A New Estimation of Distribution Algorithm based on Learning Automata

R. Rastegar
Computer Engineering and IT Department
Amirkabir University of Technology, Tehran, Iran
rrastegar@ce.aut.ac.ir

M. R. Meybodi
Computer Engineering and IT Department
Amirkabir University of Technology, Tehran, Iran
meybodi@ce.aut.ac.ir

Abstract. In this paper we introduce an estimation of distribution algorithm based on a team of learning automata. The proposed algorithm is a model based search optimization method that uses a team of learning automata as a probabilistic model of high quality solutions seen in the search process. Simulation results show that the proposed algorithm is a good candidate for solving optimization problems.

1 Introduction

The necessity to solve NP-complete problems, for which the existence of efficient exact algorithms is highly unlikely, has led to a wide range of heuristic algorithms that implement some sort of search in the solution space. One of these algorithms is genetic algorithm (GA) that is a class of optimization algorithms motivated from the theory of natural selection and genetic recombination. It tries to find better solutions by selection and recombination of promising solutions. It works well in wide verities of problem domains. The poor behavior of genetic algorithm in some problems, in which the designed operators of crossover and mutation do not guarantee that the building block hypothesis is preserved, has led to the development of other type of algorithms [6]. The Probabilistic Model Building Genetic Algorithms (PMBGAs) or Estimation of Distribution Algorithms (EDAs) is a class of algorithms that are recently developed to preserve the building blocks. The principle concept in this new technique is to prevent disruption of partial solutions contained in a chromosome by giving them high probability of being presented in the child chromosome. The EDAs are classified into three classes based on the interdependencies between variables in chromosomes; no dependency model, bivariate dependencies model, and multiple dependencies model [5] [6][11]. Instances of EDAs include Population-based Incremental Learning (PBIL) [1], bit-based simulated crossover (BSC) [15], Univariate Marginal Distribution Algorithm (UMDA) [8], Compact Genetic Algorithm (cGA) [4] for no dependency model, Mutual Information Maximization for Input Clustering (MIMIC) [3], Combining Optimizer with Mutual Information Trees (COMIT) [2] for bivariate dependencies model, and Factorized Distribution Algorithm (FDA) [7], Bayesian Optimization Algorithm (BOA) [12][11] for multiple dependencies model, to name a few. Although all no dependency model algorithms have low efficiency in solving difficult problems, due to their simplicity in terms of memory usage and computational complexity and with respect to the fact that the computational complexity of bivariate dependencies model and multiple dependencies model is high, proposing new algorithms for this model is an important issue.

Learning Automata (LA) are general-purpose stochastic optimization tools, which have been developed as a model for learning systems. They are typically used as the basis of learning systems, which through interactions with a stochastic unknown environment learn the optimal action for that environment. The learning automaton tries to determine, iteratively, the optimal action to apply to environment from a finite number of actions that are available to it. The environment returns a reinforcement signal that shows the relative quality of action of the learning automaton. This signal is given to learning automaton and learning automaton adjusts itself by a learning algorithm [9].

In this paper we propose a learning automata-based search method, called Learning Automata based Estimation of Distribution Algorithm (LAEDA), as an estimation of distribution algorithm for a class of EDAs in which there is no dependency between variables. The LAEDA is a simple EDA that ignores all the variables interactions. Since the proposed algorithm belongs to no dependency model, it will be compared with the PBIL and the UMDA that are among the most famous algorithms of the class of no dependency model.

The rest of paper is organized as follows. Section 2 briefly presents some EDAs. Learning automata are described in section 3. Section 4 demonstrates the proposed algorithm. Simulation results are given in section 5. Finally, section 6 concludes.

2 Estimation of Distribution Algorithms

In EDAs, the problem specific interactions among the variables of chromosomes are taken into consideration. In the genetic algorithm the interactions are kept implicitly in mind whereas in EDAs the interrelations are expressed explicitly through the joint probability distribution associated with the chromosomes selected at each generation. The probability distribution is calculated from a set of selected chromosomes of previous generation. Then sampling this probability distribution generates

0-7803-9363-5/05/$20.00 ©2005 IEEE.

children. Neither crossover nor mutation has been applied in EDAs. But the estimation of the joint probability distribution associated with the set containing the selected chromosomes is not an easy task. The easiest way to calculate the joint probability distribution is to consider all the variables in a problem as univariate. In all the works done based on this approach, it is assumed that *n*-dimensional joint probability distribution factorizes as a product of *n* univariate and independent probability distributions. In the reminder of this section we briefly describe the algorithms of no-dependency model.

Syswerda [15] has introduced an operator called bit-based simulated crossover (BSC) that uses the statistics in the GA's population to generate offspring. The BSC does a weighted average of alleles of chromosomes along each bit position. By using the fitnesses of the chromosomes in this computation, BSC integrates the selection and crossover operators into a single step.

The Population based incremental learning (PBIL) [1] adapts the vector of probabilities by mean of an updating rule inspired by the so called Hebbian rule used in neural network. In each generation, the PBIL adapts the *n*-dimensional vector of probabilities bringing near each component, by means of a learning rate, to the corresponding component of a set of best chromosomes found in that generation. When learning rate is 1, the PBIL is equivalent to the UMDA. In the UMDA [8], the joint probability distribution is factorized as a product of independent univariate marginal distribution, which is estimated from marginal frequencies. There is the theoretical evidence that the UMDA approximates the behavior of the Simple Genetic Algorithm (SGA) with uniform crossover [14].

Harik has presented an algorithm in [4] that called compact genetic algorithm (cGA). The algorithm initializes a probability vector whose components follow Bernoulli distributions with parameter 0.5, and then two chromosomes are generated randomly by using this probability vector and rank them by evaluating their fitnesses. Then the probability vector is updated towards the best one. This process of adaptation continues until the probability vector converges.

3 Learning Automata

Learning Automata are adaptive decision-making devices operating on unknown random environments. A Learning Automaton has a finite set of actions and each action has a certain probability (unknown for the automaton) of getting rewarded by the environment of the automaton. The aim is to learn to choose the optimal action (i.e. the action with the highest probability of being rewarded) through repeated interaction on the system. If the learning algorithm is chosen properly, then the iterative process of interacting on the environment can be made to result in selection of the optimal action. Figure 1 illustrates how a learning automaton works in feedback connection with a random environment. Learning Automata can be classified into two main families: fixed structure learning automata and variable structure learning automata (VSLA) [9]. In the following, the variable structure learning automata is described.

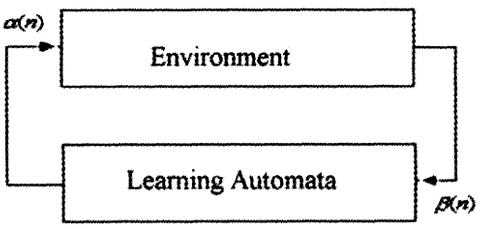

Figure 1. The interaction between learning automata and environment

A VSLA is a quintuple $<\alpha,\beta,p,T(\alpha,\beta,p)>$, where α, β, p are an action set with s actions, an environmental response set and the probability set p containing s probabilities, each being the probability of performing every action in the current internal automaton state, respectively. The function of T is the reinforcement algorithm, which modifies the action probability vector p with respect to the performed action and received response. Let a VSLA operate in an environment with $\beta=\{0,1\}$. Let $t \in N$ be the set of nonnegative integers. A general linear schema for updating action probabilities can be represented as follows. Let action i be performed at instance (iteration) t.
If $\beta(t)=0$ (reward),

$$p_i(t+1) = p_i(t) + a[1 - p_i(t)]$$
$$p_{j\neq i}(t+1) = (1-a)p_j(t)$$

If $\beta(t)=1$ (penalty),

$$p_i(t+1) = (1-b)p_i(t)$$
$$p_{j\neq i}(t+1) = (b/s-1) + (1-b)p_j(t)$$

Where a and b are the reward and penalty parameters. When $a=b$, automaton is called L_{RP}. If $b=0$ and $0<b<<a<1$, the automaton is called L_{RI} and $L_{R\varepsilon P}$, respectively.

Another learning algorithm used in this paper is the pursuit learning algorithm. The L_{RI} algorithm moves the action probability in the direction of the most recently rewarded action, whereas the pursuit algorithm moves it toward the action that possesses the highest estimate of reward [10][13]. The schema of updating action probabilities is as follows. Let action i be performed.
If $\beta(t)=0$ (reward),

$$p_{j\neq i}(t+1) = (1-a)p_j(t)$$
$$p_k(t+1) = 1 - \sum_{j\neq i} p_j(t+1)$$

If $\beta(t)=1$(penalty),

$$p_i(t+1) = p_i(t)$$

For either penalty or reward, the running estimates are subsequently updated as follows:

$$W_i(t+1) = W_i(t) + (1 - \beta(t))$$
$$Z_i(t+1) = Z_i(t) + 1$$
$$d_i(t+1) = W_i(t+1) / Z_i(t+1)$$

where p_k is the action probability that has the highest running estimate d_i of being rewarded, $W_i(t)$ is the number of times the ith action has been rewarded up to t; and $Z_i(t)$ is the number of times the ith action has been selected up to t.

4 The Learning Automata based Estimation of Distribution Algorithm

Let $\mathbf{x}=(x_1,\ldots,x_n)$ denoted a vector, that x_i belongs to $\Lambda_i=\{0,\ldots,m_i\}$. Λ_i represents a set of m_i+1 symbols, representing different alleles in the chromosome. We use the X_i to denote a variable and x_i to denote an assignment. We consider the optimization problem $\mathbf{x}_{opt}=argmax\ f(\mathbf{x})$ where $f:\boldsymbol{X}\rightarrow\boldsymbol{R}$.

In the learning automata based estimation of distribution algorithm (LAEDA), similar to other evolutionary algorithms, the parameters of the search space are encoded in the form of chromosomes. A set of n learning automata is used in LAEDA to model the probability of sampling chromosomes in the population. A learning automaton with m_i+1 actions is associated to each variable i with action set $\Lambda_i=\{0,\ldots,m_i\}$. When we consider a problem in a binary search space, Λ_i becomes $\{0,1\}$ for all i, i.e. each learning automaton has two actions. In each instance t, each learning automaton i selects N actions, $\alpha_{i1}\ldots\ \alpha_{iN}$, using its probability vector. Selection of any of these N actions does not change the action probability vector. That is the same action probability vector is used to select all N actions. Each action selected by an automaton becomes an allele of a chromosome. A total of $N\times n$ alleles will be generated by all learning automata. Now a new population of N chromosomes is formed as follows. The jth chromosome, X_j, is constructed by concatenation of actions $\alpha_{1j}\ldots\ \alpha_{nj}$. That is $X_j=(\alpha_{1j}\ldots\ \alpha_{nj})$.

According to a selection strategy, M chromosomes from the newly formed population are selected. Using these M chromosomes and a signal generation mechanism which will be explained later a reinforcement signal vector $\beta=(\beta_1\ldots\beta_n)$ is produced. Signal β_i will be used by automaton i to reward or penalize one of its actions. The selection of the action that is to be rewarded or to be penalized is done as follows. Let $k\in\Lambda_i$ and X_{ij} be ith variable of the jth chromosome, Define,

$$\alpha r_i = \arg\max_k \left(\sum_{j\in Sel_t} \varphi(X_{ij}=k)\right),$$

and

$$\alpha p_i = \arg\max_k \left(\sum_{j\in UnSel_t} \varphi(X_{ij}=k)\right),$$

where Sel_t is the set of M chromosomes that are selected from the current population using selection methods such as truncation selection schema, and the set $UnSel_t$ is the set of chromosomes not selected during the selection process. The Boolean function $\varphi(exp)$ returns 1 if exp is true and returns 0 otherwise. αr_i and αp_i are the actions of automaton i that will be rewarded or penalized. Note that αr_i and αp_i may represent the same action that is $\alpha r_i=\alpha p_i$. To generate the reinforcement signal vector, β, a random number, ρ, is generated uniformly from interval [0,1] and is compared with a predefined parameter $1\geq\upsilon\geq0$. If $\rho>\upsilon$, a positive reinforcement signal (i.e. $\beta_i=0$ for all i) is generated and input to automaton i which as a result, action αr_i of learning automaton i is rewarded according to the learning algorithm. If $\rho\leq\upsilon$, a negative reinforcement signal (i.e. $\beta_i=1$ for all i) is generated and input to automaton i which as a result action αp_i of learning automaton i is penalized according to the learning algorithm. The process of population generation, selection, and updating the probabilities vectors of all learning automata is repeated until a termination condition is satisfied which at this point the best chromosome of the last population will the solution to the problem to be solved. Selection of υ is very important issue in the design of the algorithm. The value of υ is selected by heuristics and according to a learning algorithm. If the learning mechanism is based on learning only from Sel_t, υ is set to 0 otherwise it is set to $1>>\upsilon>0$. For a learning algorithm to work in this context the value of υ must be chosen properly. In the simulations conducted the value of υ for L_{RI}, Pursuit and L_{RP}, are 0, 0, and 0.05 respectively

Remark: If $\rho>\upsilon$, then the learning automata update their action probabilities in such a manner that the probability that search process moves toward the area in the search space with high quality solution (Sel_t) increases that is the search process learns from positive past experiences. Whereas if $\rho\leq\upsilon$ the learning automata update their action probabilities in such a manner that the probability of searching the low quality search area in the search space decreases, that is the search process learns from its negative past experiences.

In order to have an effective algorithm, the designer of the algorithm must be careful about determining a suitable genome representation, fitness function for the problem at hand, the parameters of LAEDA such as the number of chromosomes (population size), the selection strategy, the signal generating mechanism and the type of the learning automata.

5 Simulations and Results

In order to show the performance of the proposed algorithm, the algorithm is tested on 5 different problems: One Max, Subset Sum, Checker Board, Equal Product, Knapsack 0/1, and TSP problems and then compared with the simple genetic algorithm (SGA), the UMDA, and the PBIL. The test problems are briefly explained below.

OneMax: This is a linear problem that can be written mathematically as,

$$F_{OneMax}(x) = \sum_{i=1}^{n} x_i$$

SubsetSum: It is a problem of finding what subset of a set of integer A has a given sum c.

CheckerBoard: This problem was used by Baluja to evaluate the performance of the PBIL algorithm [1]. In this problem a $s \times s$ grid is given. Each point of the grid can take two values 1 and 0. The goal of this problem is to create a checkerboard pattern of 0's and 1's on the grid. Each location with a value of 1 should be surrounded in all four directions by a value 0, and vise versa. Only the four primary directions are considered in the evaluation. The evaluation is measured by counting the number of correct surrounding bits for the present value of each bit position for a $(s-2)\times(s-2)$ grid. In this manner, the corners are not included in the evaluation. The chromosomes of this problem have dimension $n=s^2$. If we consider the grid as a matrix $C=[c_{ij}]_{i,j=1,\ldots,s}$ and interpret $\delta(a,b)$ as the kronecher's delta function, the checkerboard function can be written as follows,

$$F_{Checker\,Board}(C) = 4(s-2)^2 - \sum_{i=2}^{n-1}\sum_{j=2}^{n-1} \delta(c_{ij}, c_{i-1\,j}) + \delta(c_{ij}, c_{i+1\,j}) + \delta(c_{ij}, c_{i\,j-1}) + \delta(c_{ij}, c_{i\,j+1})$$

EqualProducts: Given a set of n real numbers $\{b_1,\ldots,b_n\}$ that a subset of them is chosen. The objective is to minimize the difference between the products of the selected and unselected numbers.

$$F_{Equal\,Products}(x) = \left| \prod_{i=1}^{n} x_i b_i - \prod_{i=1}^{n} (1-x_i) b_i \right|$$

For the simulations $\{b_1,\ldots,b_n\}$ is generated by sampling from a uniform distribution in the interval [0,4].

Knapsack 0/1: In the knapsack problem, there is a single bin of limited capacity, and n elements of varying size and values. The problem is to select the elements that will yield the greatest summed value without exceeding the capacity of the bin. The evaluation of the quality of the solution is measured in two ways; if the solution selects too many elements, such that the sum of the sizes of the elements is too large, the solution is judged by how much this sum exceeds the capacity of the bin. If the sum of the sizes of the elements is within the capacity of the bin, the sum of the values of the selected elements is used as the evaluations.

$$f_{Knapsack}(x) = \sum_{i=1}^{n} (x_i v_i) + \eta\left(c - \sum_{i=1}^{n} (x_i c_i)\right) u\left(\sum_{i=1}^{n} (x_i c_i) - c\right)$$

where η is a parameter that determines the penalty coefficient that is given to infeasible solutions and $u(.)$ is step function. η is 0.1 in our simulations. The values and sizes of the elements are selected uniformly between intervals of [0,30].

TSP: Given L cities, the object is to find a minimum length tour that visits each city exactly ones. The encoding used in this study requires a bit string of size $L\log L$ bits. Each city is assigned a sub string of length $\log L$ that is interpreted as an integer. The city with the lowest integer value comes first in the tour, the city with the second lowest comes second, etc.

$$f_{TSP}(x) = d(city_L, city_1) + \sum_{i=1}^{L-1} d(city_i, city_{i+1})$$

$$city_i = \sum_{j=0}^{\log_2^L - 1} 2^j x_{(i-1)L+(\log_2^L - j)}$$

Where $d(city_i, city_j)$ is Euclidian distance between two cities $city_i$ and $city_j$.

The population size is the same in all considered algorithms and set up depending on the complexity of the problem. The size of the population is 10, 10, 20, 10, 100, and 100 for OneMax, Subset Sum, Checkerboard, EqualProducts, Knapsack 0/1, and TSP, respectively. We use truncation selection schema to select the parent population in all algorithms. The number of the selected chromosomes is set up to half of the size of the population. Two termination conditions are taken into account. Firstly, The algorithm stops when a fixed number of function evaluations (Max. Evaluation) are performed. Table 1 shows the characteristics of the test bed problems; 'No. Variable', 'Max', Evaluation', 'Type' and 'optimum' refer to the length of the chromosome, the predetermined maximum number of function evaluations allowed, the type of the problem which is either a maximization problem or a minimization problem, and the optimal solution for the problem, respectively. For the simple genetic algorithm, uniform crossover with exchange probability 0.5 is used. Mutation is not used and crossover is applied all iterations. The best chromosome of the previous population is always brought into the new population and the remaining $N-1$ chromosomes of the new population are generated. Comparisons between considered algorithms are in terms of solution quality, and the number of function evaluations taken for finding the best solution. The proposed Algorithm is tested for different learning algorithms: L_{RI}, L_{RP}, and Pursuit. In all the experimentations we select the value of the reinforcement signal from $\{0,1\}$. For all the experiments, the learning rate for PBIL is set to 0.01 and the both reward and penalty parameters in LAEDA are set to 0.01.

For the sake of convenience in presentation, we use LA(*automata*)EDA to refer to the LAEDA algorithm when it uses Learning automata *automata*. Table 2 and 3 report the result of simulations for all algorithms. In these tables 'Function evaluation' and 'Objective Value' are best solution founded in the last population and the number of function evaluations taken for finding the best solution. The results reported are average over 20 runs. By careful inspection of the results reported in Tables 2 and 3, it is found that LA(L_{RP})EDA obtained the better solution for all of the test bed problems except CheckerBoard problem. For all the problems except problem OneMax, algorithm LA(Pursuit)EDA obtains the worst results. For CheckerBoard problem, simple genetic algorithm obtained the best solution. LA(L_{RI})EDA and LA(Pursuit)EDA only get the optimal solution for OneMax problems. For see

more results, reader can refer to [16]. The results show that LA(L_{RP})EDA can be a good candidate for solving optimization problems.

6 Conclusions

This paper has introduced a new estimation of distribution algorithm based on learning automata. The proposed algorithm is a model based search optimization algorithm that uses learning automata as a tool to effectively search the search space. In order to show the performance of the proposed algorithm, it is tested on number of different problems and then compared with the simple genetic algorithm (SGA), UMDA, and PBIL. Simulation results showed the effectiveness of the proposed algorithm in solving the optimization problems. LAEDA has some advantages. 1- this algorithm can be easily extended to continuous and non-binary search spaces. 2- many learning automata reported in literatures, which could be used in LAEDA with respect to problem at hand.

References

[1] Baluja, S., Caruana, R., "Removing The Genetics from The Standard Genetic Algorithm", In Proceedings of ICML'95, PP. 38–46, Morgan Kaufmann Publishers, Palo Alto, CA, 1995.

[2] Baluja, S., and Davies, S., "Using Optimal Dependency Trees for Combinatorial Optimi-zation: Learning the Structure of Search Space", Technical Report CMU-CS-97-107, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1997.

[3] De Bonet, J. S., Isbell, C. L., and Viola, P., "MIMIC: Finding Optima by Estimating Probability Densities", In Proceedings of NIPS'97, PP. 424–431, MIT Press, Cambridge, MA, 1997.

[4] Harik, G. R., Lobo, F. G., and Goldberg, D. E., "The Compact Genetic Algorithm", IEEE Transaction on Evolutionary Computing, Vol. 3, No. 4, PP. 287–297, 1999.

[5] Larranaga, P., Etxeberria, R., Lozano, J. A., and Pena, J. M., "Optimization by Learning and Simulation of Bayesian and Gaussian Networks", Technical Report EHU-KZAA-IK-4/99, Department of Computer Science and Artificial Intelligence, University of Basque Country, December 1999.

[6] Larrañaga, P., and Lozano, J. A., Estimation of Distribution Algorithms. A New tools for Evolutionary Computation, Kluwer Academic Publishers, 2001.

[7] Mühlenbein, H., and Mahnig, T., "The Factorized Distribution Algorithm for additively decomposed functions", Proceedings of the 1999 Congress on Evolutionary Computation, IEEE press, PP. 752-759, 1999.

[8] Mühlenbein, H., "The Equation for Response to Selection and Its Use for Prediction", Evolutionary Computation, Vol. 5, No. 3, PP. 303-346, 1998.

[9] Narendra, K. S., and Thathachar, M. A. L., *Learning Automata: An Introduction*, Printice-Hall Inc, 1989.

[10] Oommen, B. J., and Agache, M., "A Comparison of Continuous and Discredited Pursuit Learning Schemes", Technical Report, School of Computer Science, Carleton University, Canada, Ottawa, K1S 5B6, 1999.

[11] Pelikan, M., Goldberg D. E., and Cantz Paz, E., "BOA: the Bayesian Optimization Algorithm", Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, Morgan Kaufmann Publishers, pp. 525-532, 1999.

[12] Pelikan, M., Goldberg, D. E., and Cant-Paz, E., "Linkage Problem, Distribution Estimation and Bayesian Networks", Evolutionary Computation, Vol. 8, No. 3, PP. 311-340, 2000.

[13] Thathachar, M. A. L., and Sastry, P. S., "Estimator Algorithms for Learning Automata", Proceeding Platinum Jubilee Conferences on Systems and Signal Processing, Electrical Eng. Department, Indian Institute of Science, Bangalore, India, Dec. 1986.

[14] Mühlenbein, H., and Mahnig, T., "Evolutionary Algorithms: From Recombination to Search Distributions", Theoretical Aspects of Evolutionary Computing, Springer Publication, 2001.

[15] Syswerda, G., "Simulated Crossover in Genetic Algorithm", FOGA-2, Morgan Kaufmann Publisher s San Mateo, CA, PP. 232-255, 1992.

[16] Rastegar, R., and Meybodi, M. R., "LAEDA: A New Estimation of Distribution Algorithm", Technical Report, Computer Eng. Department, Amirkabir University, Tehran, Iran, 2004.

Table 1. Characteristics of the test bed problems

Problem	F_{OneMax}	$F_{SubsetSum}$	$F_{CheckerBoard}$	$F_{EqualProducts}$	F_{TSP}	$F_{Knapsack}$
No. Variable	128	128	100	50	128	100
Max. Evaluation	100000	100000	100000	300000	300000	300000
Type	Max.	Min.	Max.	Min.	Max.	Max.
Optimum	128	-	256	-	-	1147

Table 2. Results of simulations for LA(L_{RI})EDA, LA(Pursuit)EDA, and LA(L_{RP})EDA

Problem		*LA(L_{RI})EDA*	*LA(Pursuit)EDA*	*LA(L_{RP})EDA*
F_{OneMax}	Objective Value	128	128	128
	Function evaluation	3239	2670	5200
$F_{SubsetSum}$	Objective Value	0.00384	0.04835	0.0026
	Function evaluation	4050	380	7530
$F_{CheckerBoard}$	Objective Value	186	166	210
	Function evaluation	10000	500	44000
$F_{EqualProducts}$	Objective Value	1.16	2.97	0.843
	Function evaluation	17000	4000	22340
$F_{Knapsack}$	Objective Value	1098	910	1141
	Function evaluation	27300	8310	32850
F_{TSP}	Objective Value	1893	2324	1710
	Function evaluation	52800	7930	57200

Table 3. Results of simulations for UMDA, PBIL, and SGA.

Problem		***UMDA***	***PBIL***	***SGA***
F_{OneMax}	Objective Value	128	128	125.4
	Function evaluation	7350	4240	36210
$F_{SubsetSum}$	Objective Value	0.00332	0.00320	0.00344
	Function evaluation	6430	4050	12430
$F_{CheckerBoard}$	Objective Value	241	210	246
	Function evaluation	53000	13240	52000
$F_{EqualProducts}$	Objective Value	1.95	1.1	3.35
	Function evaluation	36320	15320	193430
$F_{Knapsack}$	Objective Value	1141	1125	1024
	Function evaluation	34120	27332	300000
F_{TSP}	Objective Value	1794	1926	1873
	Function evaluation	89740	52800	300000