2008

Improved and generalized learning strategies for dynamically fast and statistically robust evolutionary algorithms

Yogesh Dashora
University of Texas at Austin

Sanjeev Kumar
University of Bath

Nagesh Shukla
National Institute of Foundry and Forge Technology, nshukla@uow.edu.au

M K. Tiwari
Indian Institute Of Technology

Available online at www.sciencedirect.com
ScienceDirect
Engineering Applications of Artificial Intelligence 21 (2008) 525–547
Engineering Applications of ARTIFICIAL INTELLIGENCE
www.elsevier.com/locate/engappai

Improved and generalized learning strategies for dynamically fast and statistically robust evolutionary algorithms

Yogesh Dashora[a], Sanjeev Kumar[b], Nagesh Shukla[c], M.K. Tiwari[d,*]

[a]Mechanical Engineering Department, University of Texas at Austin, USA
[b]Department of Mechanical Engineering, University of Bath, UK
[c]Department of Manufacturing Engineering, National Institute of Foundry and Forge Technology, Ranchi, India
[d]Department of Industrial Engineering and Management, Indian Institute of Technology, Kharagpur 721302, India

Received 22 March 2007; accepted 27 June 2007
Available online 26 November 2007

Abstract

This paper characterizes general optimization problems into four categories based on the solution representation schemes, as they have been the key to the design of various evolutionary algorithms (EAs). Four EAs have been designed for different formulations with the aim of utilizing similar and generalized strategies for all of them. Several modifications to the existing EAs have been proposed and studied. First, a new tradeoff function-based mutation has been proposed that takes advantages of Cauchy, Gaussian, random as well as chaotic mutations. In addition, a generalized learning rule has also been proposed to ensure more thorough and explorative search. A theoretical analysis has been performed to establish the convergence of the learning rule. A theoretical study has also been performed in order to investigate the various aspects of the search strategy employed by the new tradeoff-based mutations. A more logical parameter tuning has been done by introducing the concept of orthogonal arrays in the EA experimentation. The use of noise-based tuning ensures the robust parameter tuning that enables the EAs to perform remarkably well in the further experimentations. The performance of the proposed EAs has been analyzed for different problems of varying complexities. The results prove the supremacy of the proposed EAs over other well-established strategies given in the literature.

Keywords: Evolutionary algorithms; Numerical optimization; Combinatorial optimization; Mutation step size; Chaos; Noise; Design of experiments; Parameter tuning

1. Introduction

Since its advent, evolutionary optimization has gone through various stages of evolution that have made it more robust and fast; robust enough to be applied to all spheres of optimization problems and fast enough to take decisions in micro-seconds (Dimopoulos and Zalzala, 2000; Nissen and Propach, 1998; Yao et al., 1999). In the present scenario, evolutionary algorithms (EAs) are not limited to the applications in artificial intelligence, and have been increasingly used in various dimensions to solve real world problems (Dimopoulos and Zalzala, 2000; Sinha et al., 2003). Although, a plethora of literature pertaining to the search strategies adopted by various EAs is available (Nissen and Propach, 1998; Rana et al., 1996; Kazarlis et al., 2001; Yao et al., 1999; Salomon, 1998; Choi and Oh, 2000; Yoon and Moon, 2002; Kim and Myung, 1997; Storn, 1999), most of them restrict themselves to solving a particular class of optimization problem and thus, fail to provide generalized strategies that can be robustly used for wide spectrum of optimization problems in science, business and engineering applications. In general, optimization problems can be classified into two groups—numerical optimization and combinatorial optimization (Tsai et al., 2004; Gen and Cheng, 1999). Yet another classification exists that is based on the representation schemes—binary string, floating point string and integer bit string representation (Rana et al., 1996; Kazarlis et al., 2001; Yao et al., 1999; Salomon, 1998; Choi and Oh, 2000;

*Corresponding author. Tel.: +91 651 2291116; fax: +91 651 2290860.
E-mail address: mkt09@hotmail.com (M.K. Tiwari).

0952-1976/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2007.06.005

Yoon and Moon, 2002; Kim and Myung, 1997; Storn, 1999; Zhang and Leung, 1999; Burke and Newall, 1999). In general, the first two representations are more suited and easy to implement on numerical optimization problems; whereas, the third representation enjoys solving the combinatorial optimization problems with a greater ease and efficiency (Nijssen and Bäck, 2003; Goldberg, 1989). Thus, to propose effective algorithms that would efficiently work in all the representations is a convoluted task. The schematic representation of various optimization problems along with their most suited representation classes are shown in Fig. 1.

All of the above-mentioned factors motivated the authors to propose robust and fast EAs with improved learning strategies to achieve the global optimization in all of these cases. Before introducing the proposed algorithms, the general formulation of a numerical optimization problem can be mathematically described as

$$\text{minimize } g(\mathbf{t}),\ \mathbf{t} = (t_1, t_2, \ldots, t_n) \in \mathbf{\Lambda}^n, \tag{1}$$

where $\mathbf{t} \in \mathbf{\Gamma} \cap \mathbf{X}, \mathbf{\Gamma}$ represents the 'feasible region' defined as

$$\mathbf{\Gamma} = \{\mathbf{t} \in \mathbf{\Lambda}^n | c_j(\mathbf{t}) \leqslant 0 \quad \forall j \in \{1, 2, \ldots, J\}\}, \tag{2}$$

where $c_j(\mathbf{t})$, $j \in \{1, 2, \ldots, J\}$ are the 'constraints' on the problem; the n-dimensional 'search space' is defined by $\mathbf{X} \subseteq \mathbf{\Lambda}^n$ and is constrained by the 'parametric constraints' given below:

$$\underline{t_i} \leqslant t_i \leqslant \bar{t}_i, \quad i \in \{1, 2, \ldots, n\} \tag{3}$$

where $\underline{t_i}$ and $\bar{t}_i$ represent the lower and upper limits of the variable t_i. Here, it is evident that the solution is represented as floating point string of type **A** (Fig. 1); however, it can also be represented in the binary string (type **B** of Fig. 1) by considering the representation of floating numbers into binary numbers.

For the second class of optimization problems, i.e. combinatorial optimization problems, the general formulation can be given as

$$\text{minimize } h(\mathbf{s}) \quad \mathbf{s} = (s_1, s_2 \ldots s_n) \in \mathbf{S}, \tag{4}$$

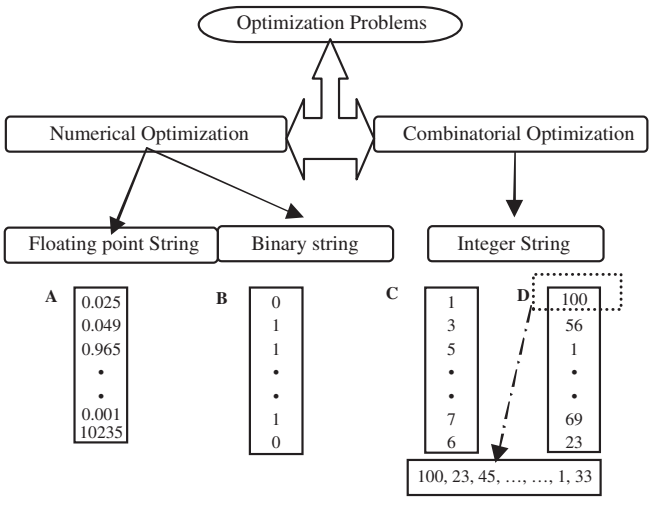

Fig. 1. Classification of optimization problems in a hierarchical manner.

where $\mathbf{S}$ is the set of integers; $\mathbf{s} \in \mathfrak{J} \cap \Re, \mathfrak{J}$ denotes the 'feasible range' of variables and is given by

$$\mathfrak{J} = \{\mathbf{s} \in \mathbf{S} | c'_j(\mathbf{s}) \leqslant 0 \quad \forall j \in \{1, 2, \ldots, J'\}\}, \tag{5}$$

where $c'_j(\mathbf{s})$ represents the 'constraints' of the problem; the 'search range' $\Re$ is defined by $\Re \subseteq \mathbf{S}$, which is constrained by the 'individual range' defined by

$$\underline{c'_i} \leqslant c'_i \leqslant \bar{c}'_i, \quad i \in \{1, 2, \ldots, n\}, \tag{6}$$

where $\underline{c_i}$ and $\bar{c}'_i$ are the lower and upper limits of the variable c'_j. Generally, this formulation is utilized for the type **D** representation. However, the representation type **C** is used in the combinatorial problems class mapped by traveling salesman problem (TSP). Its general formulation is described as

$$\text{minimize} f_{\text{TSP}} = \sum_{i=1}^{N} \sum_{j=1}^{N} \text{dist}_{ij} \Omega_{ij} \quad \forall i, j \in \{1, 2, \ldots, N\}, \text{ and } i \neq j, \tag{7}$$

where

$$\Omega_{ij} = \begin{cases} 1 & \text{if node } i \text{ and } j \text{ are selected in the tour,} \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

subject to the constraints c''_j, $j = \{1, 2, \ldots, n\}$, where N is the number of nodes; dist_{ij} is the distance between nodes i and j; Here, only minimization cases are considered as the problems pertaining to maximization of objective can also be mapped into minimization problem by making suitable changes in the formulation.

Owing to the existence of all these formulations, authors intend to present some improved learning rules suited to each formulation in order to eliminate the difficulty and dilemma of the practitioners and users of EAs while searching the best strategy for their problem. In addition to this, various testing phases based on design of experiments are introduced in order to achieve robust parameter control and tuning.

In the earlier studies, various modifications to the initial versions of EAs have been proposed to suit the varying complexity of the optimization problems. Some of the vital factors that affect the search strategy of the EAs are: (1) population representation, (2) information sharing among members of the population, (3) mutation of the current population, (4) generation scheme for new population, (5) learning from previous generations (Kazarlis et al., 2001; Yao et al., 1999; Salomon, 1998; Choi and Oh, 2000; Yoon and Moon, 2002; Kim and Myung, 1997; Storn, 1999; Zhang and Leung, 1999; Burke and Newall, 1999; Eiben et al., 1999; Harik et al., 1999; Michalewicz et al., 2000; Runarsson and Yao, 2000; François and Lavergne, 2001; Kazarlis et al., 2001; Ong and Keane, 2004). Even a small variation in any of these factors has a considerable effect on the EA performance.

The following discussion provides an insight to some of the recent developments in the field of evolutionary computation. In order to do away the slow convergence

of the classical evolutionary programming (CEP), Yao et al. (1999) introduced fast evolutionary programming (FEP) that utilized the mutation based on Cauchy function because of its higher probability of making longer jumps Runarsson and Yao (2000) introduced a novel approach to balance objective and penalty functions stochastically, i.e. 'stochastic ranking' and presented a fresh view on penalty function methods in terms of dominance of penalty and objective functions. Salomon (1998) presented a new hybrid approach, the evolutionary-gradient-search method for the problems pertaining to numerical optimization. A much broader collection of developments in evolutionary computation for manufacturing optimization can be found in Dimopoulos and Zalzala (2000). Zhang and Leung (1999) proposed an orthogonal genetic algorithm for multimedia cast routing that incorporated orthogonal design to crossover operation. Francois (1998) presented a mutation or selection evolutionary strategy (MOSES) to solve complex discrete optimization problems. It theoretically studied the relationship between convergence, parameters of the method, and the geometry of the optimization problem. An extensive empirical study on the synergy among multiple crossover operators has been provided in Yoon and Moon (2002).

Choi and Oh (2000) presented a new mutation rule for evolutionary programming (EP) that has been motivated from the back-propagation learning rule of the neural networks. A method of decomposing larger problems into smaller components, each of which is of a size that the EA can effectively handle, was proposed by Burke and Newall (1999) to solve timetabling problem. Eiben et al. (1999) provided a broader classification for parameter control mechanisms of EAs. They also provided a survey of various types of control that have been used by evolutionary computation community. A statistical method was described by François and Lavergne (2001), which was able to find good parameter settings for EAs. The method utilized a functional relationship between the algorithms performance and its parameter values. Yet another statistical approach can be found in Czarn et al. (2004).

Ahn and Ramakrishna (2003) described two elitism-based compact genetic algorithms in order to design efficient compact type GAs by treating them as the estimation of distributed algorithms (EDAs) to efficiently solve complex optimization problems. The compact GAs utilized the concept of successive learning in generations utilizing the probability strings for the solution representation. Various other learning approaches also persist in the literature. In general, learning algorithms are termed as learning automata approaches (Najim and Poznyak, 1994). In Agache and Oommen (2002), two new generalized pursuit algorithms were presented that falls in the category of fastest learning automata algorithms. Howell et al. (2002) proposed a hybrid genetic and learning automata approach that enjoys the merits of both the strategies. Various other relevant work like Papadimitriou (1994), Najim and Poznyak (1994) in the context of learning algorithms need to be mentioned here.

Many works have been reported in literature in order to map the system chaos and noise within the working dimensions of EAs (Nissen and Propach, 1998; Rana et al., 1996; Caponetto et al., 2003). Caponetto et al. (2003) introduced chaotic sequences instead of random sequences during all the phases of random evolution process. The approach was based on the spread spectrum characteristic of chaotic sequences. In Nissen and Propach (1998), noise function was introduced to the deterministic objective function values in order to map the practical scenario of the existence of noise while experimentation, stochastic simulation, sampling and even interaction with users.

As it can easily be manifested from the above literature review that though a wide range of modifications have been carried out to enhance the performance of EAs, a robust strategy is mandatory that can work in general for various formulations of objective functions found in the real world scenario. Also, an all purpose strategy for robust parameter control and tuning in the presence of noise is essential to help the practitioners of EAs to come out of the dilemma of choosing the control and tuning methods in changing situations.

The aims of this paper are manifold. First, it introduces the concept of developing general-purpose fast EAs that equivalently achieve better results in all the previously mentioned formulations (i.e. **A**, **B**, **C**, and **D**). Second, it utilizes the enhanced learning mechanisms for all these formulations in order to achieve better tradeoff between exploration and exploitation of the search space; and to achieve time gain over prevailing strategies. This has also been theoretically validated. Third, it introduces a new tradeoff function to decide the mutation step size. This function makes a tradeoff between the fast convergence of Cauchy function and the explorative search of Gauss function to achieve better solutions. Fourth, the chaotic sequences to map the random dynamics have also been utilized to mutate the solutions. Thus, the evolutionary strategies utilized in the paper, the mutation at each step is performed both by the tradeoff function and the chaotic sequence generator and the better offspring is evolved further. Fifth, the paper proposes a design of experiments model to conduct various experiments sequentially utilizing the orthogonal array that symbolizes the variation in various parameter values. This approach reduces the tedious task of parameter control and tuning that were earlier carried out by exhaustive experiments. Comparisons with the existing EAs are also carried out that utilize the well-designed experiments, and thus the efficacy of the strategies has been established. Finally, the noise factor has been incorporated in the deterministic objective function value, and designed experiments are conducted to establish the robustness of the algorithm.

The rest of the paper is organized in the following sequence: The next section introduces the main intricacies in defining the general-purpose EAs. Section 3 introduces

new generalized learning strategies and presents the new guided mutation rule. It also provides the general-purpose algorithms for all the four problem types. Section 4 deals with the mathematical aspects of the proposed algorithms and establishes their convergence. Section 5 provides the classification of the test functions utilized to evaluate the effectiveness of the proposed algorithm for numerical optimization problems on the basis of their dimensional complexity and degree of entrapment. It also presents the test problems of combinatorial optimization utilized for the comparative study. Section 6 formulates the design of experiments for the parameter tuning and the analytical study of the proposed algorithm. The results of the extensive computational experiments, inferences drawn from it and related discussions are presented in Section 7. Section 8 concludes the paper.

2. An insight to intricacies in defining general-purpose algorithms

As discussed in the previous section, a general optimization problem can be conveniently framed into one of the formulations depicted in Fig. 1, though, with slight changes. The major obstacle in the presentation of robust algorithms that can equivalently work in any of the varying environment or formulation is the peculiar data type handling method used in it.

Due to varying representation schemes, the general rules for mutation cannot be implemented in all the cases. Hence, the following proposition, though obvious, needs to be stated.

Proposition 1. *No single mutation strategy can equivalently work for all the four problem formulations.*

Validation: Let there be two formulations considered, namely—'**A**' and '**C**'. The formulation of type '**A**' requires the representation of solution as the floating point number, while, the problem type '**C**', which is characterized by the integer vector representation of the solution. In the first case, the new perturbed vector is obtained by adding or subtracting a calculated step (floating point form) into the variable, whereas for the second case the perturbed vector is obtained by the random replacements within allowable range in the existing solution vector. Thus, the two schemes are entirely different; one is based on algebraic addition and other on random combinations. Similar logic can be established for other formulations also. Thus, it can be established that a single strategy for mutation cannot equivalently work in all the cases; however, here it is imperative to be mentioned that the different strategies used may be guided by similar principles with adaptive formulations for all the representations. This fact has motivated the authors to present the generalized adaptive mutation principle for all types of problems considered. □

When learning rules are advocated, the first thing characterizing them is the dynamic learning of the optimal actions for the environment, through interaction with stochastic and unknown environment. Their ability to find out best actions amidst the persisting noise makes them hot case for the EA practitioners. However, in this case also a proposition similar to Proposition 1, can be made for the learning rules utilized in the EAs.

Proposition 2. *No same learning strategies can be used for all the four problem formulations.*

Validation: Let again the cases '**A**' and '**C**' be considered. The formulation type '**C**' utilizes the probability strings to obtain the probability to choose a set of actions among the available ones. However, for a typical formulation type '**A**', the learning can be performed utilizing the environmental (neighborhood) effects on the current solution (detailed in Section 3). Here, the probability strings are of no use, as the set of actions at a particular bit position are very large to handle through probability strings. Thus, the learning rules are defined in the form of solution perturbation rules utilizing the learning schemes. Thus, the two cases are entirely different, that establishes the proposition. □

Although, the above proposition holds, the learning rules can be guided by similar strategies, and the fact has been the prime motivation to devise the general-purpose learning rules that can be utilized in all categories of the problems.

3. The generalized dynamically fast EAs

This paper utilizes the following adaptations and modifications in the existing strategies

- Learning strategy utilizing the neighborhood information and improved updating rule.
- Guided adaptive mutation rule based on the better of the step obtained from tradeoff function between Cauchy and Gaussian distributions, and from the random chaotic distributions.

Based on the aforementioned aspects, the general rules for all the formulations are given in the following discussion.

3.1. The generalized EA exemplification

In general, an EA can be represented by six basic components namely—(1) encoding scheme of chromosomes; (2) fitness evaluation scheme; (3) initial population; (4) set of rules for the offspring generation; (5) set of operators to have comparative evaluation; and (6) working parameters utilized. Earlier, Hung and Adeli (1994) presented GA as a nine tuple entity to represent their parallel genetic/neural network learning algorithm. Utilizing the idea, this paper presents the proposed generalized EAs with a ten tuple entity

$$\mathbf{EA} = (\mathrm{PopS}, \mathrm{Pop}^0, E^{\mathrm{s}}, \acute{L}^{\mathrm{chr}}, F, L^{\mathrm{s}}, M^{\mathrm{r}}, S^{\mathrm{s}}, P, \Gamma), \tag{9}$$

where PopS is the population size; $\mathrm{Pop}^0 = \mathbf{C_h}|\mathbf{C_h} = (\mathrm{Ch}_1^0, \mathrm{Ch}_2^0, \ldots, \mathrm{Ch}_{\mathrm{PopS}}^0)$, $\mathrm{Ch}_i{}^0$ is the ith chromosome of initial population; E^s represents the encoding scheme of chromosomes; L^{chr} denotes the length of a chromosome depending on representation scheme; F is the fitness function for the parents and generated offsprings; L^s represents the learning scheme utilized for the chromosomes; M^r is the mutation rule to obtain new solutions; S^s symbolize the selection scheme utilized to select the chromosomes; P is a set representing the parameter values utilized; Γ corresponds to the termination criteria used. The sections to follow will utilize this representation scheme to detail an EA.

3.2. Improved general-purpose learning strategies

Taking cue from the genetic learning automata and S-type learning automata (Howell et al., 2002), the generalized pursuit learning schemes (Agache and Oommen, 2002), the neighborhood learning of swarm intelligence (Clerc and Kennedy, 2002; van den Bergh and Engelbrecht, 2004), and the generalization theory of learning (Butz et al., 2004) improved and dynamic learning rules have been formulated to guide the search procedure. The learning rule has different updating schemes based on the problem formulation but utilizes the similar updating functions, therefore, is defined as general-purpose learning rule. The proposed general-purpose rule is based on the updating in the form of adaptive self-learning and neighborhood learning. The learning can be in the form of probability updating rule or as trajectory guide rule; however, both are guided by the same rule as described above.

In general, a probability update learning rule utilizes some well-acclaimed terminologies like the action probability vector '**Prob**' and a learning update scheme '$\mathbf{L}^s$', from the learning automata literature, and are defined as

$$\mathbf{Prob}(g) = [p_1(g), p_2(g), \ldots, p_3(g)]^{\mathbf{L}^s}, \tag{10}$$

where $p_i(g)$ is the probability that at generation g, the learning strategy will select the action a_i. $a_i \in \hat{\mathbf{A}}$, $\hat{\mathbf{A}}$ is the set of available actions such that $p_i(g) = \Pr[a(g) = a_i]$, $i \in [1, \check{\mathbf{r}}]$, $\check{\mathbf{r}}$ is the number of actions. Here, $p_i(g)$ satisfies the following condition related to probability definition i.e.:

$$\sum_{i=1}^{\check{r}} p_i(g) = 1 \quad \forall g. \tag{11}$$

In the above discussion, the learning update scheme $\mathbf{L}^s$: $[0, 1]^{\check{\mathbf{r}}} \times \hat{\mathbf{A}} \times \boldsymbol{\beta} \to [0, 1]^{\check{\mathbf{r}}}$, where, $\boldsymbol{\beta}$ is the set of responses from the environment. Thus, a learning update rule can be represented as

$$\mathbf{Prob}(g+1) = \mathbf{L}^s(\mathrm{Prob}(g),\ a(g),\ b(g)), \quad b \in \boldsymbol{\beta}. \tag{12}$$

In the case of probability update rules, the populations of bit strings are replaced by corresponding population of probability strings. At each generation, the probability string is first converted into the action string by the probability distribution, thereafter, the probability strings are again updated utilizing the update rules and responses from the environment; and the process continues. The key factor that guides the search procedure is the probability update rule **U.** Many different rules for updating persists in the literature (Agache and Oommen, 2002; Papadimitriou, 1994; Najim and Poznyak, 1994). In the sequel, this paper presents improved and dynamic probability update rule in order to achieve better tradeoff between exploration and exploitation of the search space, and to achieve time gain over prevailing strategies. The proposed learning update rule can be defined as

$\mathbf{L}^s$:

$$p_j(g+1) = \begin{cases} (1-\varsigma)p_j(g) + (1-\upsilon)p'_j(g) + \frac{\varsigma+\upsilon}{\Xi(g)}, \\ \quad \text{if } \delta_j(g) > \delta_i(g),\ j \neq i, \\ (1-\varsigma)p_j(g) + (1-\upsilon)p'_j(g), \\ \quad \text{if } \delta_j(g) \leqslant \delta_i(g),\ j \neq i,, \\ 1 - \sum_{j \neq i} p_j(g+1), \end{cases} \tag{13}$$

where $p_j(g)$ is the probability at the generation g; p'_j is the probability of neighbors at generation g; ς: $0 < \varsigma 1$, is the speed of learning parameter with respect to previous probability; υ:$0 < \upsilon < 1$, is the speed of learning parameter with respect to probability of neighbors; $\Xi(g)$ represents number of actions with higher estimates than the chosen action at generation g; $\delta_j(g)$ is the reward estimate of action j, defined as

$$\aleph_i(g+1) = \aleph_i(g) + (1 - b(g)), \tag{14}$$

$$\Psi(g+1) = \Psi_i(g) + 1, \tag{15}$$

$$\delta_i(g+1) = \frac{\aleph_i(g+1)}{\Psi_i(g+1)}, \tag{16}$$

where $\Psi_i(g)$ represents the number of times the ith action has been chosen; $\aleph_i(g)$ denotes the number of times ith action has been rewarded. This type of rule is adopted for the formulations **B**, **C**, **D**. In the above formulation the parameters ς and υ are adaptively changed as the number of iterations increase. Their adaptive formulation and its impact are detailed in Section 6.

For the trajectory guide rule, i.e. for formulation **A**, the learning scheme is not based on the probability string as it is not feasible for this formulation. Here also the learning rule is guided by self-learning and neighborhood learning. In this case, the solution is a point-based representation; hence, the learning rule can be formulated for each solution as

$\mathbf{L}^s$:

$$\chi_i(g+1) = \chi_i(g) + \kappa_i(g+1), \tag{17}$$

$$\kappa_i(g+1) = \omega\kappa_i(g) + \varsigma \times \mathrm{ran}(0,1)(\chi_i(g) - \chi_{\mathrm{is}}(g)) + \upsilon \times \mathrm{ran}(0,1)(\chi_i(g) - \chi_{\mathrm{in}}(g)), \tag{18}$$

where ς and υ are the speed of learning parameters depending on self experience and on the neighborhood

experience, respectively; $\chi_i(g)$ is the ith variable position at generation g; χ_{is} and χ_{in} are the respective previous best and neighborhood best positions; κ_i is the update factor and ω is its corresponding dependency on previous iterations. In the trajectory learning rule, each solution acquires its new position based on the learning rules (17) and (18).

3.3. Guided adaptive mutation rule

As indicated from Proposition 1, a general guide rule should be first devised for the adaptive mutation and thereafter, the mutation rules for various encoding schemes should be defined. In this paper, the mutation rule is based on a new tradeoff function to decide the mutation step size to generate a new offspring. This function can be generalized as a tradeoff between the fast convergence of Cauchy function and the explorative search of Gauss function. Since the Cauchy functions are very good at search in a large neighborhood and Gaussian function performs better in small neighborhood (Yao et al., 1999), the adaptive scheme gives more emphasis to Cauchy mutations in the initial phases of the search procedure and the preference of Gaussian mutations is increased as the search proceeds. The Gaussian and Cauchy functions with mean zero are given below:

$$f_{\text{Gaussian}}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x^2/2\sigma^2)} \quad -\infty < x < \infty, \tag{19}$$

$$f_{\text{Cauchy}}(x) = \frac{t}{\pi(t^2 + x^2)} \quad -\infty < x < \infty, \tag{20}$$

where σ is the standard deviation of Gaussian distribution and $t > 0$ is the scale parameter for Cauchy distributions. Fig. 2 presents the comparative plot of both the functions drawn on the same scale.

Also Caponetto et al. (2003) indicated that by substituting the random mutation sequences by some chaotic

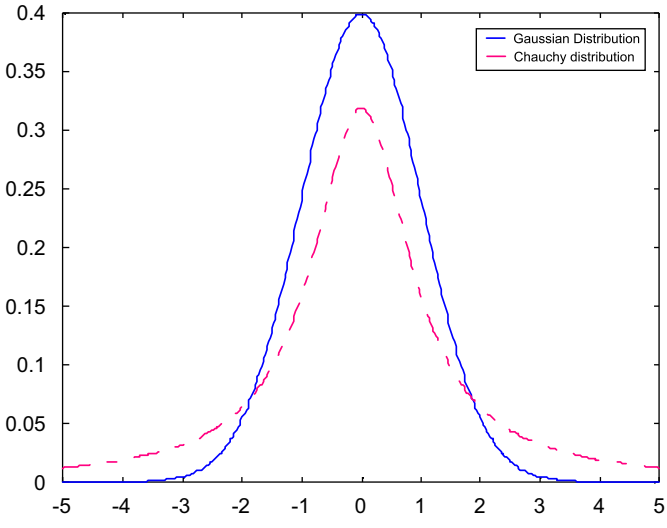

Fig. 2. Plot of Cauchy and Gaussian distributions with standard deviation 1; mean = 0; $t = 1$.

operators always enhance the performance of EAs; hence, the proposed generalized mutation scheme generates another offspring along with the one using tradeoff function, and the better one is considered for further operations. The following discussion formally introduces the mathematical aspects to get the offspring:

$\mathbf{M^r}$:

$$O^c(g) = \begin{cases} O^c_{\text{tcg}}, & \text{if } f(O^c_{\text{tcg}}) > f(O^c_{\text{chaos}}), \\ O^c_{\text{chaos}}, & \text{if } f(O^c_{\text{tcg}}) < f(O^c_{\text{chaos}}), \\ O^c_{\text{tcg}} \diamond O^c_{\text{chaos}}, & \text{otherwise}, \end{cases} \tag{21}$$

where $O^c(g)$ is the offspring generated from the c chromosome at generation g; O^c_{tcg} is the offspring generated from the tradeoff function; O^c_{chaos} represents the offspring generated utilizing the chaotic sequences; $\diamond$ denotes the operator that randomly selects any of the two quantities on its either sides; and $f(\cdot)$ is the fitness function.

The generation of two parallel offsprings is based on the following mutation rule:

$$O^c_i(g) = \text{Ch}^c_i(g) + \mu_i(g) \times \Delta^c_i(g), \tag{22}$$

where O_i^c is ith part of offspring generated from chromosome c at generation g; $\text{Ch}_i^c(g)$ represents the ith part of chromosome c at generation g; $\mu_i(g)$ is known as the standard deviation for the mutation; $\Delta_i^c(g)$ is the mutation step size obtained either from the tradeoff function or from chaotic sequences at generation g. The case pertaining to the tradeoff function is described below:

$$\Delta^c_i = \text{ƛ}(g) \times \mathbf{G_s} + (1 - \text{ƛ}(g)) \times \mathbf{C_s}, \tag{23}$$

where

$$\text{ƛ} = \frac{g}{g_{\max}}. \tag{24}$$

$\mathbf{G_s}$ and $\mathbf{C_s}$ are random steps generated utilizing Gaussian and Cauchy distribution functions. Mathematically, step size generation can be described as

$$\mathbf{G_s} = \text{rand}(+,-)\sqrt{2 \times \ln(z \times \sqrt{2\pi})}, \tag{25}$$

$$\mathbf{C_s} = \text{rand}(+,-)\sqrt{t \times \left(\frac{1}{z \times \pi} - t\right)}, \tag{26}$$

where rand$(+,-)$ randomly assigns sign to the expression; z is a random number in the range $]0, f_{\text{Gaussian}}(0)$ or $f_{\text{Cauchy}}(0)$ (as the case may be)$[$. Eqs. (25) and (26) can easily be obtained from (19) and (20), respectively.

However, for the offspring generation from chaotic sequences, many chaotic generators are defined in the literature (Caponetto et al., 2003). This paper utilizes the tradeoff function of the sinusoidal iterator and logistic map due to their better performance and fast convergence. For the offspring generation, the following formulation of $\Delta_i^c(g)$ is used:

$$\Delta^c_i = \text{rand}(+,-) \times \acute{s} \times [\hbar \times £ + (1 - \hbar) \times \check{S}], \tag{27}$$

where $\acute{s}$ is the step parameter; $\hbar$ is a random number in range $(0,1)$; £ and $\tilde{S}$ are the random numbers obtained from the logistic map and sinusoidal iterator defined as

Logistic map: xc

$$z_{k+1} = a' \times z_k(1 - z_k), \quad k = 0, 1, \ldots, 400. \tag{28}$$

Sinusoidal iterator:

$$z_{k+1} = a'' \times z_k^2 \times \sin(\pi z_k), \tag{29}$$

where a' and a'' are equation parameters. The step sizes obtained from Eq. (27) are shown in Fig. 3.

3.4. The generalized EAs for various formulations

This subsection formally presents the generalized EAs for all the four formulations. The basics of all these EAs are guided by similar principles, though, varying somewhat in implementation. The EAs represented as abstract ten tuple entities are presented in Table 1. In the interest of brevity, detailed algorithms for all the four problem types are given in Table 2. Without the loss of generality, this paper tries to put encoding schemes, learning strategies, mutation rules, and similar other tuples of all the four EAs on the same platform. This means that though the general encoding schemes for the problem types may be as shown in Fig. 1, they are somewhat modified to suit the broader objective of generalization. Also, it has been shown in the sections to follow that the proposed encoding schemes work equivalently or sometimes better than the existing strategies.

3.4.1. Type *A* problems

The problem type **A** is generally concerned with the function optimization problems that require variable values as floating point numbers. The algorithm for type **A** utilizes the encoding of variables as floating point numbers. The learning scheme utilizes the information shared by the neighbors (other chromosomes) and the previous experience of the chromosome. The mutation step is straightforward and requires the algebraic summation of mutation step obtained from (21). This algorithm is characterized by the dynamic updating and fast convergence, authenticity for which has been theoretically established in Section 4, and experimentally revealed in Section 7.

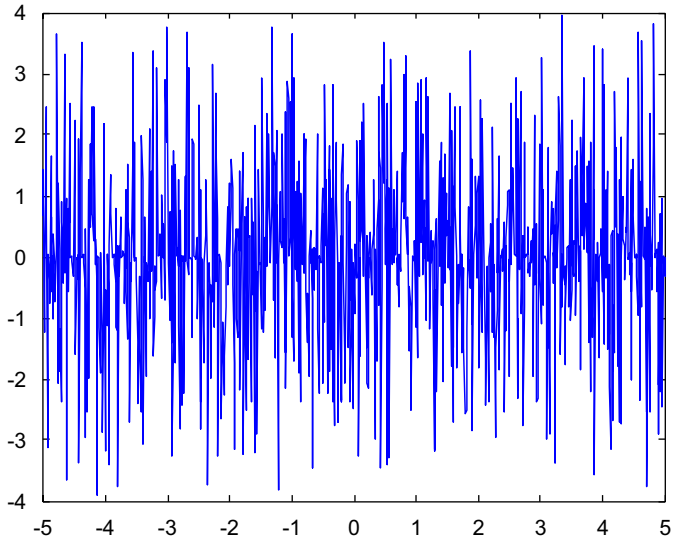

Fig. 3. Chaotic steps from the proposed chaotic rule with $s' = a' = a'' = 4.0$.

3.4.2. Type *B* problems

The type **B** problems are also generalized case of function optimization problems. The difference lies in the encoding scheme, where the binary encoding is utilized to map the decision variables. Here also, the mutation rule is straightforward and requires the binary addition of the mutation step size obtained. However, the learning scheme utilizes the probabilistic strings to get the probability of being '1' or '0' at any particular bit position. This algorithm is also marked by the fast convergence to near-optimal solutions, as validated by the Sections 4 and 7.

3.4.3. Type *C* problems

In the same fashion, type **C** problems are generalized cases of TSP problem or their modified versions. Here, the proposed algorithms utilize different type of coding schemes than what has been generally established in the literature. The type **C** problems are intended to find best sequence of nodes or objects among the given. In the interest of generality of the proposed algorithms, the encoding scheme utilizes partial binary coding, where a bit position can be acquired by either 0 or 1 but they are restricted by the following conditions and characteristics:

- The chromosome has length $\sum_{i=1}^{n-1} n - i$, where n is the number of nodes and i represents the ith node of the sequence.
- There can be only a single '1' in each subsequent $n-i$ bits, $i \in 1, 2, \ldots, n-1$.
- To decode the solution, a lookup variable string is utilized whose initial length is n and the ith position contains the integer i.

The decoding steps are detailed in the algorithm presented in Table 2.

In this case also the learning rule is similar to the type **B** case. Here, due to the use of different encoding scheme, the mutation scheme utilized in both previously mentioned type of problems is not applicable. First, '$n-1$' mutation step sizes are calculated with the similar rules as previously detailed. The step sizes obtained in the range of $\Delta_{\min}$ and $\Delta_{\max}$ are then scaled on the range of $(0, n-i)$ for each subsequent i bits, $i \in 1, 2, \ldots, n-1$. Now, based on the step size obtained, the position of single '1' is jumped and the new solution is obtained. This feature of the algorithm has also detailed in Table 2.

Table 1
The EA abstraction

The EA abstraction:	Type '**A**' problems	Type '**B**' problems	Type '**C**' problems	Type '**D**' problems
PopS	User defined parameter	User defined parameter	User defined parameter	User defined parameter
Pop^0	$= (Ch_1^0, Ch_1^0 \ldots Ch_{PopS}^0)$ $= \left(\begin{bmatrix} Ch_{1,1} \\ . \\ Ch_{1,L^{chr}} \end{bmatrix} \ldots \begin{bmatrix} Ch_{PopS,1} \\ . \\ Ch_{PopS,L^{chr}} \end{bmatrix} \right) \in E^s;$	Similar to type '**A**'	Similar to type '**A**'	Similar to type '**A**'
E^s	$\{\underline{t}_i \leqslant t_i \leqslant \bar{t}_i, \quad i \in \{1,2,\ldots,n\}\}$ t_i, n have similar meanings as in (3);	(0,1) L^{chr} (i.e. probability string)	(0,1) L^{chr} (i.e. probability string)	(0,1) L^{chr} (i.e. probability string)
L^{chr}	n;	If ith variable is encoded as a n_i digit binary number, the total length of chromosome $= \sum_{i=1}^{n} n_i$	Length of chromosome $= \sum_{i=1}^{n-1} n - i$	If number of alternatives for ith variable are n_{ai}, the total length of chromosome $= \sum_{i=1}^{n} n_{ai}$
F	$1/g(\mathbf{t})$; (Eq. (1))	$1/g(\mathbf{t})$; (Eq. (1))	$1/f_{TSP}(\mathbf{t})$; (Eq. (7))	$1/h(\mathbf{t})$; (Eq. (4))
L^s	$\approx$Eqs (17) and (18);	$\approx$Eq. (13)	$\approx$Eq. (13)	$\approx$Eq. (13)
M^s	Mutation step $\approx$Eq. (20), The mutation step is directly added to the concerning variable;	Mutation step $\approx$Eq. (21). The mutation step size converted to binary number and added to parent string;	Mutation step (Δ) $\approx$Eq. (21). The bit having '1' (in every segment) is jumped by Δ cells. (detailed in algorithm description).	Mutation step (Δ) $\approx$Eq. (21). The bit having '1' (in every segment) is jumped by Δ cells. (detailed in algorithm description).
S^s	A fixed percentage 'f_p' of best offsprings replace the f_p% worst parents.	The probability strings of populations are dynamically updated.	The probability strings of populations are dynamically updated.	The probability strings of populations are dynamically updated.
P	User defined values.	User defined values.	User defined values.	User defined values.
Γ	Number of generations.	Number of generations/ convergence of probability strings approximately to 1.	Number of generations/ convergence of probability strings is approximately to 1.	Number of generations/ convergence of probability strings approximately to 1.

3.4.4. Type **D** problems

These types of problems are modified versions of TSPs. In these types of problems, each problem variable has a finite set of alternatives and the problem is to choose the best out of them. The chromosome length is the sum of the total number of alternatives available. The encoding is similar to type **C** problems. The presence of '1' in a bit position ensures the selection of that alternative. The mutation and learning schemes are similar to the type **C** case (Table 2).

Further, an elite population has been maintained in each class of problem that contains e_{pop} best individuals found till the generation.

4. Mathematical aspects of the proposed strategies

4.1. Can the learning rule converge?

To show the convergence, the learning strategies are split into two parts—first, the learning strategies which utilize Eq. (13) and second, those utilizing Eq. (17). Let the case of first type be taken. The convergence of the strategy can be proved in two steps. First, the following theorem is established.

Theorem 1. *For some* $\aleph > 0$ and $И < \infty$, $\exists \varsigma^* > 0$, $\upsilon^* > 0$ *and* $g_{initial} < \infty$ *such that* $\forall \varsigma \in (0, \varsigma^*) \wedge \upsilon \in (0, \upsilon^*)$ *under* $\mathbf{L}^s$, *probability* ρ **(***all actions are chosen at least* $И$ *times before* g**)** $> 1 - \aleph$, $\forall g \geqslant g_{initial}$.

Proof. This theorem is parallel to that for TSE learning rule (Tathachar and Sastry, 1986). Let $\mathscr{P}_g^i$ represent a random variable as the number of times the ith action was chosen upto generation g in any specific algorithmic run. If the action a_i is chosen at generation g, then from (13),

$$p_j(g) = \begin{cases} (1-\varsigma)p_j(g-1) + (1-\upsilon)p_j'(g-1) + \frac{\varsigma+\upsilon}{\Xi(g)-1} \\ \quad \text{if } \delta_j(g-1) > \delta_i(g-1),\ j \neq i, \\ (1-\varsigma)p_j(g-1) + (1-\upsilon)p_j'(g-1) \\ \quad \text{if } \delta_j(g-1) \leqslant \delta_i(g-1),\ j \neq i. \end{cases} \tag{30}$$

Let the term $(1-\varsigma)p_i(g-1) + (1-\upsilon)p_i'(g-1)$ be written as $(1-\eta) \times \overline{p_i}$, where $\eta = 1 - [(1-\varsigma) + (1-\upsilon)]$ and

$$\overline{p_i} = \frac{(1-\varsigma)p_i(g-1) + (1-\upsilon)p_i'(g-1)}{(1-\varsigma) + (1-\upsilon)}. \tag{31}$$

Now, the two cases arise—(1) the probability of the chosen action, p_i (g) is either $(1-\eta) \times \overline{p_i}$, when $\exists$ other actions j for which estimates δ are better than those for a_i, or, (2) p_i (g) is $(1-\eta) \times \overline{p_i} + \varsigma$ when the action chosen has maximal reward estimate δ_{max}. However, in both of the cases, the following inequality holds:

$$p_i(g) \geqslant (1-\eta) \times \overline{p_i}. \tag{32}$$

Again from (13) it can be concluded that a similar inequality is valid $\forall j \in [1, \check{\mathbf{r}}]$. Thus, it can be concluded that for any g initial generations of the algorithm,

$$\rho\,(a_i \text{ is chosen}) \geqslant (1-\eta)^g \times \overline{p_i}(o) \quad \text{for any } i \in [1, \check{\mathbf{r}}]. \tag{33}$$

With the Eq. (33) established, the remaining proof is similar to that for TSE learning rule (Tathachar and Sastry, 1986), and hence is not presented for the sake of conciseness. □

Now, in order to establish the convergence of the proposed learning rule, it is to be proved that if ith action is rewarded more number of times from g_{initial} onwards, as compared to any other action, then the probability vector of actions converges with a probability one.

Theorem 2. *Let there be an action index* $i \in [1, \check{\mathbf{r}}]$ *and* $g_{\text{initial}} < \infty$ *such that* $\check{\delta}_i(g) > \check{\delta}_j(g)$; $\forall (j \neq i)(\hat{g} > g_{\text{initial}})$, *where* $\check{\delta}_i(g) = \max_{i \in [1,\check{r}]}\{\delta_i(g)\}$, *then* $p_i(g) \to 1$ *with probability* 1 *and* $g \to \infty$.

Proof:. The convergence of the theorem is proved using the *Submartingle's convergence theorem* (Narendra and Tathachar, 1989). For this purpose, it is first proved that the sequence of random variables $\{p_i(g)\}_{g \geqslant g_{\text{initial}}}$ is a *submartingle*.

To start with, the following equation is considered:

$$\Delta p_i(g) = E[p_i(g+1) - p_i(g) | \mathbf{St}(g)], \tag{34}$$

where $\mathbf{St}(g)$ is the state vector for the estimator algorithms that contains Prob(g) and $\boldsymbol{\delta}(g)$: $\boldsymbol{\delta}(g) = \{\delta_i(g),\ \forall i \in [1, \check{r}]\}$. From the proposed learning rule and the theorem assumptions, the probability for $g+1$th generation can be represented as

$$p_i(g+1) = (1-\eta) \times \overline{p_i} + \frac{\varsigma}{\Xi(g)} \quad \text{if } a_j \text{ is chosen and } j \neq i,$$

$$p_i(g+1) = 1 - \sum_{j \neq i}[(1-\eta) \times \overline{p_j}]$$

$$= \overline{p_i} + \eta \times (1 - \overline{p_j}) \quad \text{if } a_i \text{ is chosen.}$$

Thus, it can be inferred that $\forall g \geqslant g_{\text{initial}}$, $\Delta p_i(g)$ can be calculated as

$$\Delta p_i(g) = \sum_{j \neq i}\left(\frac{\eta}{\Xi(g)} - \eta \overline{p}_i(g)\right)\overline{p}_j(g) + [\eta(1 - \overline{p}_i(g))]\overline{p}_i(g), \tag{35}$$

$$\Delta p_i(g) = \eta\left(\frac{1}{\Xi(g)} - p_i(g)\right)(1 - p_i(g)) + \eta(1 - p_i(g))p_j(g). \tag{36}$$

Table 2
The generalized EAs

Algorithm for type A

Input: PopS, $\hbar$, ś, ς, υ, ω, $\text{gen}_{\max}$, n, constraints c_j, **g**(t), e_{pop}; $\grave{L}^{chr}$=n;
Sample chromosome:

1	2	….	….	n
3.25	-1.24	…	…	-1.12

Randomly **initialize** population Pop^0 v(*i.e* .give random feasible values to each variable of chromosome);
Evaluate (Pop^0);
Insert e_{pop} best individuals to Pop_{elite};
for each generation
do
  for each individual
  do
    generate offspring 1 using (23);
    generate offspring 2 using (27);
    Evaluate (offspring 1 and offspring 2);
    If *f*(offspring 1)> *f*(offspring 1)
      offspring = offspring 1;
    else
      offspring = offspring 2;
    end
    If *f*(individual)> *f*(offspring)
      individual = offspring ;
    end
    Apply learning rule of (17) over offspring and generate newindividual;
    If *f*(newindividual)>*f*(individual)
      individual = newindividual;
    end
    If *f*(individual)>*f*(worst_elite)
      Replace worst_elite by individual;
    end
  end
end
Give best_elite as **output**;

Algorithm for type B

Input: PopS, ς, υ, $\hbar$, ś, $\text{gen}_{\max}$, n, costraints c_j, **g**(t), e_{pop};

$\grave{L}^{chr} = \sum_{i=1}^{n} m : (2^{m-1} < (\bar{t}_i - \underline{t}_i) < 2^m)$;

Sample chromosome:

1	2	….	….	$\grave{L}^{chr}$
0.125	0.15	0.98	…	0.65

Randomly **initialize** population Pop^0 by 0.5.
Generate binary Pop^0_{binary} from the Pop^0;
Evaluate (Pop^0);
Insert e_{pop} best individuals to Pop_{elit};
while (probability of a bit does not converge to 1 or gen<$\text{gen}_{\max}$)
do
  for each individual
  do
    generate binary individual from the probabilistic individual;
    generate binary offspring 1 *i.e.* $off_{bin}1$ using (23);
    generate binary offspring 2 *i.e.* $off_{bin}2$ using (27);
    /* Here, binary addition of mutation step size is performed */
    /* generate the new offspring */
    Evaluate ($off_{bin}1$ and $off_{bin}2$);
    If *f*($off_{bin}1$)> *f*($off_{bin}2$)
      off_{bin} = $off_{bin}1$;
    else
      off_{bin} = $off_{bin}2$;
    end
    If *f*($individual_{binary}$)> *f*(off_{bin})
      $individual_{binary}$=off_{bin};
    end
    If *f*($individual_{binary}$)>*f*(worst_elite)
      Replace worst_elite by $individual_{binary}$;
    end
    update the probabilistic individual from (13)
  end
end
Give best_elite as **output**;

Table 2 (*continued*)

Algorithm for type **C**

Input: PopS, ς, υ, $\hbar$, ś, gen_{max}, n, costraints c''$_j$, $\mathbf{f}_{TSP}$, e_{pop};

$\grave{\mathbf{L}}^{\mathbf{chr}} = \sum_{i=1}^{n-1} n-i$;

Sample chromosome:

1	**2**	**….**	**….**	$\grave{\mathbf{L}}^{\mathbf{chr}}$
0.125	0.15	0.98	…	0.65

Randomly **initialize** population Pop^0 by 0.5.
Generate binary Pop^0_{binary} from the Pop^0;
/* To generate binary population, probabilistically obtain a */
/* position in every consecutive n-i bits and assign 1 to the */
/* position and 0 to the rest positions */
Evaluate (Pop^0);
/* The evaluation is performed on the basis of variable **lookup** string as shown below:*/

1	2	…	…	n

/*Using the lookup string, a **sequence** string is created by the following steps*/
for each individual
do
  for each subpart of i of individual
  do
    if '1' is present in j^{th} position
      sequence(i)=individual(j);
    end
    delete j^{th} position in **lookup** string;
  end
end
individual(n)=remaining element of lookup string;
/*Using sequence string, evaluate each individual */
Insert e_{pop} best individuals to Pop_{elite};
while (probability of a bit does not converge to 1 or gen<gen_{max})
do
  for each individual
  do
    generate binary individual from the probabilistic individual;
    generate binary offspring 1 *i.e.* $off_{bin}1$ using (23);
    generate binary offspring 2 *i.e.* $off_{bin}2$ using (27);
    /* Here, the mutation is not a directly visualized by the mutation step size obtained*/
      To generate mutated offspring,
        Calculate mutation step size from (24) and (28);
        Get maximum step size in all variables;
        Scale all step sized in the range (0,1) w.r.t. maximum step size;
        Multiply each scaled value with (n-i) to get step size Δ for i^{th} subpart.
        Jump the corresponding '1' in i^{th} subpart by Δ and get the mutated offspring
    /*
  */
    …
    … Rest of the steps follow the same steps as the algorithm
    … for type **B**
    …
    update the probabilistic individual from (13)
  end
end
Give best_elite as **output**;

Algorithm for type **D**

Input: PopS, ς, υ, $\hbar$, ś, gen_{max}, n, **S**, costraints c'$_j$, **h**(s), e_{pop};

$\grave{\mathbf{L}}^{\mathbf{chr}} = \sum_{i=1}^{s} n_S$; where n_s is set of alternatives for each of n variables;

Sample chromosome:

1	**2**	**….**	**….**	$\grave{\mathbf{L}}^{\mathbf{chr}}$
0.125	0.15	0.98	…	0.65

Randomly **initialize** population Pop^0 by 0.5.
Generate binary Pop^0_{binary} from the Pop^0;
/* To generate binary population, probabilistically obtain a */
/* position in every n_s bits and assign 1 to the */
/* position and 0 to the rest positions */
Evaluate (Pop^0);
/* To evaluate each variable is assigned that value corresponding to which '1' is present*/
Insert e_{pop} best individuals to Pop_{elite};
while (probability of a bit does not converge to 1 or gen<gen_{max})
do
  for each individual
  do
    generate binary individual from the probabilistic individual;
    generate binary offspring 1 *i.e.* $off_{bin}1$ using (23);
    generate binary offspring 2 *i.e.* $off_{bin}2$ using (27);
    /* Here, the mutation is not a directly visualized by the mutation step size obtained*/
      To generate mutated offspring,
        Calculate mutation step size from (24) and (28);
        Get maximum step size in all variables;
        Scale all step sized in the range (0,1) w.r.t. maximum step size;
        Multiply each scaled value with n_s to get step size Δ for i^{th} subpart.
        Jump the corresponding '1' in i^{th} subpart by Δ and get the mutated offspring
    /*
  */
    …
    … Rest of the steps follow the same steps as the algorithm
    … for type **B**
    …
    update the probabilistic individual from (13)
  end
end
Give best_elite as **output**;

Therefore,

$$\Delta p_i(g) = \frac{\eta}{\Xi(g)}(1 - p_i(g)) \geqslant 0. \tag{37}$$

Thus, it can be concluded that $p_i(g)$ is a *submartingle*. Hence, by *submartingle* convergence theorem $\{p_i(g)\}_{g \geqslant g_{\text{initial}}}$ converges as $g \to \infty$ and

$$E[p_i(g+1) - p_i(g) | \mathbf{St}(g)] \to 0 \text{ with a probability 1.} \tag{38}$$

Thus, $p_i(g) \to 1$ with probability one that proves the theorem. □

Now, the result of ε-optimal convergence condition can be given as

Theorem 3. *In every static random scenario,* $\exists$ *a number* $\varsigma^* > 0$, $\upsilon^* > 0$ *and* $g_{\text{initial}} > 0$, *such that* $\forall \varsigma \in (0, \varsigma^*) \hat{\upsilon} \in (0, \upsilon^*)$, *for any* $\daleth \in (0,1)$ *and any* $\varepsilon \in (0,1)$:

$$\text{Prob}[p_i(g) > 1 - \varepsilon] > 1 - \daleth \quad \forall g > g_{\text{initial}}. \tag{39}$$

Proof:. Trivial from Theorems 1 and 2 (as a logical consequence). □

Thus, the proposed learning rule is converging that theoretically authenticates its formulation. With the similar strategies, the theoretical convergence of second type of learning rule can be established.

4.2. Why new tradeoff function?

This paper proposes a new tradeoff function that is utilized to carry out mutations. In general, an efficient search strategy should be characterized by more exploratory

search in the initial phases and thereafter converge by exploiting the neighborhood rigorously. Let the case of numerical optimization pertaining to the type **A** problems be analyzed. If Gaussian distribution is used for mutation operation in Eq. (22), the expected value of mutation jump can be given as (Yao et al., 1999)

$$E_{\text{Gaussian}}(x) = 2\int_0^{+\infty} x\frac{1}{\sqrt{2\pi}}\,\mathrm{e}^{-(x^2/2)}\,\mathrm{d}x = \frac{2}{\sqrt{2\pi}} = 0.80, \tag{40}$$

$$E_{\text{Cauchy}}(x) = 2\int_0^{+\infty} x\frac{1}{\pi(1+x^2)}\,\mathrm{d}x = +\infty. \tag{41}$$

Thus, the Gaussian mutations can be regarded as more localized than Cauchy mutations. It is to be noted that the generation of infinite variation is practically not feasible; however, (41) indicates that it is possible to get large deviations even at first generation—thus, avoiding possibility of entrapment in local optima. In order to establish the relevance of the proposed tradeoff function, first the effect of mutation step size over the search is analyzed mathematically.

Let the probability of generating a solution in the neighborhood of x^* (optimal point) using any distribution D be $P_{\text{D}}(|x-x^*|\leqslant\varepsilon)$, where, ε is the neighborhood size. The following theorem is stated

Theorem 4. *For any number* α *such that* $0<\alpha<2\varepsilon$, *larger the value of* σ *implies larger* P_{Gaussian} $(|x-x^*|\leqslant\varepsilon)$, *if* $\sigma<|x-\varepsilon+\alpha|$; *and larger value of* σ *corresponds to smaller* P_{Gaussian} $(|x-x^*|\leqslant\varepsilon)$, *if* $\sigma>|x-\varepsilon+\alpha|$.

Proof. Fig. 4 demonstrates the various variables and their assumed positions over the arbitrary probability distribution curve. For the Gaussian distribution function (19), the probability that the a point is generated in the neighborhood of x^* can be given as

$$P_{\text{Gaussian}}(|x-x^*|\leqslant\varepsilon) = \int_{x^*-\varepsilon}^{x^*+\varepsilon} f_{\text{Gaussian}}(x)\,\mathrm{d}x. \tag{42}$$

It has the step size equal to $\sigma\cdot$ To assess the impact of σ, the derivative of (42) is to be evaluated with respect to σ. The mean value theorem of definite integrals (Hunt, 1986), states that $\exists$ a number α such that

$$\int_{x^*-\varepsilon}^{x^*+\varepsilon} f_{\text{Gaussian}}(x)\,\mathrm{d}x = 2\varepsilon f_{\text{Gaussian}}(x-\varepsilon+\alpha). \tag{43}$$

Thus,

$$\frac{\partial}{\partial\sigma}P_{\text{Gaussian}}(|x-x^*|=\varepsilon) = \frac{\partial}{\partial\sigma}(2\varepsilon f_{\text{Gaussian}}(x-\varepsilon+\alpha)). \tag{44}$$

Following the logical consequences to obtain derivative, (44) can easily be written as (Yao et al., 1999)

$$\frac{\partial}{\partial\sigma}P_{\text{Gaussian}}(|x-x^*|=\varepsilon) = \frac{2\varepsilon}{\sigma^2\sqrt{2\pi}}\,\mathrm{e}^{-(x^*-\varepsilon+\alpha)^2/2\sigma^2}\left(\frac{(x^*-\varepsilon+\alpha)^2}{\sigma^2}-1\right). \tag{45}$$

Thus,

$$\frac{\partial}{\partial\sigma}P_{\text{Gaussian}}(|x-x^*|=\varepsilon)>0 \quad \text{if } \sigma<|x^*-\varepsilon+\alpha|, \tag{46}$$

$$\frac{\partial}{\partial\sigma}P_{\text{Gaussian}}(|x-x^*|=\varepsilon)<0 \quad \text{if } \sigma>|x^*-\varepsilon+\alpha|. \tag{47}$$

Hence, from (46) and (47), it can be concluded that larger the value of σ implies larger P_{Gaussian} $(|x-x^*|\leqslant\varepsilon)$, if $\sigma<|x-\varepsilon+\alpha|$; and larger value of σ corresponds to smaller P_{Gaussian} $(|x-x^*|\leqslant\varepsilon)$, if $\sigma>|x-\varepsilon+\alpha|$; that in turn validates the theorem. □

A similar theorem can be stated for the Cauchy distribution function (Eq. (20)).

Theorem 5. *For any number* α *such that* $0<\alpha<2\varepsilon$, *larger the value of* 't' *implies larger* P_{Gaussian} $(|x-x^*|\leqslant\varepsilon)$, *if* $t<|x-\varepsilon+\alpha|$; *whereas larger value of* 't' *corresponds to smaller* P_{Gaussian} $(|x-x^*|\leqslant\varepsilon)$, *if* $t>|x-\varepsilon+\alpha|$.

Proof:. The proof can easily be followed from that of Theorem 4. □

Since, σ and t can be directly linked with mutation step size, it can be concluded that larger step size is disadvantageous if the distance between current search point and neighborhood of optimal point is less than the step size; however ,in the opposite case the larger step sizes are advantageous in order to reach the optimal/near-optimal point. Thus, using any one particular type of probability distribution for mutation may deteriorate the efficiency of the search procedure. This may be attributed to the fact that in the initial stages, the population is formed in a perfectly random manner, which requires explorative and thorough search with large mutation steps, whereas, in the

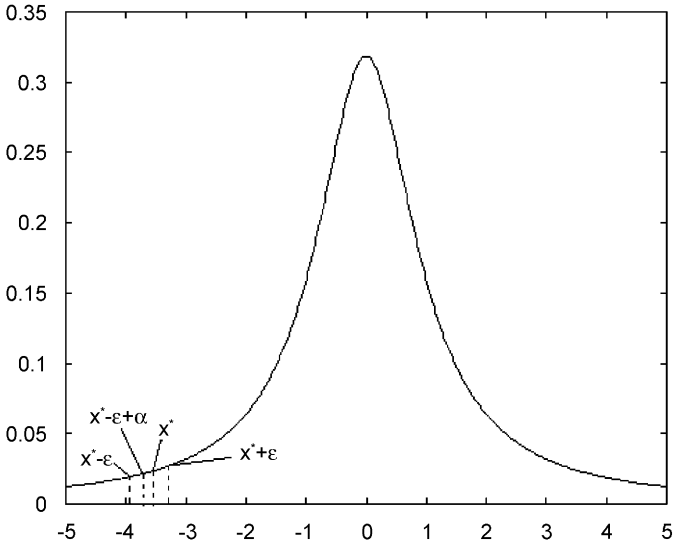

Fig. 4. Plot to demonstrate neighborhood search. The curve is of probability distribution function.

later stages more thorough neighborhood search is required, though giving some chances to large steps. The proposed tradeoff function (Eq. (24)) takes care of such requirements efficiently due to the generation-dependent adaptive parameter 'ƛ'. The following discussion is dedicated to mathematical analysis of the attributes and efficacy of the proposed tradeoff function.

4.3. Are the mean-square displacements promising?

The displacement $\phi(g)$ at any generation g can be attributed to the cumulative steps during g generations. The proposed tradeoff function exhibits the following expectation property of root mean-square displacements:

$$\sqrt{\langle \Phi(g) \rangle^2} = \infty. \tag{48}$$

This is due to the fact that the expected root mean-square displacement for Cauchy mutations is infinity, thus influencing the expectancy for tradeoff function, thanks to the chances given to Cauchy mutations in tradeoff function even at later generations. To empirically evaluate the effect of displacements, Fig. 5 has been plotted that compares the effect of mutations due to the three distribution functions. Here, it is evident that the tradeoff mutations initially explore the region thoroughly and at later generations, the search is more explorative in neighborhood regions. This fact is visualized by the diamond patches. The existence of some patches far away can be interpreted as the effect of Cauchy mutations at later generations. Thus, the mutations of the proposed distribution are more likely to reach at better solutions without entrapping into local optimas.

4.4. Generation of distinct values from mutation

Any offspring at a generation g can be considered as an outcome of g-step random walk following a particular random distribution. The number of distinct values $N_d(g)$ obtained upto generation g is an important criterion to decide the effectiveness of mutation operation. To have a better appraisal, the expectation $\langle N_d(g) \rangle$ is calculated. Let $p_{g'}$ be the probability that the value obtained at generation g' is new. Since obtaining a new value or not is a Bernoulli trial, hence the expected number of distinct values can be given as

$$\langle N_d(g) \rangle = \sum_{g'}^{g} p_{g'}. \tag{49}$$

Now, the value of $\langle N_d(g) \rangle$ is analyzed for Gaussian, Cauchy (Yao et al., 1999) and tradeoff distributions:

$$\langle N_d(g) \rangle_{\text{Gaussian}} \propto g^{1/2}, \tag{50}$$

$$\langle N_d(g) \rangle_{\text{Cauchy}} \propto \frac{g}{\ln(g)}, \tag{51}$$

$$\langle N_d(g) \rangle_{\text{Tradeoff}} \propto \frac{g}{\ln(g)}. \tag{52}$$

Thus, from the above equations, it can be seen that the proposed tradeoff function yields more distinct values than Gaussian distribution, while equivalent to Cauchy distribution. Thus, the tradeoff function has more explorative and distinct search.

Although all the aforementioned proofs shown keeping in mind the type **A** and **B** problems, similar arguments can be established for the remaining cases also. Therefore, it can be said that the new learning and mutation rules are theoretically promising. Further sections would provide the empirical study of these attributes.

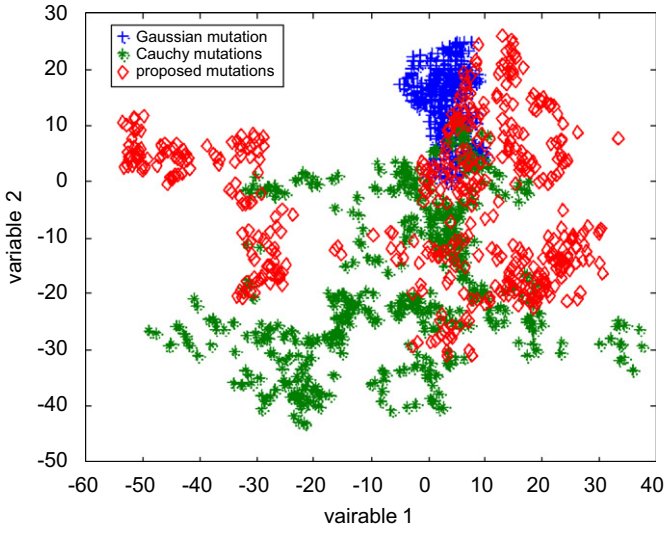

Fig. 5. Mutations obtained by utilizing different distributions over 500 generations and two variables x_1 and x_2. The initial values are set to 0.

5. Classification of the test problems

5.1. Test functions pertaining to type *A* and type *B*

To evaluate the performance of the proposed EAs and carry out the designed experiments for the numerical optimization problems (**A** and **B**), various benchmark test functions available in the literature have been utilized (Yao et al., 1999). Broadly the following classification of test functions has been utilized:

(1) Unimodal functions, f_1–f_3 (i.e. with no local minima).
(2) Multimodal functions with many local minima, f_4–f_5.
(3) Multimodal functions with few local minima, f_6–f_7.

The functions and their characteristic curves for the two variables have been shown in Fig. 6 and Table 3. Each category has two/three functions with increasing complexity. Several plots in Fig. 6 have been shown to visualize the general trend followed by the functions in the case of two variables.

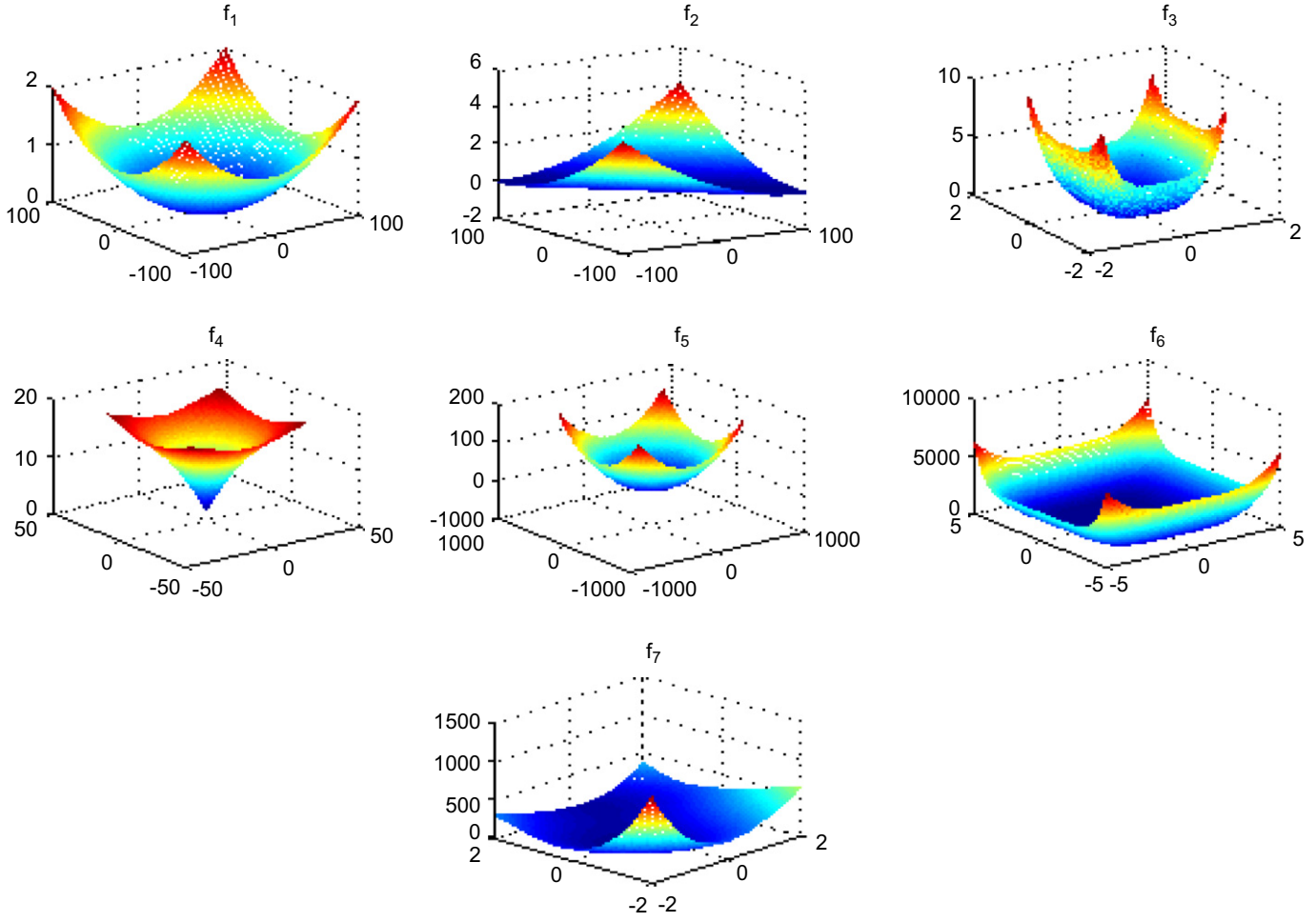

Fig. 6. Graphs of the test functions (f_1–f_7) with $n = 2$.

5.2. Test problem pertaining to type **C** formulations

To evaluate the performance EA pertaining to type **C** problems, two real world TSPs (http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB) have been utilized along with five simulated small size academic problems. The objective has been to minimize the total distance covered (Eq. (7)). The related data set has been provided in Appendix A.

5.3. Test problem related to type **D** formulation

Further, to assess the applicability and performance of the proposed EA for type **D** problems, a real-time reliability optimization problem has been considered from the literature (Gen and Cheng, 1999). This problem had been a benchmark problem for various researchers (Gen, 1975; Gen et al., 1989). The problem is to maximize the system reliability that is subjected to three nonlinear constraints with parallel redundant units in the subsystem. These subsystems are subject to type *A* failures that occur when entire subsystem is subjected to failure condition. Its mathematical formulation can be given as

$$\text{Max } f_{\text{reliability}} = \prod_{i=1}^{3}\left[1-[1-(1-q_{i1})^{m_i+1}]-\sum_{u=2}^{4}(q_{iu})^{m_i+1}\right]. \tag{53}$$

Subject to

$$C_1''(\mathbf{m}) = (m_1+3)^2+(m_2)^2+(m_3)^2 \leqslant 51, \tag{54}$$

$$C_2''(\mathbf{m}) = 20\sum_{i=1}^{3}(m_i+\exp(-m_i)) \geqslant 120, \tag{55}$$

$$C_3''(\mathbf{m}) = 20\sum_{i=1}^{3}(m_i\exp(-m_i/4)) \geqslant 65, \tag{56}$$

$$1 \leqslant m_1 \leqslant 4, \quad m_2 \leqslant 1, \quad m_2 \leqslant 7, \tag{57}$$

$$m_i \geqslant 0: \ \text{integer}, \quad i = 1,2,3, \tag{58}$$

where $\mathbf{m{:}m} = \{m_i, \forall i\}$ and m_i denotes redundant units in the system; q_{iu} represents failure probability for ith failure

Table 3
Test functions for type **A** and type **B**

Test function	Expression of function	N (number of variables)	Solution space	$f_{\min}$
f_1 (Sphere function)	$\sum_{i=1}^{30} x_i^2$	2	$[-100, 100]^2$	0
f_2 (Shwefell's function)	$\sum_{i=1}^{30} \left(\sum_{i=1}^{i} x_j^2 \right)^2$	2	$[-100, 100]^2$	0
f_3 (Quartic function or noise)	$\sum_{i=1}^{30} (ix_i^4 + \text{random}[0, 1))$	2	$[-1.28, 1.28]^2$	0
f_4 (Ackley's function)	$-20 \exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30} x_i^2} \right) - \exp\left(\frac{1}{30}\sum_{i=1}^{30} \cos(2\pi x_i) \right) + 20 + \text{e}$	2	$[-32, 32]^2$	0
f_4 (Griewank function)	$\frac{1}{4000}\sum_{i=1}^{30} x_i^2 - \prod_{i=1}^{30} \cos\left(\frac{x_i}{\sqrt{i}}\right) + i$	2	$[-600, 600]^2$	0
f_6 (Six-hump camel-back function)	$4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	2	$[-5, 5]^2$	0.398
f_7 (Goldstien Price function)	$[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$ $\times [30 + (2x_1 - 3x_2)^2 + (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	2	$[-2, 2]^2$	3

mode. The data regarding the failure modes and probability q_{iu} in each system are presented in Appendix B.

6. Experimental design for parameter tuning

It has been an issue of challenge in the EA optimization field to determine the parameter setting that yield efficient performance on the problems at hand. Recently, no free lunch theorem (Wolpert and Macready, 1997) has revealed that the average performance of about all pairs of algorithms is approximately same. This means that if any algorithm performs better in some particular set of problems, it is bound to perform worse in other problems. Also, if an algorithm works well on a particular problem or a class of problems, with a particular parameter setting, it does not guarantee that it would work similarly for other type of problems. Thus, each time, before attempting a new class of problems, parameter tuning is required. The previous sections throw light on the parameter control policies in the form of adaptive mutation schemes. Very often, it is not time efficient and also cumbersome to test all possible combinations of parameters to gain best out of them. Thus, a small but representative sample of combinations is utilized, thanks to the introduction and development of orthogonal design and orthogonal arrays (Phadke, 1989).

In this paper also, the parameter tuning has been performed using various orthogonal arrays. Before getting into the details of the arrays used, let the tuning parameters be detailed first. The basic tuning parameters for various algorithms described in the paper are speed of learning parameters ς and υ; the standard deviation coefficient for adaptive mutations 'μ_i' and population size PopS. Apart from the aforementioned tuning parameters, a noise factor has also been considered as tuning parameter, the formulation of which is described in the following subsection.

Table 4
Range of parameters utilized (distinguished as levels)

Parameter	Level 2	Level 3	Level 4	Level 5
PopS	25	50	75	100
e_{pop}	0.2	0.4	0.6	0.8
ś	0.25	0.50	0.75	1.0
ς	0.2	0.4	0.6	0.8
υ	0.2	0.4	0.6	0.8

In general, real-time optimization problems require the solution evaluation through experimentation, stochastic simulation, sampling, etc. (Nissen and Propach, 1998). Hence, most of the problems involve noise in one way or the other.

Earlier, the effects of additive Gaussian noise have been demonstrated on a set of five test functions (Rana et al., 1996). The conclusions of the study were that after adding noise, some cases exhibited soft annealing effect while most

Table 5
Performance of the proposed EA for problem type **A** (preliminary tuning results)

Experiment number	PopS	e_{pop}	ś	ς	υ	Deviation from optimal (%)					
						f_1 (Sphere function)		f_4 (Ackley,s function)		f_6 (Six-hump camel-back function)	
						Without noise	With noise	Without noise	With noise	Without noise	With noise
1	1	1	1	1	1	1.23 E−10	6.30 E−1	9.39 E−7	1.07 E 0	6.58 E−2	1.02 E−2
2	1	2	2	2	2	2.68 E−11	8.54 E−2	1.05 E−8	6.74 E−1	7.68 E−3	1.02 E 0
3	1	3	3	3	3	2.67 E−11	5.48 E 0	5.28 E−7	5.55 E−2	5.89 E−4	9.98 E−2
4	1	4	4	4	4	9.84 E−11	1.11 E 0	6.34 E−9	4.05 E−1	7.85 E−3	5.64 E−1
5	2	1	2	3	4	4.25 E−13	2.02 E−1	5.67 E−7	3.90 E−2	1.25 E−4	2.55 E−2
6	2	2	1	4	3	7.54 E−12	8.52 E−2	9.78 E−8	7.08 E−2	9.46 E−5	4.58 E−2
7	**2**	**3**	**4**	**1**	**2**	3.33 E−15	4.51 E−3	6.58 E−8	1.59 E−1	**7.08 E−5**	**6.68 E−3**
8	**2**	**4**	**3**	**3**	**1**	**5.63 E−16**	**3.68 E−3**	**1.64 E−9**	**2.47 E−2**	8.08 E−5	5.02 E−3
9	3	1	3	4	2	1.15 E−14	4.51 E−2	1.07 E−9	6.34 E−1	1.11 E−3	9.65 E−2
10	3	2	4	3	1	5.26 E−13	1.02 E−3	2.05 E−7	9.77 E 0	4.83 E−4	1.02 E−1
11	3	3	1	2	4	1.00 E−14	1.11 E−3	5.08 E−8	4.05 E−2	1.05 E−4	6.85 E−3
12	3	4	3	1	3	8.95 E−12	8.58 E−1	3.28 E−8	1.28 E−1	8.16 E−3	5.63 E−2
13	4	1	4	2	3	4.23 E−15	5.64 E−2	1.25 E−7	8.57 E−2	3.11 E−2	3.54 E−1
14	4	2	3	1	4	5.24 E−14	1.25 E−3	5.44 E−9	5.66 E 0	9.63 E−3	4.38 E−1
15	4	3	2	4	1	1.02 E−16	2.58 E 0	2.14 E−8	5.71 E−1	6.59 E−4	9.57 E−3
16	4	4	1	3	3	2.57 E−16	1.09 E−3	8.47 E−8	3.14 E−2	8.47 E−3	7.77 E−2

Table 6
Performance of EA problem type **B** (preliminary tuning results)

Experiment number	PopS	e_{pop}	ś	ς	υ	Deviation from optimal (%)					
						f_1 (Sphere function)		f_4 (Ackley,s function)		f_6 (Six-hump camel-back function)	
						Without noise	With noise	Without noise	With noise	Without noise	With noise
1	1	1	1	1	1	7.68 E−11	2.19 E−2	8.63 E−7	6.74 E−2	1.10 E−3	2.65 E−3
2	1	2	2	2	2	5.55 E−12	5.21 E−3	5.98 E−9	2.53 E 0	1.03 E−4	9.15 E−1
3	1	3	3	3	3	4.68 E−13	6.54 E−1	7.77 E−8	4.19 E−2	4.35 E−3	7.85 E−2
4	1	4	4	4	4	8.97 E−14	7.58 E 0	1.54 E−6	9.39 E−3	1.33 E−2	8.64 E−2
5	2	1	2	3	4	3.68 E−13	6.49 E−3	6.34 E−7	2.87 E−1	4.55 E−3	9.27 E−4
6	2	2	1	4	3	5.68 E−15	7.45 E−4	2.53 E−8	3.80 E−2	3.17 E−5	1.59 E−3
7	**2**	**3**	**4**	**1**	**2**	2.53 E−15	9.51 E−3	**1.01 E−9**	**8.52 E−3**	6.36 E−4	3.19 E−2
8	**2**	**4**	**3**	**3**	**1**	**7.89 E−16**	**8.95 E−4**	6.07 E−9	3.24 E−3	2.95 E−4	5.26 E−3
9	3	1	3	4	2	1.64 E−12	4.45 E−2	6.21 E−9	7.86 E−2	8.00 E−5	1.52 E−3
10	3	2	4	3	1	4.57 E−15	5.95 E−3	3.71 E−8	8.96 E−3	1.04 E−4	9.39 E−2
11	3	3	1	2	4	1.20 E−14	9.03 E−4	1.94 E−9	2.83 E−2	3.74 E−4	1.03 E−3
12	**3**	**4**	**3**	**1**	**3**	3.51 E−14	3.99 E−3	4.90 E−7	5.13 E−1	**5.44 E−5**	**9.10 E−4**
13	4	1	4	2	3	6.01 E−13	7.93 E−2	8.18 E−6	6.47 E−2	2.08 E−4	5.51 E−3
14	4	2	3	1	4	9.07 E−12	7.87 E−1	6.75 E−8	5.12 E−2	7.98 E−3	7.04 E−3
15	4	3	2	4	1	1.05 E−12	5.27 E−2	9.44 E−8	7.77 E−2	6.64 E−4	9.61 E−4
16	4	4	1	3	3	3.66 E−11	1.49 E−2	2.94 E−9	8.75 E−2	1.41 E−3	6.99 E−3

of the cases represented extra optima added to the function. Yet another paper on the similar theme exists (Hammel and Bäck, 1994), which concluded that increasing the amount of noise generally deteriorates the algorithmic performance, whereas, in general, increasing the sample size per individual solution improves performance, as it reduces the amount of uncertainty in the evaluation process.

In the present paper, various parameters discussed above have been robustly tuned by taking into account the noise factors also. The experimental model concerning the impact of noise level has been adopted from Nissen and Propach (1998) and has been briefly stated here. The noisy objective function used in the experimentations is given as

$$F(x_i, \sigma) = f(x_i) + N(0, \sigma), \tag{59}$$

Table 7
Results for algorithm to problem types **C** and **D** (preliminary tuning results)

Experiment number	PopS	e_{pop}	ś	ς	υ	Deviation from optimal (%)			
						Problem type **C** (simulated TSP5)		Problem type **D** ($f_{reliability}$)	
						Without noise	With noise	Without noise	With noise
1	1	1	1	1	1	1.65 E−4	4.10 E−2	−5.54 E−2	−7.80 E−1
2	1	2	2	2	2	5.48 E−4	6.85 E−2	−6.34 E−2	−4.67 E 0
3	1	3	3	3	3	6.52 E−4	1.04 E−2	−5.46 E−3	−5.01 E−2
4	1	4	4	4	4	5.14 E−3	7.58 E−1	−9.16 E−3	−8.40 E−1
5	2	1	2	3	4	9.72 E−5	8.86 E−1	−3.56 E−3	−4.53 E−0
6	2	2	1	4	3	8.52 E−3	6.61 E−2	−4.07 E−3	−6.56 E−1
7	2	3	4	1	2	5.90 E−5	5.78 E−2	−6.15 E−3	−7.86 E−2
8	2	4	3	3	1	4.06 E−4	1.47 E−3	−3.07 E−2	−1.66 E−2
9	3	1	3	4	2	1.25 E−6	5.33 E−2	−8.77 E−3	−5.68 E−1
10	**3**	**2**	**4**	**3**	**1**	3.16 E−5	5.77 E−2	**−9.39 E−4**	**−1.08 E−2**
11	3	3	1	2	4	7.19 E−5	5.90 E−2	−6.10 E−2	−8.45 E−2
12	3	4	3	1	3	1.79 E−6	9.30 E−1	−2.11 E−3	−8.38 E−2
13	**4**	**1**	**4**	**2**	**3**	**2.80 E−6**	**2.70 E−3**	−7.25 E−3	−5.79 E−1
14	4	2	3	1	4	4.50 E−5	7.02 E−2	−6.10 E−2	−1.42 E−2
15	4	3	2	4	1	7.33 E−4	1.60 E−3	−6.74 E−3	−3.62 E−2
16	4	4	1	3	3	5.81 E−5	2.75 E−2	−1.44 E−3	−7.77 E−2

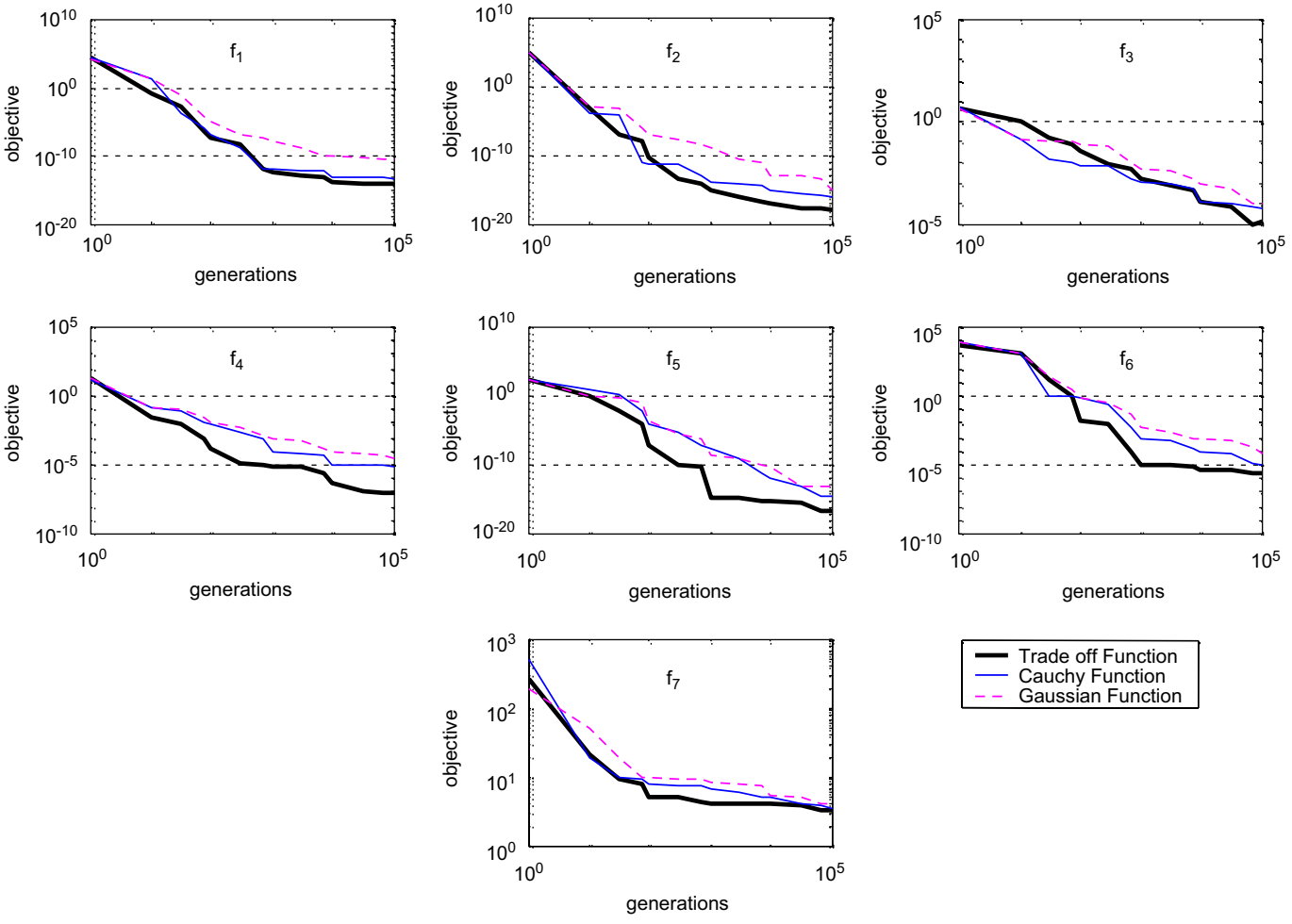

Fig. 7. Comparative results of the proposed tradeoff function-based EAs with the EAs that use Cauchy and Gaussian mutations. Here other parameters and search procedure are kept similar (problem type A).

$$\sigma = \text{ч}\,|\overline{m} - f(x^*)|, \tag{60}$$

$$\bar{m} = \frac{1}{100}\sum_{j=1}^{100} f(x_j);\ \ x_j \text{ is chosen randomly in the search space,} \tag{61}$$

where $N(0,1)$ represents the normally distributed random variable with mean '0'and standard deviation σ; ч is noise constant; x^* is the best known solution for the actual test function; and $\bar{m}$ is the average of the 100 random function values of the test functions (i.e. objective functions). The following discussion presents the actual orthogonal arrays utilized for the robust parameter tuning and provides the description of actual experimental setup.

Since all the four EAs described previously deal with generalized strategies, they encounter similar tuning parameters. All in all, five parameters have been recognized the variation in values those significantly affect the algorithm performance: (a) population size (PopS), (b) elite population (e_{pop}) as the fraction of main population, (c) step size (ś), (d) self-learning rate (ς), and (e) neighborhood learning rate (υ). For the computational study, four levels of all these parameters have been considered and $L_{16}'(4^5)$ orthogonal array (Phadke, 1989) has been utilized. The range of parameters considered for the experimentation according to the $L_{16}'(4^5)$ array are detailed in Table 4. In order to obtain the best set of parameter values, separate experiments have been conducted for all the four problem types. The experiments have been performed for one problem of each kind in the presence of noise as well as in the absence of noise, and the obtained sets of parameters were used for further experiments on similar problems. The results were reported for an average of 10,000 function evaluations in terms of deviation from the optimal value.

After the parameter tuning, the experiments have been performed in order to assess the performance of the proposed generalized strategies over the established strategies used so far in the literature. For the later experiments, two main criteria have been considered: (1) number of generations and (2) average deviation for the whole class of problems. Here, the second criterion gives a glimpse to the average performance of these algorithms for the concerned problem types for which they have been defined. Next section is devoted to the results obtained as per the experimental setup described and a critical analysis of the obtained results is being presented. All the experiments have been done on a 1.8 GHz Pentium 4 Processor and the algorithms have been coded in MATLAB 6.1.

7. Computational results and inferences

In order to start the computational analysis, first the best performing set of parameters has been obtained according to the previously mentioned procedure. The following

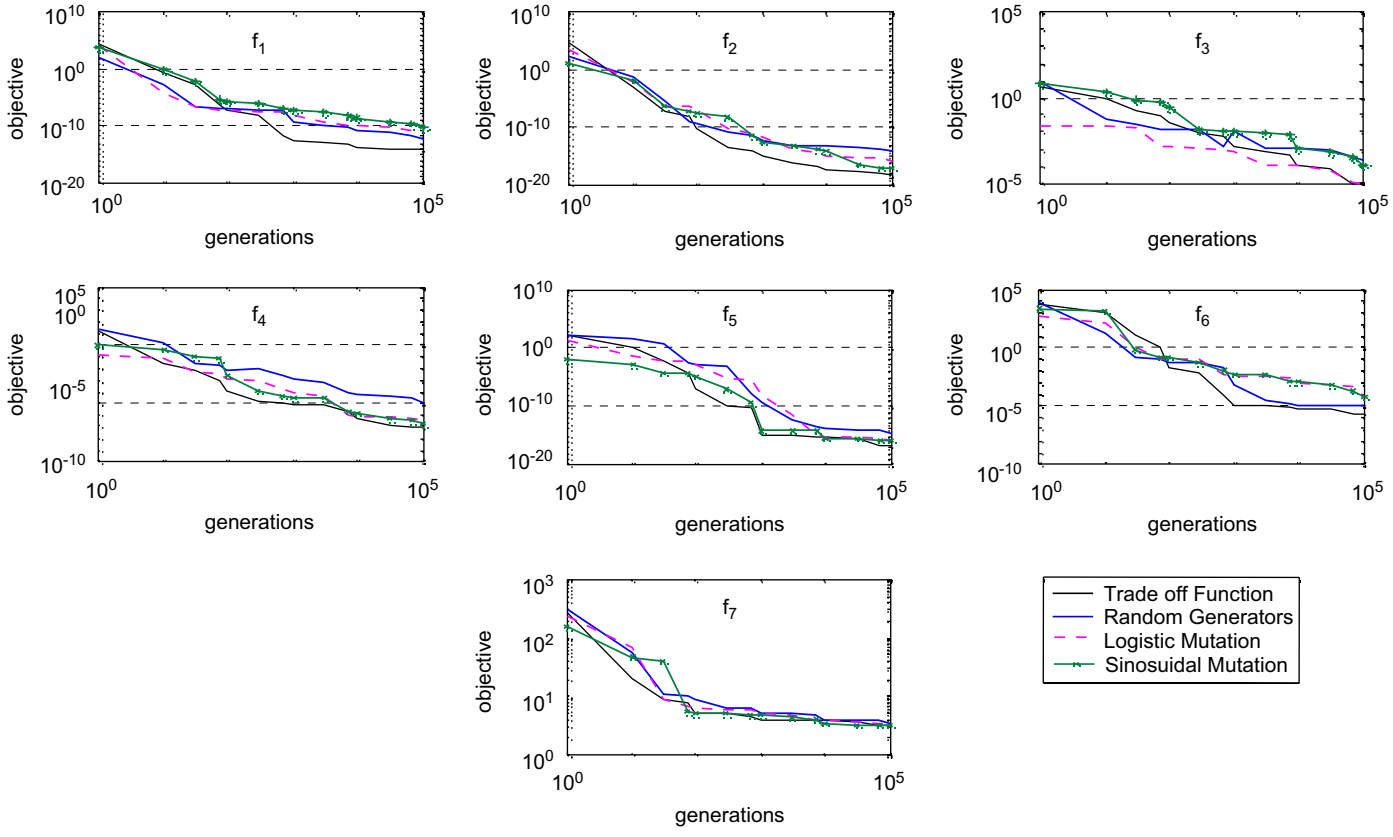

Fig. 8. Comparative results of the proposed tradeoff function-based EAs with the EAs that use random, logistic and sinusoidal mutations. Here other parameters and learning scheme are kept similar (problem type A) (values of both axes are in log).

subsection analyzes various results obtained for parameter tuning.

7.1. Parameter tuning

First, the Problem type **A** has been analyzed. The results obtained as per $L'_{16}(4^5)$ orthogonal array have been reported in Table 5. In this case, one function from each of the three categories has been used and the experimental runs have been performed in the presence of noise as well as in the absence of noise (Eq. (59)). Noise attribute has been utilized in the experiments as the presence of noise simulates the real-time application and thus contributes to establishing the robustness of the algorithm. Here, the value of ч was set to 0.002 for each experimental run as it is found to provide a better estimate of noise (Nissen and Propach, 1998). The results are presented in the form of deviation from optimal value of objective function and is subsequently calculated as

$$\text{Deviation} = \frac{f_{\text{calculated}} - f_{\text{optimal}}}{\overline{m}} \times 100. \quad (62)$$

Similar experiments have also been conducted for the algorithms concerned to the problem types **B**, **C** and **D**. Tables 6 and 7 summarizes the respective results of tuning experiments on these problems. The test conditions have been kept similar to the previous case.

A closer look over the Tables 5–7 suggests that in the first and second case, due to the presence of high dimensional instabilities, the tuning results vary largely, whereas the impact of tuning, although high, is comparatively less visible in the third and fourth case. In these tables, the experiments in bold provide the best parameter settings obtained, for both noisy and deterministic cases, which have been used for the further experiments in the following subsections.

7.2. Comparative performance evaluation

In order to have a comparative evaluation of the proposed generalized strategies with the existing EA strategies found in literature, various set of experiments have been conducted. The results have been presented in three categories:

(i) comparative convergence with the EAs utilizing simple Gaussian mutations, simple Cauchy mutations and those with tradeoff mutations;
(ii) comparative convergence with the EAs using random mutations, Logistic mutations, sinusoidal mutations, and tradeoff function-based mutations;
(iii) best objective functions obtained and the respective deviations from the best known value all the test cases.

Table 8
Best objective values and deviations obtained for the respective test cases

Problem type	Problem	Simple GA		Proposed EAs	
		Best objective value obtained	Deviation	Best objective value obtained	Deviation
A (floating point representation)	f_1	5.69E−15	0.006336	1.02E−17	0.022368
	f_2	1.23E−07	0.012693	1.63E−08	0.155238
	f_3	1.25E−03	0.022007	2.35E−04	0.036776
	f_4	6.96E−06	0.072274	1.58E−09	0.002013
	f_5	5.68E−14	0.006651	4.56E−16	0.004583
	f_6	4.00E−01	0.003419	3.98E−01	0.001055
	f_7	3.01E+00	0.003133	3.00E+00	0.000192
B (binary representation)	f_1	4.57E−15	0.002274	6.34E−16	0.000704
	f_2	6.35E−08	0.007453	1.07E−08	0.005313
	f_3	2.05E−03	0.004824	2.01E−03	0.021606
	f_4	2.05E−07	0.000322	6.69E−09	0.001833
	f_5	5.37E−10	0.260836	5.64E−16	0.000792
	f_6	4.01E−01	0.007474	4.00E−01	0.004666
	f_7	3.17E+00	0.050411	3.06E+00	0.01751
C	Djibouti TSP-89	6.71E+03	0.007044	6.70E+03	0.006138
	Western Sahara TSP-29	2.78E+04	0.005542	2.76E+04	0
	Simulated TSP1	1.79E+02	0	1.79E+02	0
	Simulated TSP2	3.00E+02	0	3.00E+02	0
	Simulated TSP3	2.22E+02	0	2.22E+02	0
	Simulated TSP4	1.66E+02	0.015212	1.63E+02	0
	Simulated TSP5	307.569	0.011318	3.06E+02	0.005952
D	$f_{\text{reliability}}$	0.660685	0	6.61E−01	0

Here, it is important to mention that while any of the experimental run was performed by varying a particular strategy, all other steps and parameters related to other strategies were kept similar.

First the seven functions, characterized as problem type **A** on the basis of their coding strategy, have been targeted by algorithm **A**. The parameters have been set according to the values suggested in Table 5. Fig. 7 presents the comparative convergence of the proposed tradeoff mutations with Gaussian and Cauchy mutations. It is evident from the figure that, in general, the tradeoff mutations with the modified learning strategies are able to give better convergence trends. Although in some cases (pertaining to f_1 and f_3) Cauchy mutation-based strategies have converged earlier, but in terms of solution quality, tradeoff strategy is found to overrule other strategies. In the same vein, Fig. 8 presents the comparative convergence of the tradeoff mutations with random mutations, logistic mutations and sinusoidal mutations. It can be easily assessed from the figure that the tradeoff mutations perform outstandingly better than the other three strategies, their relative convergence being generally affirmative. In order to have a comparison of simple GA with the proposed generalized EAs a comparative analysis for the best objective value obtained along with the concerned deviations has been presented in Table 8. From the table it is quite clear that the proposed strategies are outperforming in almost all the cases. The results of the problems related other types, although presented in the table, have been analyzed later for the sake of maintaining the discussion flow.

Having investigated the supremacy of the generalized rules for problem type **A**, the task remains to investigate the performance over other problem types. Thus, the problem type **B** has been considered next. The problem functions taken in this case are same as those used for type **A**, the difference being the encoding schema utilized by algorithm **B** (Table 2). In this case also the experiments have been performed on the similar line as discussed for type **A**. The results for the comparative performance over simple GA have already been presented in Table 8, whereas, the comparative convergence analysis has been portrayed in Figs. 9. In this case also, the results and trends obtained confirm the claim of faster convergence and improved solution quality as has been theoretically established.

Similar experiments were performed for the type **C** and **D** problems. For type **C**, the respective deviations and the objective function values obtained have been found to be congruent with the optimal results for the smaller problems

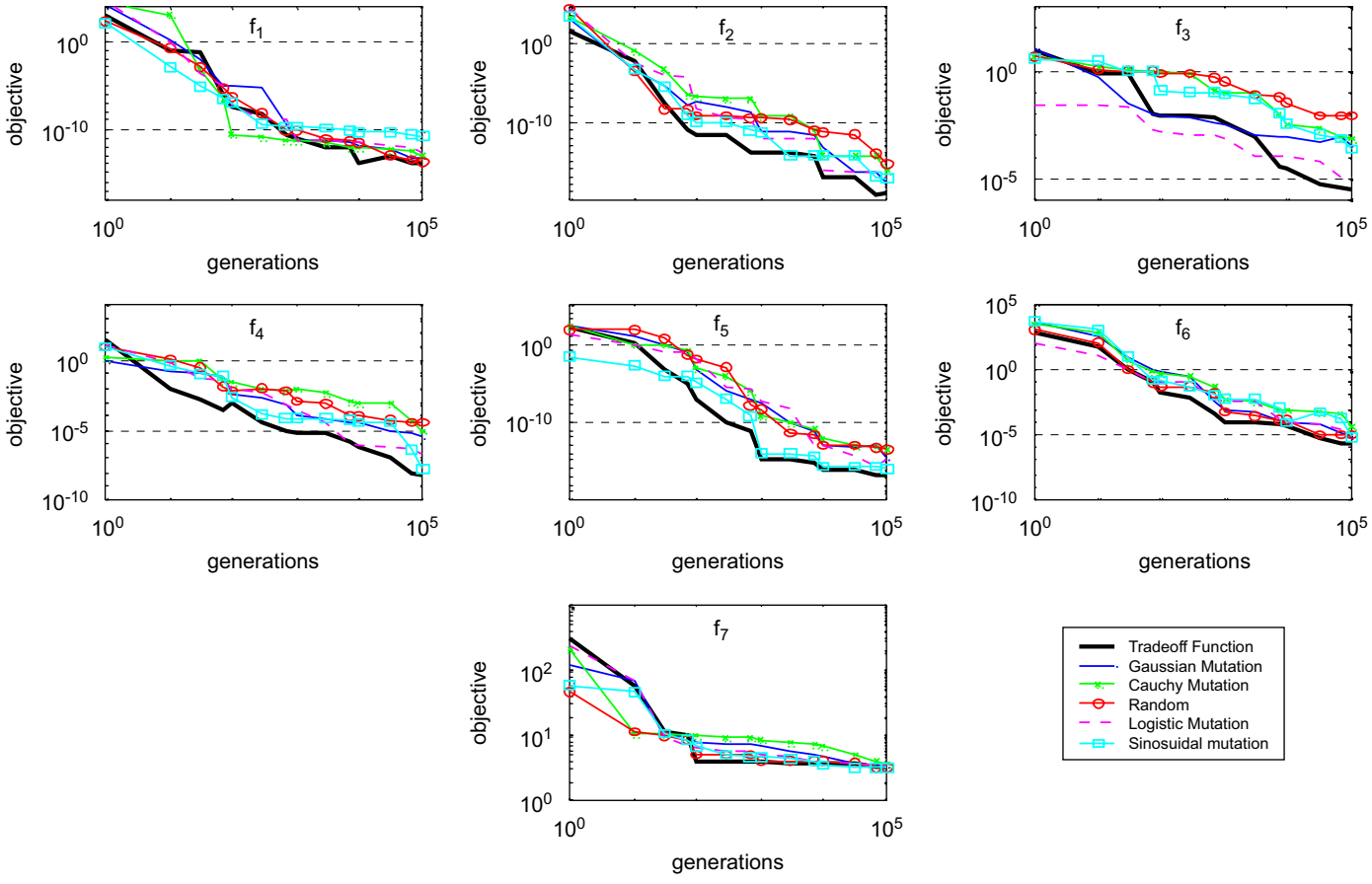

Fig. 9. Comparative results of the proposed tradeoff function-based EAs with the EAs that use Cauchy, Gaussian, random, logistic and sinusoidal mutations. Here other parameters and learning scheme are kept similar (problem type B) (values of both axes are in log).

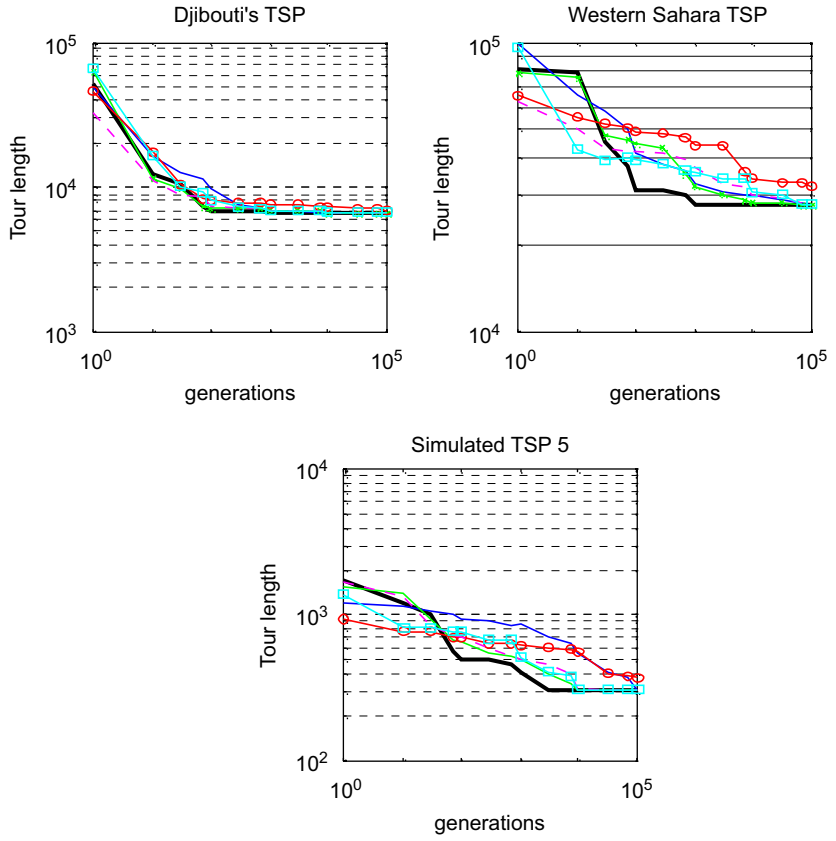

Fig. 10. Comparative results of the proposed tradeoff function-based EAs with the EAs that use Cauchy, Gaussian, random, logistic and sinusoidal mutations. Here other parameters and learning scheme are kept similar (problem type **C**) (values of both axes are in log).

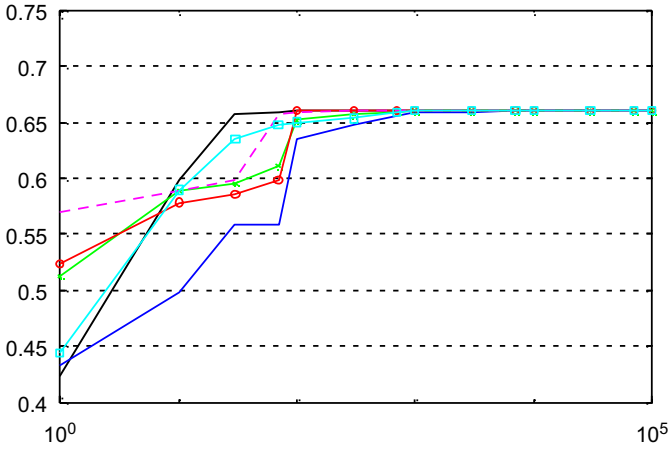

Fig. 11. Comparative results of the proposed tradeoff function-based EAs with the EAs that use Cauchy, Gaussian, random, logistic and sinusoidal mutations. Here other parameters and learning scheme are kept similar (problem type **D**) (values of x-axis are in log; x-axis denoted number of generations and y-axis represents corresponding value of objective function).

while outperforming the results obtained by simple GA for the larger problems. Also the fast convergence is visualized in almost all the cases studied. Fig. 10 portrays the relative convergence trend of the proposed EA with various established strategies. It can easily be concluded from the figure that the tradeoff function provide better results as compared to other strategies, although in order to tackle larger problem of this type, some further investigations are needed.

In the case of problem type **D,** the benchmark problem from Gen and Cheng (1999) has been studied for the convergence analysis. Fig. 11 shows the relative convergence trend for the problem with various strategies. The figure reconfirms the supremacy of the proposed EA. Further applications and performance of the algorithm over different real-time problems coming under type **D** are under the testing phase and the supremacy of the algorithm for much larger sized problems of this type are yet to be proved and should be the focus of upcoming research.

8. Concluding remarks and future scope

This paper targets the real-time optimization from a distinctive perspective. The generalized optimization problems have been first characterized as four basic type problems on the basis of varying solution representations utilized. Further, some generalized and enhanced learning rules have been defined for the EAs. In addition, a new tradeoff function-based mutation has been introduced that ensures more explorative and thorough search. Based on these modifications in the general EAs, four new algorithms, one for each problem type, have been proposed. Although these algorithms are different, yet they are marked by the

similarity in the use of same tradeoff-based mutations and generalized learning rules. In addition to this, the proposed learning rule has been theoretically investigated for convergence and has been proved to have better convergence. A theoretical analysis over the need and explorative strategy of tradeoff function has also been performed. It validates the use and advantages of new tradeoff function instead of single Gaussian or Cauchy functions.

Talking about the experimentations, an intensive study has been done. To start the experiments, first parameter tuning has been performed. Five parameters have been recognized to be of importance. Four levels of values of each parameter have been considered and the tuning has been done using the concept of orthogonal arrays. This significantly eases the tedious tack of tuning and can be relied upon for the considerably good results. The utilization of noise-based parameter tuning ensures the emergence of robust set of performing parameters. In the later sections of the experiments, the algorithms have been tested for their relative convergence with various strategies along with the simple GA. To have a better appraisal, seven functions for type **A** and type **B** problems have been tested. For type **B** problems, five simulated TSPs of varying dimensions and two real-time TSPs have been utilized. A realistic benchmark reliability problem has been considered to investigate the performance for the type **D** problems. All the results have shown the comparative supremacy of the proposed EAs.

However, the only limitation that lies in the paper is concerned with the problems of type **D** as not much problems have been considered under this type. Although, the performance of the proposed strategies has been established over a benchmark problem of type **D**, it is yet to be proved over other problems of the same class. The authors are under the testing phase of the application of these generalized EAs over a wide range of such problems, most of which comprise of real-time manufacturing applications.

Having completed the preliminary tests of the proposed strategies, a clear and more exhaustive future research is ahead. These algorithms have to be tested and suitably modified for the applications that are real challenges to the optimization. More thorough research is needed to study the various other critical aspects of the proposed strategies. Presently authors are working over the extensive study of algorithm applications over various problems of type **D**. A parallel work concerning the application of algorithms for type **C** problems related to controllers in FMS is under research. Preliminary results over the mixed class problems, like those in Tiwari et al. (2005), have also been encouraging.

In nutshell, the present paper is a small step towards the bigger aim of generalization of the diverse field of optimization. Though this paper presents the preliminary results, the performance of the proposed strategies is encouraging and supporting the notion of optimizing the most crucial real-time problems.

Appendix A

See Tables A1–A3.

Appendix B

See Table B1.

Table A1
Djibouti TSP-89 cities (points as (x,y) and distance taken is Euclidean)

(Cities 1–30)		(Cities 31–60)		(Cities 61–89)	
x	y	x	y	x	y
11511.3889	42106.3889	11963.0556	43290.5556	11569.4444	136.6667
11503.0556	42855.2778	11416.6667	42983.3333	11155.8333	42712.5000
11438.3333	42057.2222	11416.6667	42983.3333	11155.8333	42712.5000
11438.3333	42057.2222	11595.0000	43148.0556	11155.8333	42712.5000
11438.3333	42057.2222	12149.4444	42477.5000	11155.8333	42712.5000
11785.2778	42884.4444	11595.0000	43148.0556	11133.3333	42885.8333
11785.2778	42884.4444	11595.0000	43148.0556	11133.3333	42885.8333
11785.2778	42884.4444	11108.6111	42373.8889	11133.3333	42885.8333
11785.2778	42884.4444	11108.6111	42373.8889	11133.3333	42885.8333
12363.3333	43189.1667	11108.6111	42373.8889	11133.3333	42885.8333
11846.9444	42660.5556	11108.6111	42373.8889	11003.6111	42102.5000
11503.0556	42855.2778	11183.3333	42933.3333	11770.2778	42651.9444
11963.0556	43290.5556	12372.7778	42711.3889	11133.3333	42885.8333
11963.0556	43290.5556	11583.3333	43150.0000	11690.5556	42686.6667
12300.0000	42433.3333	11583.3333	43150.0000	11690.5556	42686.6667
11973.0556	43026.1111	11583.3333	43150.0000	11751.1111	42814.4444
11973.0556	43026.1111	11583.3333	43150.0000	12645.0000	42973.3333
11461.1111	43252.7778	11583.3333	43150.0000	12421.6667	42895.5556
11461.1111	43252.7778	11822.7778	42673.6111	12421.6667	42895.5556
11461.1111	43252.7778	11822.7778	42673.6111	11485.5556	187.2222
11461.1111	43252.7778	12058.3333	42195.5556	11423.8889	000.2778
11600.0000	43150.0000	11003.6111	42102.5000	11423.8889	000.2778

Table A1 (*continued*)

(Cities 1–30)		(Cities 31–60)		(Cities 61–89)	
x	*y*	*x*	*y*	*x*	*y*
12386.6667	43334.7222	11003.6111	42102.5000	11715.8333	41836.1111
12386.6667	43334.7222	11003.6111	42102.5000	11297.5000	42853.3333
11595.0000	43148.0556	11522.2222	42841.9444	11297.5000	42853.3333
11595.0000	43148.0556	12386.6667	43334.7222	11583.3333	43150.0000
11569.4444	43136.6667	12386.6667	43334.7222	11569.4444	43136.6667
11310.2778	42929.4444	12386.6667	43334.7222	12286.9444	43355.5556
11310.2778	42929.4444	11569.4444	43136.6667	12355.8333	43156.3889
11310.2778	42929.4444	11569.4444	43136.6667		

Table A2
Five simulated TSP problems

Test problem number	Number of cities	*x*-coordinate	*y*-coordinate
1	4	82.1407	44.4703
		61.5432	79.1937
		92.1813	73.8207
		17.6266	40.5706
2	8	93.5470	91.6904
		41.0270	89.3650
		5.7891	35.2868
		81.3166	0.9861
		13.8891	20.2765
		19.8722	60.3792
		27.2188	19.8814
		1.5274	74.6786
3	10	44.5096	93.1815
		46.5994	41.8649
		84.6221	52.5152
		20.2647	67.2137
		83.8118	1.9640
		68.1277	37.9481
		83.1796	50.2813
		70.9471	42.8892
		30.4617	18.9654
		19.3431	68.2223
4	12	30.2764	54.1674
		15.0873	69.7898
		37.8373	86.0012
		85.3655	59.3563
		49.6552	89.9769
		82.1629	64.4910
		81.7974	66.0228
		34.1971	28.9726
		34.1194	53.4079
		72.7113	30.9290
		83.8496	56.8072
		37.0414	70.2740
5	14	54.6571	44.4880
		69.4567	62.1310
		79.4821	95.6843
		52.2590	88.0142
		17.2956	97.9747
		27.1447	25.2329
		87.5742	73.7306
		13.6519	1.1757
		89.3898	68.45328
		29.8723	66.1443
		28.4409	46.9224
		6.4781	98.8335
		58.2792	42.3496
		51.5512	33.3951

Table A3
Western Sahara TSP-29 cities (points as (x, y) and distance taken is Euclidean)

Cities 1–15		Cities 16–29	
x	*y*	*x*	*y*
20833.3333	17100.0000	26150.0000	10550.0000
20900.0000	17066.6667	26283.3333	12766.6667
21300.0000	13016.6667	26433.3333	13433.3333
21600.0000	14150.0000	26550.0000	13850.0000
21600.0000	14966.6667	26733.3333	11683.3333
21600.0000	16500.0000	27026.1111	13051.9444
22183.3333	13133.3333	27096.1111	13415.8333
22583.3333	14300.0000	27153.6111	13203.3333
22683.3333	12716.6667	27166.6667	9833.3333
23616.6667	15866.6667	27233.3333	10450.0000
23700.0000	15933.3333	27233.3333	11783.3333
23883.3333	14533.3333	27266.6667	10383.3333
24166.6667	13250.0000	27433.3333	12400.0000
25149.1667	12365.8333	27462.5000	12992.2222
26133.3333	14500.0000		

Table B1
Failure modes and probabilities in each subsystem (Gen and Cheng, 1999)

Subsystem *i*	Failure modes	Failure probabilities (q_{iu})
1	O	0.01
	A	0.05
	A	0.10
	A	0.18
2	O	0.08
	A	0.02
	A	0.15
	A	0.12
3	O	0.04
	A	0.05
	A	0.20
	A	0.10

References

〈http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html〉.

Agache, M., Oommen, B.J., 2002. Generalized pursuit learning schemes: new families of continuous and discretized learning automata. IEEE Transactions on Systems, Man and Cybernetics—Part B 32 (6), 738–749.

Ahn, Chang Wook, Ramakrishna, 2003. Elitism based compact genetic algorithms. IEEE Transactions on Evolutionary Computation 7 (4).

Burke, E.K., Newall, J.P., 1999. A multistage evolutionary algorithm for the timetable problem. IEEE Transactions on Evolutionary Computation 3 (1), 63–74.

Butz, M.V., Kovacs, T., Lanzi, P.L., Wilson, S.W., 2004. Toward a theory of generalization and learning in XCS. IEEE Transactions on Evolutionary Computation 8 (1), 28–46.

Caponetto, R., Fortuna, L., Fazzino, S., Xibilia, M.G., 2003. Chaotic sequences to improve the performance of evolutionary algorithms. IEEE Transactions on Evolutionary Computation 7 (3), 289–304.

Choi, D.H., Oh, S.Y., 2000. A new mutation rule for evolutionary programming motivated from backpropogation learning. IEEE Transactions on Evolutionary Computation 4 (2), 188–191.

Clerc, M., Kennedy, J., 2002. The particle swarm—explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation 6 (2), 58–73.

Czarn, A., MacNish, C., Vijayan, K., Turlach, B., Gupta, R., 2004. Statistical exploratory analysis of genetic algorithms. IEEE Transactions on Evolutionary Computation 8 (4), 405–421.

Dimopoulos, C., Zalzala, A.M.S., 2000. Recent developments in evolutionary computation for manufacturing optimization: problems, solutions and comparisons. IEEE Transactions on Evolutionary Computation 4 (2), 93–113.

Eiben, A.E., Hinterding, R., Michalewicz, Z., 1999. Parameter control in evolutionary algorithms. IEEE Transactions on Evolutionary Computation 3 (2), 124–141.

Francois, O., 1998. An evolutionary strategy for global minimization and its Markov chain analysis. IEEE Transactions on Evolutionary Computation 2 (3), 77–90.

François, Olivier, Lavergne, Christian, 2001. Design of evolutionary algorithms—a statistical perspective. IEEE Transactions on Evolutionary Computation 5 (2), 129–148.

Gen, M., 1975. Reliability optimization by 0–1 programming for a system with several failure modes. IEEE Transactions on Reliability R-24, 206–210.

Gen, M., Cheng, R., 1999. Genetic Algorithms, first ed. Wiley, New York.

Gen, M., Ida, K., Sasaki, M., Lee, J., 1989. Algorithm for solving large scale 0–1 goal programming and its applications to reliability optimization problem. International Journal of Computers and Industrial Engineering 17, 525–530.

Goldberg, D., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA.

Hammel, U., Bäck, T., 1994. Evolution strategies on noisy functions. How to improve convergence properties,. In: Davidor, Y., Schwefel, H.-P., Ŵanner, R. (Eds.), Proceedings of the PPSN III—Third International Conference on Parallel Problem Solving from Nature. Springer, Berlin, Germany, pp. 159–168.

Harik, G.R., Lobo, F.G., Goldberg, D.E., 1999. The compact genetic algorithm. IEEE Transactions on Evolutionary Computation 3 (4), 287–297.

Howell, M.N., Gordon, T.J., Brandao, F.V., 2002. Genetic learning automata for function optimization. IEEE Transactions on System, Man, and Cybernatics—Part B 32 (6), 804–815.

Hung, S.L., Adeli, H., 1994. A parallel genetic/neural network learning algorithm for MIMD shared memory machines. IEEE Transactions on Neural Networks 5 (6), 900–909.

Hunt, R.A., 1986. Calculus with Analytic Geometry. Harper & Row, New York.

Kazarlis, S.A., Papadakis, S.E., Theocharis, J.B., Petridis, V., 2001. Microgenetic algorithms as generalized hill-climbing operators for GA optimization. IEEE Transactions on Evolutionary Computation 5 (3), 204–217.

Kim, J.H., Myung, H., 1997. Evolutionary programming techniques for constrained optimization problems. IEEE Transactions on Evolutionary Computation 1 (2), 129–140.

Michalewicz, Z., Deb, K., Schmidt, M., Stidsen, T., 2000. Test-case generator for nonlinear continuous parameter optimization techniques. IEEE Transactions on Evolutionary Computation 4 (3), 197–215.

Najim, K., Poznyak, A.S., 1994. Learning Automata: Theory and Applications. Pergamon, New York.

Narendra, K.S., Tathachar, M.A.L., 1989. Learning Automata. Prentice Hall, Englewood Cliffs, NJ.

Nijssen, S., Bäck, T., 2003. An analysis of the behavior of simplified evolutionary algorithms on trap functions. IEEE Transactions on Evolutionary Computation 7 (1), 11–22.

Nissen, V., Propach, J., 1998. On the robustness of population-based versus point-based optimization in the presence of noise. IEEE Transactions on Evolutionary Computation 2 (3), 107–119.

Ong, Yew Soon, Keane, A.J., 2004. Meta-Lamarckian learning in memetic algorithms. IEEE Transactions on Evolutionary Computation 8 (2), 99–110.

Papadimitriou, G.I., 1994. Hierarchical discretized pursuit nonlinear learning automata with rapid convergence and high accuracy. IEEE Transactions on Knowledge Data Engineering 6, 654–659.

Phadke, S.M., 1989. Quality Engineering Using Robust Design. Prentice Hall, Englewood Cliffs, NJ.

Rana, S., Whitley, D., Cogswell, R., 1996. Searching in the presence of noise. In: Voigt, H.-M., Ebeling, W., Rechenberg, I., Schwefel, H.-P. (Eds.), Proceedings of PPSN IV—Fourth International Conference on Parallel Problem Solving from Nature. Springer, Berlin, Germany, pp. 198–207.

Runarsson, T.P., Yao, X., 2000. Stochastic ranking for constrained evolutionary optimization. IEEE Transactions on Evolutionary Computation 4 (3), 284–294.

Salomon, R., 1998. Evolutionary algorithms and gradient search: similarities and differences. IEEE Transactions on Evolutionary Computation 2 (2), 45–55.

Sinha, N., Chakrabarti, R., Chattopadhyay, P.K., 2003. Evolutionary programming techniques for economic load dispatch. IEEE Transactions on Evolutionary Computation 7 (1), 83–94.

Storn, R., 1999. System design by constraint adaptation and differential evolution. IEEE Transactions on Evolutionary Computation 3 (1), 22–34.

Tathachar, M.A.L., Sastry, P.S., 1986. Estimator algorithms for learning automata. In: Proceedings of the Platinum Jubilee Conference on system Signal Processing, Department of Electrical Engineering, Indian Institute of Science, Bangalore, India, December 1986.

Tiwari, M.K., Kumar, S., Kumar, S., Prakash, A., Shankar, R., 2005. Solving part type selection and operation allocation problems in an FMS: an approach using constraints based fast simulated annealing algorithm. IEEE Transactions on System Man and Cybernetics, Part A 36 (6), 1170–1184.

Tsai, Jinn-Tsong, Liu, Tung-Kuan, Chou, Jyh-Horng, 2004. Hybrid Taguchi-genetic algorithm for global numerical optimization. IEEE Transactions on Evolutionary Computation 8 (4), 365–377.

van den Bergh, F., Engelbrecht, A.P., 2004. A cooperative approach to particle swarm optimization. IEEE Transactions on Evolutionary Computation 8 (3), 225–239.

Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1 (1).

Yao, X., Liu, Y., Lin, G., 1999. Evolutionary programming made faster. IEEE Transactions on Evolutionary Computation 3 (2), 82–102.

Yoon, H.S., Moon, B.R., 2002. An empirical study on the synergy of multiple crossover operators. IEEE Transactions on Evolutionary Computation 6 (2), 212–223.

Zhang, Qingfu, Leung, Yiu-Wing, 1999. An orthogonal genetic algorithm for multimedia multicast routing. IEEE Transactions on Evolutionary Computation 3 (1), 53–62.