.): Towards a new evolutionary computation. Advances in estimation of distribution algorithms. pp. 75-102, Springer (2006)
9. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley (1989)
10. Jourdan,L., Corne, D., Savic, D., Walters, G.: Hybridising rule induction and multiobjective evolutionary search for optimizing water distribution systems, in Proc of the 4th Hybrid Intelligent Systems conference, published in 2005 by IEEE Computer Society Press. pp. 434-439, ISBN 0-7695-1916-4 (2005)
11. Kaufmann,K., Michalski, R.S. :Learning from inconsistent and noisy data, the AQ18 approach, 11th Int'l Symp. on Foundations of Intelligent Systems (1999)
12. Larranaga, P., Lozano, J.A. (eds) : Stimulation of Distribution Algorithms: A New Tool for Evolutionary Computation, Kluwer Academic Publishers (2002)
13. Michalski, R.S.: LEARNABLE EVOLUTION MODEL Evolutionary Processes Guided by Machine Learning, Machine Learning, Vol. 38, pp. 9–40, (2000)
14. Pena, J.M., Robles, V., Larranaga, P., Herves, V., Rosales, F., Perez, M.S. : GA-EDA: Hybrid Evolutionary Algorithm using Genetic and Estimation of Distribution Algorithms, in Orchard, Yang, Ali (eds.) Innovations in Applied Intelligence: 17th Intel Conf. on AI and Expert Systems, Springer LNAI 3029 (2004)
15. Syswerda, G. : Uniform Crossover in Genetic Algorithms, Proc. of 3rd International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc (1989)
16. Wnek, J., Kaufmann, K., Bloedorn, E., Michalski R.S.:Inductive Learning System AQ15c: the method and user's guide. Reports of the Machine Learning and Inference Laboratory, MLI95-4, George Mason University,Fairfax, VA, USA (1995)
17. Wojtusiak, J. Michalski, R.S. : The LEM3 System for Non-Darwinian Evolutionary Computation and Its Application to Complex Function Optimization, Reports of the Machine Learning and Inference Laboratory, MLI 05-2, George Mason University, Fairfax, VA, USA (2005)
18. Wojtusiak, J., Michalski, R. S. : The LEM3 implementation of learnable evolution model and its testing on complex function optimization problems, in Proc. GECCO 2006 (2006)
19. Zhang, Q., Sun, J., Tsang, E., Ford, J.: Hybrid estimation of distribution algorithm for global optimisation, Engineering Computations 21(1): 91–107 (2003)
20. Zhang, Q., Sun, J., Tsang, E., Ford, J.: Estimation of distribution algorithm with 2-opt Local Search for the Quadratic Assignment Problem, in Lozana, Larranaga, Inza and Bengoetxea (eds) Towards a new evolutionary computation: advances in estimation of distribution algorithms (2006)
21. Suganthan, P.N., Hansen, N., Liang, J.J., Deb, K., Chen, Y.-P., Auger, A., Tiwari, A.: Problem definitions and evaluation criteria for the CEC 2005 Special Session on Real Parameter Optimization. Technical Report 2005005, Nanyang Technological University, Singapore. (2005)