Realization of an Adaptive Memetic Algorithm Using Differential Evolution and Q-Learning: A Case Study in Multirobot Path Planning

Pratyusha Rakshit, Amit Konar, Senior Member, IEEE, Pavel Bhowmik, Indrani Goswami, Swagatam Das, Member, IEEE, Lakhmi C. Jain, and Atulya K. Nagar

Abstract—Memetic algorithms (MAs) are population-based meta-heuristic search algorithms that combine the composite benefits of natural and cultural evolutions. An adaptive MA (AMA) incorporates an adaptive selection of memes (units of cultural transmission) from a meme pool to improve the cultural characteristics of the individual member of a population-based search algorithm. This paper presents a novel approach to design an AMA by utilizing the composite benefits of differential evolution (DE) for global search and Q-learning for local refinement. Four variants of DE, including the currently best self-adaptive DE algorithm, have been used here to study the relative performance of the proposed AMA with respect to runtime, cost function evaluation, and accuracy (offset in cost function from the theoretical optimum after termination of the algorithm). Computer simulations performed on a well-known set of 25 benchmark functions reveal that incorporation of Q-learning in one popular and one outstanding variant of DE makes the corresponding algorithm more efficient in both runtime and accuracy. The performance of the proposed AMA has been studied on a real-time multirobot path-planning problem. Experimental results obtained for both simulation and real frameworks indicate that the proposed algorithm-based path-planning scheme outperforms the real-coded genetic algorithm, particle swarm optimization, and DE, particularly its currently best version with respect two standard metrics defined in the literature.

Index Terms—Adaptive memetic algorithm (MA), differential evolution (DE), multirobot path planning, Q-learning.

I. INTRODUCTION

C

OINED by Dawkins in 1976, the word “meme” refers to the basic unit of cultural transmission or imitation [1]. Memetic algorithms (MAs) are population-based meta-heuristic search algorithms that combine the composite benefits of natural and cultural evolutions. Natural evolution realized by evolutionary algorithm (EA) works on the Darwinian principle of the struggle for existence and aims at determining the global optima in a given search landscape. Traditional EA usually takes an excessively large time to locate a precise enough solution because of its inability to exploit local information. Cultural evolution, on the other hand, is capable of local refinement. MA captures the power of global search by its evolutionary component and the power of local search by its cultural component.

Early research on MA was confined to the manual crafting of dedicated memes for a given problem [2]. A paradigm shift in research to adaptively select a meme from a pool of memes for application to an individual member of the population has been observed during the new millennium. The class of algorithms incorporating the adaptive selection of memes is referred to as adaptive MA (AMA). AMAs “promote both cooperation and competition among various problem-specific memes and favor neighborhood structures containing high quality solutions” [3] to be attained at low computational costs. Usually, the selection of the meme for an individual member of the population is done based on its ability to perform local improvement.

Several variants of AMAs are found in the literature [3]–[5]. The one we would use in this paper is Roulette choice strategy based hyperheuristic AMA [4]. In the Roulette choice strategy, a meme $M_e$ is selected with probability relative to the overall improvement. Given that $g(.)$ is a choice function, then the probability of selection of $M_e$ is $g(M_e)/\sum_{i=1}^{n} g(M_i)$, where $n$ is the total number of memes considered.

The AMA to be proposed requires an evolutionary optimization algorithm for global search and a reinforcement learning algorithm for local refinement. The evolutionary component has been realized here by the differential evolution (DE) algorithm for its proven merits in global optimization [10], [20], [26]. Some of the attractive features of DE, justifying its selection in the design of AMA, include simplicity of its structure, leading to ease of coding, very few control parameters, and faster convergence with respect to other swarm algorithms/EAs. Temporal difference Q-learning (TDQL), on the other hand, has been selected as the reinforcement module in AMA for its wide popularity in real-time learning.

The particular variant of DE (DE/current-to-best/1) used in the proposed AMA has two parameters called scaling factors, which are adaptively selected from a meme pool. It is important to mention here that the scaling factors for all the members
of the population evolving with a DE algorithm should not be equal for best performance. A member with a good fitness should search in the local neighborhood, whereas a poor performing member should participate in the global search. A good member thus should have small scaling factors, whereas worse members should have relatively large scaling factors [10]. This is realized in the paper with the help of TDQL.

The TDQL works on the principle of reward and penalty. It employs a Q-table [6] to store the reward/penalty given to an individual member of the population. Members are assigned suitable values of their scaling factors from a given meme pool before participation in the evolutionary process. After completion of the evolutionary process, members are rewarded based on their fitness, and the reward/penalty given to the member depending on the improvement/deterioration in fitness measures of the trial solution is stored in the Q-table. The process of evolution and Q-table updating thus synergistically helps each other, resulting in an overall improvement in the performance of the AMA.

The AMA algorithm to be proposed takes care of three major issues. First, it employs a Roulette choice function based hyper-heuristic scheme to adaptively select memes (scaling factors) for the individual members before participation in the DE. Second, it evaluates the total reward/penalty to be given to the evolved members based on their immediate reward measured by improvements in fitness because of selection of suitable scaling factors prior to evolution and the future reward to be obtained from the Q-table. This is done by one step of the Q-learning. Third, it stores the total reward/penalty in the Q-table by identifying the right location based on its row and column address. The row address of the Q-table is determined from the rank of the evolved members, computed from their individual fitness. The column address is determined from the selected scaling factors with which the individual participated in evolutionary step in the last iteration.

Performance analysis of the proposed AMA realized with DE and TDQL (hereafter referred to as DE-TDQL) is studied using a set of 25 benchmark functions and compared with classical DE/rand/1, DE/current-to-best/1, and DE/rand/either- or algorithms. Experiments reveal that the proposed realization outperforms other variants of DE both by computational accuracy and runtime. Experiments have also been undertaken to compare the performance of a classical self-adaptive DE (SaDE) [28] algorithm with its extended version realized with the proposed adaptation mechanism of the scaling factors. The results confirm that the extended SaDE, referred to as SaDE-TDQL, outperforms the classical SaDE with respect to most (21 out of 25) of the benchmark functions used in the study.

A case study is undertaken here to compare the relative performance of the proposed AMA with some of the popular competitive algorithms. The case study in the present context refers to the online trajectory planning of mobile robots from given starting positions to fixed goal positions without hitting teammates and obstacles [25], [47]. There exist two alternative approaches, i.e., centralized and distributed, to handle the problem. In a centralized approach, the next position of all the robots is determined from their current positions by minimizing an objective function concerning the total path of traversal by the robots, satisfying the necessary constraints on collision avoidance of individual robots with teammates and obstacles. The foregoing problem is solved by iteratively identifying the next positions of the robots until the goal position for all the robots is reached.

In the distributed approach, the objective function with all the necessary constraints for the centralized problem is divided into \( n \) objective functions for \( n \) robots, where the \( i \)th objective function refers to the distance objective and collision avoidance constraints for the \( i \)th robot. The minimization problem in the centralized approach usually has a high degree of computational overhead, which in a distributed environment boils down to a relatively simplified problem of minimization of \( n \) objective functions for \( n \) robots. The dynamic changes in the current and next positions for each robot are taken care of in an iterative manner as in case of the centralized approach.

There exists extensive literature on path planning by single robotic agent employing graphs [48]–[50], neural nets [51], [52], fuzzy logic [53], [54], and EAs [44], [55], [56]. Recently, researchers have taken keen interest to consider multirobot path-planning problems for their possible future applications in factory environment (for transportation of raw materials and (partially) finished products from selected source stations to fixed destinations), defense and security systems, and patient-carrying systems in hospitals/airports. Relatively fewer research works have been undertaken so far in multirobot path planning in comparison to its single-robot counterpart. The existing literature on multirobot path planning employs graphs [62], Voronoi diagrams [63], and potential field techniques [64] to handle the problem. Only in the last few years could traces of EAs in a multirobot environment [24], [58], [66], [67] be detected. The evolutionary multirobot path planners found in the literature usually consider point-mass robots [57], [58] and thus are not amenable for real-world applications. Although the shape and size of robots are considered in a few evolutionary path-planning systems, full merits of the works cannot be judged as they were tested in simulation environments [66], [68], [69] only.

This paper considers an evolutionary path planning with robots of definite size and circular cross section and tested in both simulation and real environments. The workspace here is partitioned into square grids of equal size. The starting and goal positions of each robot on the grid map are given, and the proposed AMA is used to locally plan the trajectory of motion of the robots with an aim to minimize the total path traversed by the robots without collision with obstacles. Performance metrics used in the existing literature [24] have been used here to compare the relative merits of the proposed AMA with respect to the genetic algorithm (GA)-based realization given in [47]. Experiments undertaken further to compare the relative performance of the AMA-based path planner with other swarm/EA-based design reveal that the proposed AMA-based planner outperforms other realizations designed with particle swarm optimization (PSO), DE/current-to-best/1, and SaDE.

This paper is divided into eight sections. Section II provides an overview of the classical Q-learning. Section III introduces the DE algorithm. In Section IV, we propose the AMA realized with DE and TDQL. Section V reports the results of the
II. OVERVIEW OF THE CLASSICAL Q-LEARNING

Learning helps an agent perform better in similar situations. Q-learning falls under the class of reinforcement learning algorithms. In reinforcement learning, the learner performs an action causing a state transition in the environment it resides and receives a reward (or penalty) for the action performed to reach a definite goal. The task of the agent here is to learn a control policy to select an action (from a set of possible actions) at a given state $s$ to maximize the expected sum of the rewards for a sequence of state transitions originating at $s$ and leading to the goal. In the evaluation of the expected sum of rewards, the future rewards are discounted exponentially by their delay [59], and the discounted rewards are added to the “immediate reward” obtained for the current state transition from $s$ to $s’$. Predicting the future rewards at state $s’$, when the agent is at state $s$, is not easy. In Q-learning, we formulate the expected future reward by considering only the discounted next reward for the best action selection at $s’$. However, because of a recursive definition of $Q$ (total-reward) in Q-learning, the above formulation of expected future reward iteratively approximates the actual reward [7] obtained by summing up all the discounted future rewards with the immediate reward.

Let

$$S = \{s_1, s_2, \ldots, s_n\}$$

be a set of states of an agent in a given environment;

$$A = \{a_1, a_2, \ldots, a_n\}$$

be a set of actions that the agent can select in each state $s_i \in S$;

$$r(s_i, a_j)$$

be the immediate reward that the agent acquires by execution of an action $a_j$ at state $s_i$;

$$\delta(s_i, a_j)$$

due to selection of action $a_j$ at state $s_i$, i.e., $s_k = \delta(s_i, a_j)$;

$\gamma$ be the discounting factor used to penalize the future reward after a delay of $k$ units by scaling it by a factor $\gamma^k$ for positive integer $k$ (usually, $\gamma$ lies in [0,1]);

$$Q(s_i, a_j)$$

be the total reward that the agent receives by executing action $a_j$ at state $s_i$.

In Q-learning, the agent selects its next state from its current state by using a policy. The policy attempts to maximize the cumulative reward that the agent could attain in subsequent state transitions to its next state. Let $V(s)$ be the total cumulative reward that the agent earns at state $s$. This cumulative reward at state $s$ is the sum of the discounted immediate rewards obtained by selection of best action at each state transition, starting at state $s$. In Q-learning, $V(s)$ is approximated as $\max_{a'} Q(s, a')$. Thus, $Q(s, a)$ evaluation becomes simplified by

$$Q(s, a) = r(s, a) + \gamma V Q(\delta(s, a))$$

(by definition)

$$= r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a').$$

The classical Q-learning for deterministic state transition is given below. Here, the algorithm begins with a randomly selected initial state. A randomly selected action $a \in A$ enables the agent to receive an immediate reward $r$ and moves to the new state using the $\delta$ transition rule provided in a table. The $Q$ value of the previous state $s$ due to the selected action $a$ is updated in a 2-D Q-table using (1). Now, the next state $s’ = \delta(s, a)$ is considered as the initial state, and the steps of action selection, receiving immediate reward, transition to next state, and Q-table updating are repeated forever.

### Pseudocode of Deterministic Q-Learning

For each state $s$ and action $a$

- Initialize $Q(s, a) \leftarrow 0$;
- Observe the current state $s$;

End For;

Repeat

- Select $a \in A = \{a_1, a_2, \ldots, a_n\}$ randomly and execute it;
- Receive an immediate reward $r(s, a)$;
- Observe the new state $s’ = \delta(s, a)$;
- Update the table entry $Q(s, a)$ by

$$Q(s, a) \leftarrow r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a’);$$

$$s \leftarrow s’;$$

For Ever.

Differential Q-learning is a modified version of Q-learning. The Q-table update policy in differential Q-learning is different from classical Q-learning. It has the ability to remember the effect of past $Q$ value of a particular state–action pair while updating the corresponding $Q$ value. The modified $Q$ update equation is given by

$$Q(s, a) \leftarrow (1 - \alpha) \times Q(s, a) + \alpha \times \left( r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a’) \right).$$

The formula has the effect that the $Q$-value $Q(s, a)$ is incremented when the action $a$ leads to a state $\delta(s, a)$, in which there exists an action $a’$ such that the best possible $Q$-value $Q(\delta(s, a), a’)$ in the next time step and the achieved reward $r(s, a)$ is greater than the current value of $Q(s, a)$. This is exactly the desired behavior, because in such a situation the old estimate of $Q(s, a)$ was too pessimistic. The learning rate $\alpha$ determines the extent to which the newly acquired information will override the old information. A setting of $\alpha = 0$ makes the agent stop learning, whereas $\alpha = 1$ would make the agent consider only the most recent information. The discount factor $\gamma$ determines the importance of future rewards. A factor of 0 will make the agent “opportunist” by only considering current rewards, whereas a factor approaching 1 will make it strive for a long-term high reward. If the discount factor is greater than or equal to 1, the $Q$ values may diverge.

III. OVERVIEW OF THE DE ALGORITHM

DE is a population-based meta-heuristic algorithm that has earned wide publicity for its simple structure with few lines
of codes, fewer parameters, and its excellent performance in numerical optimization with respect to speed and accuracy [11], [71]. DE involves a population of NP parameter vectors, which at generation \( G \) is denoted by \( P_G = \{ \vec{X}_1(G), \vec{X}_2(G), \ldots, \vec{X}_{NP}(G) \} \).

1) Initialization: The \( i \)th member \( \vec{X}_i(G) \) for \( i = 1 \) to \( NP \) at generation \( G = 0 \) is selected by uniformly randomizing individuals in the range \([\vec{X}_{\min}, \vec{X}_{\max}]\), where \( \vec{X}_{\min} = \{ x_{\min-1}, x_{\min-2}, \ldots, x_{\min-D} \} \), and \( \vec{X}_{\max} = \{ x_{\max-1}, x_{\max-2}, \ldots, x_{\max-D} \} \), and thus, the \( j \)th component of the \( i \)th member at \( G = 0 \) is given by

\[
x_{i,j}(0) = x_{j-\min} + \text{rand}_{i,j}(0,1) \times (x_{j-\max} - x_{j-\min})
\] (3)

where \( \text{rand}_{i,j}(0,1) \) is a uniformly distributed random number lying between 0 and 1. Initialize the crossover rate \( Cr \) randomly in \([0,1]\). There are other mutation operations, details of which are available in [10] and [42].

2) Mutation: A donor vector \( \vec{V}_i(G) \) corresponding to each population member or target vector \( \vec{X}_i(G) \) is created by randomly selecting two other members \( \vec{X}_{\text{rand-1}}(G) \) and \( \vec{X}_{\text{rand-2}}(G) \) from the current population \( P_G \), where

\[
\vec{V}_i(G) = \vec{X}_i^+(G) + F_1(X_{\text{best}}^-(G) - \vec{X}_i^-(G))
\]

\[
+ F_2'(X_{\text{rand-1}}(G) - X_{\text{rand-2}}(G))
\] (4)

and \( F_1' \) and \( F_2' \) are two scaling factors in \([0,2]\). The mutation operation given in (4) is referred to as DE/current-to-best/1. This is done for \( i = 1 \) to \( NP \). There are other mutation operations, details of which are available in [10] and [42].

3) Crossover: There are two types of crossover (recombination) schemes, i.e., binomial and exponential [10], [11]. A brief outline of both binomial and exponential crossovers is presented here, which would be used in the AMA algorithm of Section IV.

In case of binomial crossover, a trial vector \( \vec{U}_i(G) \) is generated for each pair of donor vector \( \vec{V}_i(G) \) and target vector \( \vec{X}_i(G) \) by the following operation:

\[
u_{i,j}(G) = \begin{cases} v_{i,j}(G), & \text{if } \text{rand}_{i,j} \leq Cr \text{ or } j = j_{\text{rand}} \\ x_{i,j}(G), & \text{otherwise} \end{cases}
\] (5)

where \( \text{rand}_{i,j}(0,1) \) is a uniformly distributed random number lying in \([0,1]\) and is initiated independently for each \( j \)th component of the \( i \)th vector. \( j_{\text{rand}} \in [1, D] \) is a randomly chosen index, which ensures that \( \vec{U}_i(G) \) gets at least one component from \( \vec{V}_i(G) \).

In case of exponential crossover, an integer \( n \) is randomly selected from \([1, D]\) and is used as a starting point in the target vector to represent the beginning of the crossover or exchange of components with the donor vector. Another integer \( L \) is also selected from \([1, D]\), where \( L \) denotes the number of components; the donor vector contributes to the target. After a choice of \( n \) and \( L \), we obtain the trial vector \( \vec{U}_i(G) \) with

\[
u_{i,j}(G) = \begin{cases} v_{i,j}(G), & \text{for } j = \langle n \rangle_D, \langle n+1 \rangle_D, \ldots, \langle n+L-1 \rangle_D \\ x_{i,j}(G), & \text{for all other } j \in [1, D] \end{cases}
\] (6)

where the angular brackets \( \langle \rangle_D \) denote a modulo function with modulus \( D \). The selection of integer \( L \) from \([1, D]\) is determined by the following pseudocode:

\[
\begin{align*}
L &= 0; \\
\text{DO} & \\
& \quad \{ \\
& \quad \quad L = L + 1; \\
& \quad \} \text{ WHILE } ((\text{rand} (0, 1) < Cr) \text{ AND } (L < D)).
\end{align*}
\]

4) Selection: For a given objective \( f(\vec{x}) \) to be minimized, the selection operator is described as

\[
\vec{X}_i(G + 1) = \vec{U}_i(G), \quad \text{if } f(\vec{U}_i(G)) \leq f(\vec{X}_i(G))
\]

\[
= \vec{X}_i(G), \quad \text{if } f(\vec{U}_i(G)) > f(\vec{X}_i(G)).
\] (7)

Steps 2–4 are repeated until a stopping criterion is reached.

IV. PROPOSED AMA

The AMA to be proposed shortly includes DE for global exploration and TDQL for adaptive selection of memes. These two modules work in a synergistic manner to improve the quality of solutions for a given optimization problem. After each evolutionary step, the performance of the members is evaluated based on their fitness. High performing members are rewarded with positive immediate reward, whereas low performing members are penalized. The reward/punishment given to a member is stored in the Q-table using the TDQL learning rule. A meme pool of the parameter \( F \) is maintained to select two parameters (scaling Factors) \( F'_1 \) and \( F'_2 \) for the individual members of the DE. This is performed by a hyperheuristic choice-metric-based adaptive selection from the meme pool. The process of adaptive selection of \( F'_1 \) and \( F'_2 \) from the meme pool, followed by one step of DE and reward/punishment updating in the Q-table, is continued until the condition for convergence of the AMA is satisfied. In this paper, \( F'_1 \) and \( F'_2 \) are set equal to \( F \) to save complexity and avoid maintenance of two Q-tables for each individual scaling factor.

The proposed AMA algorithm accesses the Q-table to select the scaling factors of the individual members before evolution and updates the Q-table after one evolution. The row indices of the Q-table represent states \( S_1, S_2, \ldots, S_{NP} \) of the population obtained from the last iteration of the DE algorithm. A fitness function-based rank evaluation of individual members is used to allocate the member to a specific state. Thus, state \( S_k \) includes a member of rank \( k \). The column indices of the Q-table correspond to uniformly quantized values of the scaling factors to be used in the EA. Letting the parameter under consideration be \( F \) with possible quantized values \( F_1, F_2, \ldots, F_{10} \) then \( Q(S_i, 10F_j) \) represents the total reward given to a member at state \( S_i \) for selecting \( F = F_j \). The Roulette choice strategy is used to select a particular value of \( F \) from the meme pool \( \{F_1, F_2, \ldots, F_{10}\} \) using \( Q(S_i, 10F_j) \), \( j = 1, 2, \ldots, 10 \), for the
individual member located at state $S_i$. It must be noted that the factor 10 is used to get integer index of $Q(.,.)$.

The adaptation of $Q(S_i, 10F_j)$ is done through a reward/penalty mechanism as used in classical TDQL. If a member of the population, residing at state $S_i$ on selecting $F = F_j$, moves to a new state $S_k$ by the EA, and such state transition causes an improvement in fitness measure, then $Q(S_i, 10F_j)$ is given a positive reward following the TDQL algorithm. If the state transition results in no improvement in fitness measure, then a penalty is given to the selected $Q(S_i, 10F_j)$. The penalty is introduced by a decrease in $Q$-value. Principles used in designing the AMA are introduced below.

5) Initialization: DE-TDQL starts with a population of $NP$ D-dimensional parameter vectors representing the candidate solutions within the prescribed minimum and maximum bounds

\[
\begin{align*}
\bar{x}_{\min} &= [x_{1\min}, x_{2\min}, \ldots, x_{D\min}] \\
\bar{x}_{\max} &= [x_{1\max}, x_{2\max}, \ldots, x_{D\max}].
\end{align*}
\]

Hence, we may initialize the $j$th component of the $i$th vector at generation $G = 0$ as

\[
x_{i,j}(0) = x_{j\min} + \text{rand}_{i,j}(0, 1) \times (x_{j\max} - x_{j\min}),
\]

where $\text{rand}_{i,j}(0, 1)$ is a uniformly distributed random number lying between 0 and 1. The entries for the Q-table are initialized as small values. If the maximum $Q$-value attainable is 100, then we initialize the $Q$ values of all cells in the Q-table as 1.

6) Adaptive Selection of Parameters of the DE: The reward/penalty-based adaptation of the Q-table helps in the right selection of meme $F$ for the members of the population. For example, a member at state $S_i$ has a high probability of selecting $F = F_j$ if $Q(S_i, 10F_j)$ is the largest among $Q(S_i, 10F_l)$ for $l = 1, 2, \ldots, 10$. It is apparent that if $Q(S_i, 10F_j) > Q(S_i, 10F_l)$, for all $l$, then selection of $F = F_j$ at state $S_i$ by the member was rewarded many times before in the evolution process. Naturally, the learning experience will guide the member to select $F = F_j$ with a high probability when the member is at state $S_i$. The probability of selection of $F = F_j$ from the meme pool $\{F_1, F_2, \ldots, F_{10}\}$ is given by

\[
P(F_j) = \frac{Q(S_i, 10F_j)}{\sum_{l=1}^{10} Q(S_i, 10F_l)}. \tag{9}
\]

To maintain adaptation and learning in all $Q$ present in each row, we select a particular $F$ from the meme pool by random selection. This random selection is realized by generating a random number $r$ between (0, 1), and then we determine $F_j$, such that the cumulative probability of $F = F_1$ through $F_{j-1}$ is less than a randomly generated number $r$, and the cumulative probability for $F = F_1$ through $F = F_j$ is greater than $r$. Symbolically, we need to satisfy

\[
\sum_{m=1}^{j-1} P(F = F_m) < r \leq \sum_{m=1}^{j} P(F = F_m). \tag{10}
\]

7) DE: The DE/current-to-best/1 algorithm used here employs mutation, recombination, and selection, as introduced in Section III. The basic difference of the current realization is the selection of $F_1^\prime$ and $F_2^\prime$ from the meme pool adaptively by step 2 before invoking the DE process.

8) Ranking of the Members and State Assignment: Let $f_i$ be the fitness of the $i$th member in the last iteration. A ranking policy is designed to compute the normalized fitness $f_i/\sum_{j=1}^{NP} f_j$, $\forall i$, and then sort them in descending order. The $r$th element of the sorted list has rank $r$, and this member is allocated to state $S_r$. This is repeated for all $r = 1$ to $NP$. Let $f_i$ on selection of $F_j$ to move to a new state $S_k$. If the fitness of the member increases due to transition from $S_i$ to $S_k$, then $Q(S_i, F_j)$ will be updated following (12) with a positive reward function reward($S_i, 10F_j$) is increase in fitness of the member, where

\[
Q(S_i, 10F_j) = (1 - \alpha)Q(S_i, 10F_j) + \alpha \left(\text{reward}(S_i, 10F_j) + \gamma \max_{F^\prime} Q(S_k, 10F^\prime)\right) \tag{12}
\]

else $Q(S_i, 10F_j)$ will be evaluated by (12) with a negative reward $= -\text{K}$ of constant value however small.

10) Convergence: After each evolution, steps 2–5 are repeated until one of the following conditions for convergence is satisfied. The conditions include restricting the number of iterations, maintaining error limits, or both, whichever occurs earlier.

---

**Pseudo Code of AMA**

I. Set the generation number $t = 0$ and randomly initialize a population of $NP$ individuals $P_t = \{X_{1}(t), X_{2}(t), \ldots, X_{NP}(t)\}$ with $X_{i}(t) = \{x_{i,1}(t), x_{i,2}(t), \ldots, x_{i,D}(t)\}$ for $i = [1, 2, \ldots, NP]$ and each individual uniformly distributed in the range $[\bar{x}_{\min}, \bar{x}_{\max}]$, where $\bar{x}_{\min} = \{x_{\min-1}, x_{\min-2}, \ldots, x_{\min-D}\}$ and $\bar{x}_{\max} = \{x_{\max-1}, x_{\max-2}, \ldots, x_{\max-D}\}$. Set $\alpha = 0.25$, $\gamma = 0.8$. Evaluate $f(X_{i}(t))$, for target vector $X_{i}(t)$, $i = [1, 2, \ldots, NP]$. Rank each vector according ascending order of cost function $f()$. Let the ranked population be $R(0) = \{r_{1}(0), r_{2}(0), \ldots, r_{NP}(0)\}$, where $r_{i}(0)$ denotes a target vector of rank $i$ in $t$th generation. Initialize $\{Q[r_{i}(0), j]\} = 1, j = [1, 2, \ldots, 10]$, where $r_{i}(0)$ is defined as above for $r_{i} \in [1, NP]$ and $j \in [1, 10]$ denotes the index of uniformly quantized scaling factor $F$, and $t$ denotes $t$th iteration.

II. While stopping criterion is not reached, do

Begin

Initialize $[\text{reward}(r_{i}(t), j)] = 0$, $r_{i} = [1, 2, \ldots, NP]$, $j = [1, 2, \ldots, 10]$. 


For i = 1 to NP do
Begin
II.a. Roulette choice selection:
Randomly select a scaling factor $F_{r_i}$ from 0.1 to 1.0 with an interval of 0.1 such that the probability of selection of a particular $F = F_j$ is $P(j) = Q(r_i(t),j)/\sum_{i=1}^{10} Q(r_i(t),i)$. 
II.b. Mutation:
Generate a donor vector $\vec{V}_i(t) = \{v_{i,1}(t), v_{i,2}(t), \ldots, v_{i,D}(t)\}$ corresponding to the $i$th target vector $\vec{X}_i(t)$ via the following mutation scheme $\vec{V}_i(t) = \vec{X}_i(t) + F(\vec{X}_{best}(t) - \vec{X}_i(t)) + F(\vec{X}_{rand1}(t) - \vec{X}_{rand2}(t))$; where rand1 and rand2 are mutually exclusive integers randomly chosen from the range [1, NP], and all are different from the base index i, and $f\vec{X}_{best}(t)$ is the best individual vector with the best fitness (i.e., lowest cost function value) in the population at generation $t$.
II.c. Crossover:
Generate trial vector $\vec{U}_i(t) = \{u_{i,1}(t), u_{i,2}(t), \ldots, u_{i,D}(t)\}$ for the $i$th target vector $\vec{X}_i(t)$ through binomial or exponential crossover like classical DE as in Section III.
II.d. Selection:
Evaluate the trial vector $\vec{U}_i(t)$ by measuring its cost function $f(\vec{U}_i(t))$.
If $f(\vec{U}_i(t)) < f(\vec{X}_i(t))$ Then do
Begin
reward($r_i(t), 10 \times F_{r_i}$) = $f(\vec{X}_i(t)) - (\vec{U}_i(t))$; $\vec{X}_i(t + 1) = \vec{U}_i(t)$;
If $f(\vec{U}_i(t)) < f(\vec{X}_{best}(t))$ Then do
Begin
$\vec{X}_{best}(t) = \vec{U}_i(t)$;
Evaluate $f(\vec{X}_{best}(t))$ and save it for future.
End;
End If;
End;
Else do
Begin
reward($r_i(t), 10 \times F_{r_i}$) = $-K$; $\vec{X}_i(t + 1) = \vec{X}_i(t)$;
End;
End If;
End;
End For;
II.e. Rank each vector according ascending order of cost function $f(.)$. Let the ranked population be $R(t+1) = [r_1(t+1), r_2(t+1), \ldots, r_{NP}(t+1)]$, where $r_{i}(t+1)$ denotes a target vector of rank i in (t+1)th generation.
II.f. Update Q-table:
For i = 1 to NP do
Begin
For $F_j = 0.1$ to 1.0 do
Begin
If reward($r_i(t), 10 \times F_j$) $\neq 0$ Then $Q(r_i(t), 10 \times F_j) = (1 - \alpha)Q(r_i(t), 10 \times F_j) + \alpha[\text{reward}(r_i(t), 10 \times F_j) + \gamma \max_{F} Q(r_i(t+1), 10 \times F_j)]$;
End If;
End For;
End;
Increase the counter value $t = t + 1$.
End While;

V. Simulation Results

In this section, we compare DE-TDQ with the other four variants of DE, including DE/rand/1, DE/rand/either-or, DE/current-to-best/1 [10], [11], and SaDE [28]. The comparative study focuses on three important aspects of all the competitor algorithms [42]: 1) the quality of the final solution produced by each algorithm, irrespective of the computational time it consumes; 2) the speed of the convergence measured in terms of the number of function evaluations (FEs) required by an algorithm to reach a predefined threshold value of the objective function; and 3) the frequency of hitting the optima (or success rate) measured in terms of the number of runs of an algorithm that converge to a threshold value within a predetermined number of FEs.

A. Benchmark Functions

The most challenging issue in validation of an EA is to identify the right benchmark functions with diverse characteristics of the functions, such as unimodality, multimodality, and particularly location of the optima on the surface. Traditional benchmark functions usually have the global optimum at the center, surrounded by several local optima along the axes. Naturally, these benchmark functions are inadequate to exhaustively test the performance of an optimization algorithm. To overcome the foregoing problem, a set of recommended benchmark functions [48] were proposed in the Congress of Evolutionary Computation (CEC’05) conference. The proposed benchmarks include shifting of the global optimum and rotation of the local optima, thereby incorporating the diversity of the optimization problems in the traditional benchmark functions.

The relative performance of our algorithm is tested with other variants of DE using 25 benchmark functions, recommended in [48], of 10, 30, and 50 dimensions. The list of functions is available in [48] and cannot be given here for lack of space. The experiments are conducted for 25 independent runs. A performance analysis of the variants of DE for different settings of crossover rate for different dimensional problems has been undertaken here with simulation results. The maximum number of fitness evaluation (Max_FEs) is set to 100,000 for 10-D, 300,000 for 30-D, and 500,000 for 50-D. For lack of space, the experimental results for the 50-D problem only with $Cr = 0.9$ are provided in Table I.

B. Initial Population and Method of Initialization

For all the contestant algorithms, we used the same population size, which is 10 times the dimension D of the problem.
To make the comparison fair, the populations for all the DE variants (over all problems tested) were initialized using the same random seeds. Fogel and Beyer [27] have shown that the popularly used symmetric initializations to compare evolutionary computations can give false impressions of relative performance. In many comparative experiments, the initial population is considered to have a uniform distribution about the entire search space, which is usually defined to be symmetric about the origin [42]. In this paper, we have adopted an asymmetric initialization procedure following the work reported in [72].

### C. Comparison of Quality of the Final Solution

The accuracy of different DE variants is judged by allowing each of them run over every benchmark function until the number of FEs exceeds a given upper limit (which was fixed depending on the dimension of the problem). The mean and the standard deviation (within parentheses) of the best-of-run values for 25 independent runs of each of the algorithms are presented in Table I for $D = 50$ and $C_r = 0.9$.

Since all the algorithms start with the same initial population over each problem instance, we used paired t-tests to compare the means of the results produced by the best and the second best algorithms (with respect to their final accuracies). The t-tests are quite popular among researchers in evolutionary computing, and they are fairly robust to violations of a Gaussian distribution with large number of samples, e.g., 25. In the last columns of Table I, we report the statistical significance level of the difference of the means of the best two algorithms.

Note that here “+” indicates that the t value of 49 degrees of freedom is significant at a 0.05 level of significance by two-tailed test, whereas “−” means the difference of means is not statistically significant, and “NA” stands for not applicable, covering cases for which two or more algorithms achieve the best accuracy results. For all the t-tests carried in Table I, the sample size is taken to be 25. The best algorithm is marked in bold.

<table>
<thead>
<tr>
<th>Function No.</th>
<th>DE/rand/1</th>
<th>DE/rand/either-or</th>
<th>DE/current-to-best-1</th>
<th>DE-TDQL</th>
<th>Statistical Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>f01</td>
<td>7.62981e+09 (5.76298e-09)</td>
<td>1.2294e+06 (7.71943e-07)</td>
<td>0.00000e+00 (0.00000e+00)</td>
<td>0.00000e+00 (0.00000e+00)</td>
<td>NA</td>
</tr>
<tr>
<td>f02</td>
<td>1.06345e-02 (8.7392e-03)</td>
<td>1.4631e-06 (7.19554e-07)</td>
<td>0.00000e+00 (0.00000e+00)</td>
<td>0.00000e+00 (0.00000e+00)</td>
<td>NA</td>
</tr>
<tr>
<td>f03</td>
<td>1.46740e+04 (7.5233e+03)</td>
<td>5.9421e-03 (2.37703e-03)</td>
<td>9.8472e-01 (2.94160e-01)</td>
<td>0.00000e+00 (0.00000e+00)</td>
<td>+</td>
</tr>
<tr>
<td>f04</td>
<td>9.38707e-02 (7.25890e-02)</td>
<td>7.9104e-05 (1.53892e-05)</td>
<td>0.00000e+00 (0.00000e+00)</td>
<td>0.00000e+00 (0.00000e+00)</td>
<td>NA</td>
</tr>
<tr>
<td>f05</td>
<td>2.42849e+02 (5.08024e+01)</td>
<td>2.8547e+00 (1.21284e+00)</td>
<td>9.5635e-05 (1.2491e-05)</td>
<td>2.2737e-02 (1.31602e-02)</td>
<td>+</td>
</tr>
<tr>
<td>f06</td>
<td>1.11454e+02 (2.60841e+02)</td>
<td>1.64746e+02 (2.76432e+02)</td>
<td>1.35054e+00 (5.97550e+00)</td>
<td>4.78389e+01 (1.32220e+02)</td>
<td>+</td>
</tr>
<tr>
<td>f13</td>
<td>2.82947e+00 (4.29449e-01)</td>
<td>2.90694e+00 (5.35088e-01)</td>
<td>2.15193e+00 (4.3762e-01)</td>
<td>1.18067e+00 (2.70458e-01)</td>
<td>+</td>
</tr>
<tr>
<td>f15</td>
<td>6.53924e+02 (1.7351e+01)</td>
<td>6.80948e+02 (9.3627e+01)</td>
<td>4.28239e+02 (7.51763e+01)</td>
<td>4.29234e+02 (1.12549e+01)</td>
<td>-</td>
</tr>
<tr>
<td>f16</td>
<td>1.80043e+02 (1.31896e+01)</td>
<td>1.77648e+02 (1.09782e+01)</td>
<td>1.65812e+02 (2.06795e+01)</td>
<td>1.12813e+02 (1.31497e+01)</td>
<td>+</td>
</tr>
<tr>
<td>f17</td>
<td>2.03029e+02 (1.51955e+01)</td>
<td>2.01405e+02 (1.74092e+01)</td>
<td>1.78097e+02 (1.91219e+01)</td>
<td>1.34020e+02 (1.84988e+01)</td>
<td>+</td>
</tr>
<tr>
<td>f18</td>
<td>8.40003e+02 (4.35765e+01)</td>
<td>8.48183e+02 (4.92894e+01)</td>
<td>7.49505e+02 (2.34925e+02)</td>
<td>7.63288e+02 (1.77011e+02)</td>
<td>+</td>
</tr>
<tr>
<td>f19</td>
<td>8.30678e+02 (2.2685e+01)</td>
<td>8.46269e+02 (4.60059e+01)</td>
<td>8.23536e+02 (1.18894e+02)</td>
<td>8.19489e+02 (1.20251e+02)</td>
<td>+</td>
</tr>
<tr>
<td>f20</td>
<td>8.75395e+02 (5.77017e+01)</td>
<td>8.51891e+02 (5.06452e+01)</td>
<td>8.16640e+02 (1.64708e+02)</td>
<td>6.84876e+02 (2.51560e+02)</td>
<td>+</td>
</tr>
<tr>
<td>f21</td>
<td>1.10406e+03 (8.25569e+00)</td>
<td>1.07624e+03 (1.11391e+01)</td>
<td>1.03193e+03 (1.53154e+02)</td>
<td>7.08847e+02 (1.38802e+02)</td>
<td>+</td>
</tr>
<tr>
<td>f22</td>
<td>5.28406e+02 (4.13118e+01)</td>
<td>5.27321e+02 (4.55182e+01)</td>
<td>6.06447e+02 (1.30617e+02)</td>
<td>7.63716e+02 (1.88575e+02)</td>
<td>+</td>
</tr>
<tr>
<td>f23</td>
<td>1.10932e+03 (5.24023e+00)</td>
<td>1.09531e+03 (9.43862e+00)</td>
<td>1.02561e+03 (1.56514e+02)</td>
<td>8.88932e+02 (2.21409e+02)</td>
<td>+</td>
</tr>
<tr>
<td>f24</td>
<td>8.4588e+02 (2.98505e+00)</td>
<td>8.0102e+02 (8.3361e+00)</td>
<td>7.5841e+02 (1.8062e+00)</td>
<td>7.5238e+02 (4.34601e+00)</td>
<td>+</td>
</tr>
<tr>
<td>f25</td>
<td>8.3175e+02 (4.01108e+00)</td>
<td>7.9250e+02 (3.4899e+00)</td>
<td>7.6823e+02 (2.99705e+00)</td>
<td>9.9675e+02 (5.12610e+00)</td>
<td>+</td>
</tr>
</tbody>
</table>

**TABLE I**

**PERFORMANCE OF THE PROPOSED DE-TDQL-BASED AMA WITH OTHER VARIANTS OF DE (FOR $D = 50$ AND $C_r = 0.9$)**
#### TABLE II

(a) No. of Successful Runs out of 25 Runs and Success Performance in Parenthesis (Success Performance = MEAN (FEs for Successful Runs) * (# of Total Runs)/( # of Successful Runs)) for f01–f13. (b) No. of Successful Runs out of 25 Runs and Success Performance in Parenthesis (Success Performance = MEAN (FEs for Successful Runs) * (# of Total Runs)/( # of Successful Runs)) for f14–f25

<table>
<thead>
<tr>
<th>Function No.</th>
<th>Tolerance</th>
<th>DE/rand/1</th>
<th>DE/rand/either-or</th>
<th>DE/current-to-best-1</th>
<th>DE-TDQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>f'01</td>
<td>1.00e-04</td>
<td>25 (1.64190e+004)</td>
<td>25 (2.06650e+004)</td>
<td>25 (4.51000e+003)</td>
<td>25 (2.63200e+003)</td>
</tr>
<tr>
<td>f'02</td>
<td>1.00e-02</td>
<td>13 (2.43519e+004)</td>
<td>25 (1.53580e+004)</td>
<td>25 (5.56200e+003)</td>
<td>25 (2.80000e+003)</td>
</tr>
<tr>
<td>f'03</td>
<td>1.00e+05</td>
<td>25 (1.90340e+004)</td>
<td>25 (6.96700e+003)</td>
<td>25 (2.43000e+003)</td>
<td>25 (1.59600e+003)</td>
</tr>
<tr>
<td>f'04</td>
<td>1.00e+00</td>
<td>25 (1.98130e+004)</td>
<td>25 (1.24390e+004)</td>
<td>25 (4.21900e+003)</td>
<td>25 (2.09400e+003)</td>
</tr>
<tr>
<td>f'05</td>
<td>1.00e+03</td>
<td>25 (1.41600e+004)</td>
<td>25 (6.95300e+003)</td>
<td>25 (2.06400e+003)</td>
<td>25 (7.21000e+002)</td>
</tr>
<tr>
<td>f'06</td>
<td>5.00e+02</td>
<td>24 (1.61698e+004)</td>
<td>22 (1.66045e+004)</td>
<td>25 (2.88100e+003)</td>
<td>25 (1.39100e+003)</td>
</tr>
<tr>
<td>f'07</td>
<td>2.00e+00</td>
<td>25 (7.18900e+003)</td>
<td>25 (6.88300e+003)</td>
<td>25 (1.68000e+003)</td>
<td>25 (9.31000e+002)</td>
</tr>
<tr>
<td>f'08</td>
<td>2.04e+01</td>
<td>15 (1.08550e+004)</td>
<td>18 (1.23597e+004)</td>
<td>18 (9.07500e+003)</td>
<td>25 (8.21800e+003)</td>
</tr>
<tr>
<td>f'09</td>
<td>5.00e+01</td>
<td>25 (4.53800e+003)</td>
<td>25 (9.69300e+003)</td>
<td>25 (1.74600e+003)</td>
<td>25 (5.13000e+002)</td>
</tr>
<tr>
<td>f'10</td>
<td>5.00e+01</td>
<td>25 (9.78600e+003)</td>
<td>25 (9.36200e+003)</td>
<td>25 (2.22500e+003)</td>
<td>25 (8.75000e+002)</td>
</tr>
<tr>
<td>f'11</td>
<td>9.20e+00</td>
<td>19 (9.39079e+003)</td>
<td>13 (7.97308e+003)</td>
<td>17 (9.55294e+003)</td>
<td>25 (6.18400e+003)</td>
</tr>
<tr>
<td>f'12</td>
<td>4.00e+04</td>
<td>25 (4.21100e+003)</td>
<td>25 (7.22700e+003)</td>
<td>25 (5.44900e+003)</td>
<td>25 (2.65300e+003)</td>
</tr>
<tr>
<td>f'13</td>
<td>0.0e+00</td>
<td>16 (1.25391e+004)</td>
<td>13 (1.71327e+004)</td>
<td>25 (7.53300e+003)</td>
<td>25 (2.15400e+003)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function No.</th>
<th>Tolerance</th>
<th>DE/rand/1</th>
<th>DE/rand/either-or</th>
<th>DE/current-to-best-1</th>
<th>DE-TDQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>f'14</td>
<td>4.15e+00</td>
<td>14 (1.30357e+004)</td>
<td>23 (8.75217e+003)</td>
<td>25 (6.40800e+003)</td>
<td>25 (7.38000e+002)</td>
</tr>
<tr>
<td>f'15</td>
<td>7.08e+02</td>
<td>25 (4.74000e+003)</td>
<td>19 (1.01908e+004)</td>
<td>25 (5.70000e+002)</td>
<td>25 (1.66000e+002)</td>
</tr>
<tr>
<td>f'16</td>
<td>2.24e+02</td>
<td>25 (6.45000e+003)</td>
<td>25 (7.20100e+003)</td>
<td>25 (5.88000e+003)</td>
<td>25 (1.28000e+003)</td>
</tr>
<tr>
<td>f'17</td>
<td>2.50e+02</td>
<td>25 (6.91100e+003)</td>
<td>25 (8.22500e+003)</td>
<td>25 (5.89000e+003)</td>
<td>25 (1.28400e+003)</td>
</tr>
<tr>
<td>f'18</td>
<td>8.50e+02</td>
<td>21 (5.96667e+003)</td>
<td>19 (2.53158e+003)</td>
<td>18 (1.01111e+003)</td>
<td>22 (2.36250e+003)</td>
</tr>
<tr>
<td>f'19</td>
<td>8.50e+02</td>
<td>23 (5.43913e+003)</td>
<td>19 (2.86184e+003)</td>
<td>20 (8.66250e+002)</td>
<td>16 (1.54688e+03)</td>
</tr>
<tr>
<td>f'20</td>
<td>8.75e+02</td>
<td>13 (4.81538e+003)</td>
<td>18 (2.19444e+003)</td>
<td>17 (6.85294e+002)</td>
<td>20 (2.17250e+03)</td>
</tr>
<tr>
<td>f'21</td>
<td>1.10e+03</td>
<td>6 (1.96667e+004)</td>
<td>24 (1.77708e+004)</td>
<td>25 (3.81000e+003)</td>
<td>25 (1.14200e+003)</td>
</tr>
<tr>
<td>f'22</td>
<td>8.00e+02</td>
<td>25 (5.45000e+002)</td>
<td>25 (4.46000e+002)</td>
<td>20 (5.85000e+003)</td>
<td>22 (8.60227e+002)</td>
</tr>
<tr>
<td>f'23</td>
<td>1.15e+03</td>
<td>25 (4.69000e+003)</td>
<td>25 (6.51000e+003)</td>
<td>25 (1.02500e+003)</td>
<td>24 (8.35417e+002)</td>
</tr>
<tr>
<td>f'24</td>
<td>4.10e+02</td>
<td>25 (4.54500e+003)</td>
<td>25 (1.00000e+003)</td>
<td>25 (8.86000e+002)</td>
<td>25 (5.40000e+002)</td>
</tr>
<tr>
<td>f'25</td>
<td>4.50e+02</td>
<td>25 (1.64800e+003)</td>
<td>25 (1.49400e+003)</td>
<td>25 (9.40000e+002)</td>
<td>20 (7.3750e+002)</td>
</tr>
</tbody>
</table>

### D. Performance Analysis

To compare the speeds of different algorithms, we select a threshold value of the objective function for each benchmark problem. We run each algorithm on a function and stop as soon as the best fitness value determined by the algorithm falls below the predefined threshold. The number of FEs of the algorithm is recorded. A lower number of FEs corresponds to a faster algorithm. Table II reports the number of runs (out of 25) that managed to find the optimum solution (within the given tolerance) as well as the success performance obtained by the algorithms to converge within the prescribed threshold value.

A close inspection of Table I indicates that the performance of the proposed DE-TDQL algorithm has remained clearly and consistently superior to that of the three classical DE schemes (DE/rand/1, DE/rand/either-or, and DE/current-to-best-1). It is interesting to see that out of 25 benchmark instances, in 21 cases, DE-TDQL outperforms its nearest neighbor competitor in a statistically significant fashion. It may be noted from Table I that apart from a few relatively simpler test functions like Shifted Sphere (f01), Shifted Schwefel’s Problem 1.2 (f02), and Shifted Schwefel’s Problem 1.2 with noise in fitness (f04), most of the algorithms end up...
with almost equal accuracy. Substantial performance differences, however, are noticed for the rest of the more challenging benchmark functions. In three cases (f15, f18, and f25), DE/current-to-best-1 achieved the best average accuracy, beating DE-TDQL, which remained the second best algorithm.

Table II indicates that DE-TDQL yields not only the most accurate results for nearly all the benchmark problems, but the number of runs that converge below a prespecified cutoff value is also greater for DE-TDQL over most of the benchmark problems covered here. This indicates the higher robustness (i.e., the ability to produce similar results over repeated runs on a single problem) of the algorithm as compared to its other three competitors.

The latter part of the experiment attempts to improve the performance of the most popular variant of DE called SaDE [28] by incorporating TDQL in the algorithm for adaptation of scaling factors. SaDE focuses on adaptation for crossover rate and mutation strategies of DE. The motivation in SaDE is to solve the dilemma that crossover rate Cr and mutation strategies involved in DE are often highly problem dependent. SaDE adopts four DE mutation strategies and introduces a probability “p” to determine the right one to use. The probability p is gradually adapted according to its learning experience.

TABLE III

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>D=10</td>
<td>3.02e-27 (2.13e-21)</td>
<td>6.24e-29 (1.65e-10)</td>
<td>+</td>
<td>6.24e-29 (1.65e-10)</td>
<td>+</td>
<td>2.33e-10 (7.50e-07)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f02</td>
<td>1.56e-09 (2.52e-05)</td>
<td>1.66e-05 (1.12e-04)</td>
<td>+</td>
<td>1.66e-05 (1.12e-04)</td>
<td>+</td>
<td>1.59e-02 (5.07e-07)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f03</td>
<td>8.02e-04 (5.36e-04)</td>
<td>4.75e-04 (3.35e-05)</td>
<td>+</td>
<td>4.75e-04 (3.35e-05)</td>
<td>+</td>
<td>4.75e-01 (7.21e-05)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f04</td>
<td>5.40e-03 (3.17e-05)</td>
<td>5.22e-03 (1.92e-05)</td>
<td>+</td>
<td>5.22e-03 (1.92e-05)</td>
<td>+</td>
<td>2.77e-01 (6.85e-02)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f05</td>
<td>6.74e-05 (5.05e-09)</td>
<td>8.00e-03 (0.00e+00)</td>
<td>+</td>
<td>8.00e-03 (0.00e+00)</td>
<td>+</td>
<td>8.00e-03 (3.15e-08)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f06</td>
<td>2.05e+01 (8.69e+06)</td>
<td>3.33e+01 (1.93e+03)</td>
<td>+</td>
<td>3.33e+01 (1.93e+03)</td>
<td>+</td>
<td>5.97e+00 (7.70e+01)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f07</td>
<td>1.25e-03 (0.00e+00)</td>
<td>1.27e-03 (0.00e+00)</td>
<td>+</td>
<td>1.27e-03 (0.00e+00)</td>
<td>NA</td>
<td>1.27e-03 (0.00e+00)</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f08</td>
<td>2.06e+01 (1.41e-17)</td>
<td>2.04e+01 (7.07e-10)</td>
<td>+</td>
<td>2.04e+01 (7.07e-10)</td>
<td>-</td>
<td>2.05e+01 (7.07e-12)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f09</td>
<td>2.71e-10 (0.00e+00)</td>
<td>8.35e-08 (5.37e-08)</td>
<td>+</td>
<td>8.35e-08 (5.37e-08)</td>
<td>+</td>
<td>2.37e-09 (1.56e-11)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f10</td>
<td>2.45e+01 (1.18e-04)</td>
<td>3.38e+01 (3.35e-02)</td>
<td>+</td>
<td>3.38e+01 (3.35e-02)</td>
<td>+</td>
<td>2.88e+00 (4.94e+01)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f11</td>
<td>6.29e+00 (6.64e+13)</td>
<td>8.40e+00 (1.19e+11)</td>
<td>+</td>
<td>8.40e+00 (1.19e+11)</td>
<td>+</td>
<td>6.13e+00 (9.89e+02)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f12</td>
<td>2.45e+03 (4.31e-04)</td>
<td>1.80e+03 (3.74e-04)</td>
<td>-</td>
<td>1.80e+03 (3.74e-04)</td>
<td>+</td>
<td>1.27e+03 (8.76e+02)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f13</td>
<td>6.38e-01 (3.11e-11)</td>
<td>5.64e-01 (1.10e-10)</td>
<td>-</td>
<td>5.64e-01 (1.10e-10)</td>
<td>-</td>
<td>7.20e-01 (5.79e-07)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f14</td>
<td>3.53e+00 (3.14e+00)</td>
<td>2.59e+00 (1.05e+05)</td>
<td>-</td>
<td>2.59e+00 (1.05e+05)</td>
<td>-</td>
<td>4.08e+00 (8.99e+01)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f15</td>
<td>3.32e+00 (4.46e-01)</td>
<td>4.00e+00 (0.00e+00)</td>
<td>+</td>
<td>4.00e+00 (0.00e+00)</td>
<td>+</td>
<td>4.00e+00 (2.61e+00)</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f16</td>
<td>1.32e+02 (4.95e-04)</td>
<td>1.43e+02 (1.06e-04)</td>
<td>+</td>
<td>1.43e+02 (1.06e-04)</td>
<td>+</td>
<td>1.28e+02 (7.07e+01)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f17</td>
<td>1.85e+02 (6.36e-07)</td>
<td>1.73e+02 (2.40e-01)</td>
<td>+</td>
<td>1.73e+02 (2.40e-01)</td>
<td>+</td>
<td>2.07e+02 (2.82e+00)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f18</td>
<td>8.00e+02 (1.63e-04)</td>
<td>3.98e+02 (6.36e-07)</td>
<td>-</td>
<td>3.98e+02 (6.36e-07)</td>
<td>+</td>
<td>3.89e+02 (5.79e+02)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f19</td>
<td>5.31e+02 (2.00e-03)</td>
<td>8.00e+02 (2.54e-02)</td>
<td>+</td>
<td>8.00e+02 (2.54e-02)</td>
<td>+</td>
<td>4.40e+02 (1.06e+00)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f20</td>
<td>8.00e+02 (2.00e+01)</td>
<td>8.00e+02 (0.00e+00)</td>
<td>+</td>
<td>8.00e+02 (0.00e+00)</td>
<td>+</td>
<td>8.00e+02 (4.94e+01)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f21</td>
<td>5.00e+02 (0.00e+00)</td>
<td>5.00e+02 (0.00e+00)</td>
<td>NA</td>
<td>5.00e+02 (0.00e+00)</td>
<td>+</td>
<td>5.00e+02 (0.00e+00)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f22</td>
<td>7.68e+02 (2.40e-00)</td>
<td>7.62e+02 (1.90e+01)</td>
<td>+</td>
<td>7.62e+02 (1.90e+01)</td>
<td>+</td>
<td>7.62e+02 (2.19e+01)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f23</td>
<td>9.29e+02 (1.47e-03)</td>
<td>5.59e+02 (0.00e+00)</td>
<td>+</td>
<td>5.59e+02 (0.00e+00)</td>
<td>+</td>
<td>5.59e+02 (0.00e+00)</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f24</td>
<td>2.00e+02 (0.00e+00)</td>
<td>2.00e+02 (0.00e+00)</td>
<td>NA</td>
<td>2.00e+02 (0.00e+00)</td>
<td>+</td>
<td>2.00e+02 (0.00e+00)</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f25</td>
<td>2.00e+02 (1.13e-07)</td>
<td>2.00e+02 (0.00e+00)</td>
<td>NA</td>
<td>2.00e+02 (0.00e+00)</td>
<td>NA</td>
<td>2.00e+02 (0.00e+00)</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Additionally, the crossover rate $Cr$ is self-adapted by recording $Cr$ values that make trial vectors [10] successfully enter the next generation.

Table III provides a comparative estimate of the relative performance of SaDE extended with TDQL (SaDE-TDQL) for scaling factor adaptation. In Table III, the mean and standard deviation (given within parenthesis) of the cost function of 25 independent runs for each algorithm are presented. To test the statistical significance of the results, we use paired t tests between the two algorithms with a tolerance of 5% and sample size of 25. The best algorithm is marked in bold. It is apparent from Table III that with a setting of learning period $LP = 50$ for $D = 10$, SaDE-TDQL outperforms all the 25 benchmark functions, excluding three functions: $f_{13}$, $f_{14}$, and $f_{20}$. SaDE-TDQL performs consistently better than SaDE over all the 58 benchmark instances out of 75 (considering 25 benchmark
instances for each of three settings of dimensions, $D = 10, 30, 50$, as reported in Table III, and the advantage of SaDE-TDQL is very prominent as well.

To compare the relative speed of convergence and quality of solution (accuracy) of DE-TDQL with five optimization algorithms, namely, DE/current-to-best/1, DE/rand/1, DE/rand/either-or, SaDE, and SaDE-TDQL, the mean value of the objective function (mean best fitness) taken over 25 runs versus FEs is plotted in Fig. 1. The observation from Fig. 1 illustrates that DE-TDQL outperforms all the other variants of DE considered above, whereas SaDE-TDQL outperforms all algorithms including DE-TDQL. The foregoing observation indicates that the incorporation of TDQL for scaling factor adaptation in a given variant of DE always improves its performance in both quality of solution and FEs.

A relative analysis of performance of the six algorithms including DE-TDQL can be obtained from Fig. 2. In Fig. 2(a), we present a plot of accuracy (i.e., the difference between the best cost function obtained after convergence and the cost function at the theoretical optimum [48]) versus number of
FEs, whereas in Fig. 2(b) we provide a plot of accuracy versus runtime complexity. These plots provide a visual means of demonstrating the performance of the algorithms with respect to both accuracy and FEs/runtime. After scaling the x- and y-coordinates (to have a uniformity in order of magnitude), we use the distance of a point from origin as a measure of its performance. The smaller the measure, the better is the performance of the algorithm. We now use the “>” symbol to represent the relative performance of the two algorithms. Using this convention, we note from Fig. 2(a) and (b) that the performance of the six algorithms respectively is

\[
\text{DE-TDQL} > \text{SaDE-TDQL} > \text{SaDE} = \text{DE/rand/either-or} > \text{DE/rand/1}, \text{and}
\]

\[
\text{SaDE-TDQL} > \text{DE-TDQL} > \text{SaDE} = \text{DE/current-to-best/1} > \text{DE/rand/either-or} > \text{DE/rand/1} \text{ respectively.}
\]

Our experience of working with DE-TDQL substantiated by the experimental results given in Fig. 2 indicates that the proposed algorithm in general offers a good level of accuracy at lower computational cost, measured by FEs and runtime complexity. It is thus apparent from the above inequalities that the introduction of TDQL in a particular variant of DE improves its relative performance with that variant.

As no EA is full proof, this also is equally applicable for DE-TDQL as well. Although in most of the traditional optimization problems with/without constraints and any extension thereof (for example, multiobjective optimization), the proposed DE-TDQL is expected to outperform most of its competitors, it has a relatively poor performance in dynamic optimization. The justification to its failure is apparent because the scaling factors selected by TDQL for the individual member of the population usually should not work under dynamic environment. For example, in stock prediction problem [60], [61] undertaken by EAs, DE-TDQL is expected to have worse performance than common variants of DE particularly in time points, where the parameters of the model used for prediction (for example, a polynomial time function) have abrupt changes. The justification to this is apparent as the learnt scaling factors for a given member of the population cannot be utilized for a new set of model parameters. A similar situation is expected to occur in the path-planning problem of mobile robots with noisy sensory measurements. We have performed some experiments with the above preceding problems, but the details of this are outside the purview of the present paper.

VI. CASE STUDY IN MULTIROBOT PATH PLANNING

Multirobot path planning refers to determining the trajectory of motion of robots between predefined starting and goal positions in a given world map. Usually, one or more optimality criteria are imposed in path-planning problems. The criteria include minimization of the total path/time of traversal or energy or their combinations. This paper provides a solution to the multirobot path-planning problem using EA. Here, we formulated the multirobot path planning as an optimization problem with an objective to minimize the total traversed path by the robots without hitting obstacles/teammates. The problem has been solved here using the proposed DE-TDQL-based AMA. Both centralized and distributed approaches to multirobot path planning are found in the literature [24], [25], [47]. Here, we attempt to solve the problem using a distributed approach, particularly for its good time efficiency [70]. Computer simulations are used to study the relative performance of the proposed realization using DE-TDQL with respect to other well-known optimization algorithms.

A. Formulation

In the present context, we consider a 2-D workspace, partitioned into equal-sized square grids containing two or more mobile robot and obstacles with linear boundary. The robots are considered to have circular geometry with radius less than half of a side of the square grids. Grids are referred to by their distinct integer addresses. The obstacles are represented by the coordinates of their vertices. They can be of arbitrary shape and need not be encapsulated within grids. The starting and goal coordinates for each robot are fixed in the workspace, and these coordinates need not necessarily fall in a grid. A potential robot path between a given starting and a goal position is constructed by joining two or more line segments. The line segments pass through a number of junctions called intermediate nodes. The intermediate nodes are symbolized by their grid numbers. A robot path is considered to be feasible if none of the line segments intersect any obstacles. While planning a trajectory for a robot, other mobile robots are considered as moving obstacles. The path-planning problem for each robot is executed in steps until all robots reach their respective (predefined) goal positions. The formulation considers the evaluation of the next position of the robots from their current position.

Here, we represent a solution by a structure containing \( n \) fields, where the first (S) and last fields (T) indicate the starting and goal positions of the mobile robot. The second onwards successive \( (n - 2) \) fields represent the intermediate nodes. Fig. 3 gives a pictorial representation of a solution encoding a possible path for a mobile robot within the workspace, as shown in Fig. 4.

We now propose an evaluation method to first check the feasibility of a path by detecting the intersection between its constituent line segments and obstacles/teammates in the robots’ world map. If all the line segments in a path are found to
be free from intersection, the path length, defined by the sum of the length of the line segments in the planned path, is assigned as its cost indicating the quality of the solution. Otherwise, the evaluation method assigns the cost by estimating the “depths” of intersection of the constituent lines lying on the path with obstacles. The cost thus measured is an indirect measure of the difficulty faced by the path to escape from obstacles. A cost function for a solution representative of a possible path for the path to escape from obstacles. A cost function for a solution representative of a possible path for the path to escape from obstacles.

\[ F_i = \sum_{j=1}^{N} (d_{ij} + \beta_j C) \]  \hspace{1cm} (13)

where \( N \) is the number of line segments in a path, and \( d_{ij} \) is the Euclidean distance of the two successive nodes forming the \( j \)-th line segment. The factor \( C \) is used to maintain uniformity of magnitude of the two summations. \( \beta_j \) is the coefficient denoting the depth of collision, which is defined as

\[ \beta_j = \begin{cases} \sum_{k=1}^{M} \alpha_k & \text{if } j \text{-th line segment intersects obstacles} \\ 0 & \text{if } j \text{-th line segment is feasible} \end{cases} \]  \hspace{1cm} (14)

Here, \( M \) is the number of obstacles that the \( j \)-th line segment intersects, and \( \alpha_k \) is determined by measuring the depth of an intersecting line segment with an obstacle \( k \). \( \alpha_k \) is defined as the shortest moving distance for escaping the intersected obstacle [47]. The cost function \( F_i \) is to be minimized to determine the next position of robot \( i \). The minimization of \( F_i \) is to be performed for all \( i \) in parallel. This has been taken care of by \( n \) DE-TDQLs each engaged to minimize one \( F_i \) for \( i = 1 \) to \( n \), where \( n \) is number of robots.

We now illustrate the measurement of \( \alpha_k \) and \( \beta_j \) in Fig. 5. In Fig. 5(a), \( \alpha_k \) is treated as the shortest distance to move the line out of the obstacle \( k \). Fig. 5(d) elucidates a special example, where the line segment intersects two obstacles. It is evident that it is very difficult to determine the amount of shift of the line segment that will make it possible to move away from both obstacles. Therefore, the sum of \( \alpha_1 \) and \( \alpha_2 \) is used for the calculation of \( \beta_j \). For other complex configurations, the reader may consult the paper of Yang [47].

B. Experiments

The experiments were undertaken in two phases: first by computer simulation on a Pentium machine, and later on a real platform using two Khepera II mobile robots.

1) Experiments in Simulated Environment: Experiments were performed with \( n \) (2 \( \leq n \leq 14 \)) similar soft-bots of circular cross section on a Pentium machine. The radius of robot was set to 6 pixels. For each robot, the starting and goal points are predefined prior to initiating the experiment. The experiments were performed with 2, 4, 6, 8, and 10 differently shaped obstacles. While performing the experiments, old obstacles were retained, and new obstacles were added. Extensive experiments were performed with 50 world maps of diverse configurations. One of our experimental world maps with two dark obstacles, given starting and goal positions of six circular soft-bots, and theoretical (straight line paths) and planned trajectories (curved paths) obtained by minimization of (13) in each step of planning using DE-TDQL, is shown in Fig. 6.

2) Experiments in Real Environment on Khepera II Platform: The experiment was undertaken with a world map of \( 8 \times 6 \) grids of equal size and two Khepera II mobile robots (diameter of 7 cm). Each robot is equipped with eight infrared sensors, two motor-driven side wheels, and one caster wheel. The range sensors are positioned at fixed angles and have limited range detection capabilities. The sensors are numbered between 0 and 7 with the leftmost sensor designated by 0 and the rightmost by 7. The robot represents measured range data in the scale [0, 1023]. When an obstacle is away from the sensor by more than 5 cm, it is represented by zero. When an obstacle is approximately 2 cm away, it is represented by 1023. The onboard microprocessor includes a flash memory of 512 kB and a Motorola 68331 25-MHz processor.

The robots were controlled by two Pentium-IV personal computers (PCs) through wired connections. The robots were used to sense obstacles around them in the world map and turn wheels by motor firing for controlled movement in prescribed directions. A control program that determines the next position of a robot from its current position using DE-TDQL-based optimization algorithm is run on the attached Pentium machine. The necessary commands for motor movements are transferred to the robots from their connected computers. One sample run of path planning in the real environment is given in Fig. 7. It is observed from Fig. 7 that the robots follow the shortest paths, avoiding collision with obstacles. The experiment was
performed on 10 different world maps of different grid counts, each with five different obstacle maps, and in all the 50 environments, the robots could successfully trace the shortest paths.

C. Performance Analysis

To determine a quantitative measure of the relative performance of the different algorithms, two metrics are used here, as suggested in [24]–[26]. The definitions of the two performance metrics are given below for the sake of completeness of this paper.

Average Total Path Deviation (ATPD) [24]: Let $P_{ik}$ be a path from the starting point $S_i$ to the goal point $G_i$ generated by the program for robot $R_i$ in the $k$th run. If $P_{i1}, P_{i2}, \ldots, P_{ik}$ are the paths generated over $k$ runs, then the average path traversed (APT) by robot $R_i$ is given by $\frac{\sum_{j=1}^{k} P_{ij}}{k}$, and the average path deviation for this robot is evaluated by measuring the difference between APT and the ideal shortest path between $S_i$ and $G_i$ (with minimum threshold spacing with each obstacle). The threshold in our experiment was considered to be one pixel. If the ideal path for robot $R_i$ obtained geometrically is $P_{i\text{-ideal}}$, then the average path deviation is given by $P_{i\text{-ideal}} - \frac{\sum_{j=1}^{k} P_{ij}}{k}$. Therefore, for $n$ robots in the workspace, the ATPD is $\sum_{i=1}^{n} P_{i\text{-ideal}} - \sum_{j=1}^{k} P_{ij}/k$.

Average Uncovered Target Distance (AUTD) [26]: Given a goal position $G_i$ and the current position $C_i$ of a robot on a 2-D workspace, where $G_i$ and $C_i$ are 2-D vectors, the uncovered distance of robot $i$ is $||G_i - C_i||$, where $||.||$ denotes Euclidean norm. For $n$ robots, the UTD is the sum of $||G_i - C_i||$, i.e., $UTD = \sum_{i=1}^{n} ||G_i - C_i||$. Now, for $k$ runs of the program, we evaluate the average of UTDs and call it the average UTD. In all our experiments, we set $k = 10$ (Fig. 8).

The performance analysis was undertaken on a simulation environment. First, we plot the APT for $n$ robots, called the average total path traversed (ATPT) by varying $n$ from 2 to 10 by generating paths using five different algorithms, including real-coded GA, PSO, DE/current-to-best/1, SaDE, and DE-TDQL. It is noteworthy from Fig. 9 that DE-TDQL has the least ATPT in comparison to other algorithms irrespective of the number of robots.

The second study on performance analysis was undertaken by plotting ATPD by generating paths by five different EAs (as used in APT) with the number of robots as variable. Fig. 10 provides the results of ATPD computation when the number of robots varies between 2 and 10. Here, too, we observe that DE-TDQL outperforms the remaining four algorithms as ATPD remains the smallest for DE-TDQL irrespective of the number of robots.

The last analysis on performance was undertaken by comparing AUTD over the number of planning steps. Fig. 11 provides a plot of AUTD when the paths are planned using the five algorithms referred to above with number of obstacles = 5 and no. of robots = 5. It is apparent from Fig. 11 that AUTD returns
the smallest value for DE-TDQL irrespective of the number of planning steps.

In brief, the proposed DE-TDQL-based path planning outperforms all the four other algorithms with respect to all three popular metrics.

Fig. 7 provides experimentally obtained trajectories planned by two mobile robots using five different algorithms, including DE-TDQL, SaDE, DE/current-to-best/1, PSO, and GA. Results of the experiments performed are summarized in Table IV. Three performance metrics, namely, 1) total number of steps taken to reach the goal, 2) APT, and 3) ATPD, have been used here too to determine the relative merits of DE-TDQL over the other algorithms. Table IV confirms that DE-TDQL outperforms the remaining four algorithms with respect to all the three metrics.

VII. CONCLUSION

This paper has introduced a new technique for efficiently employing both DE and Q-learning to develop an AMA. A relative comparison of the proposed technique with four variants of DE algorithms, including SaDE (the currently known best performing DE), envisages that the proposed DE-TDQL algorithm outperforms all its competitors with respect to accuracy and runtime required for convergence. A set of 25 CEC’2005 benchmark functions proposed by Suganthan has been used to arrive at the foregoing conclusions.

Besides the foregoing, one more fundamental claim of this paper is that if TDQL is used to select scaling factors of any variants of DE, the modified algorithm would outperform its fundamental counterpart in both accuracy and convergence time. For example, SaDE-TDQL has shown to have better performance in accuracy and runtime jointly than SaDE, its fundamental constituent. The basis of the foregoing conclusion follows directly from a measure of distance between the coordinates of each algorithm and the origin in the accuracy versus runtime space. The experimental results thus strongly claim that SaDE-TDQL can significantly enhance the performance of SaDE due to the adaptation of scaling factors of SaDE by the use of TDQL.

A case study on the multirobot path-planning problem has been undertaken to demonstrate the relative merits of using the proposed DE-TDQL-based AMA over other algorithms. A formulation of the objective function for the problem has been given following [47], and the DE-TDQL algorithm is employed to minimize the objective function to determine the next position of all the robots from their current positions in the given world map. The experiments undertaken reveal that the DE-TDQL-based AMA outperforms the classical DE and PSO, real-coded GA, and SaDE algorithms with respect to two parameters AUTD and ATPD. The experiments performed with Khepera II mobile robots also indicate that DE-TDQL-based AMA outperforms other realizations in a real environment, thereby justifying the efficacy of the proposed algorithm.

REFERENCES


Pratyusha Rakshit received the B.Tech. degree in electronics and communication engineering from the Institute of Engineering and Management, Salt Lake, India, in 2010, and the M.E. degree in control engineering from Jadavpur University, Kolkata, India, in 2012, where she is currently working toward the Ph.D. degree in the identification of protein and gene regulatory networks to control neuronal firing in mammals. She is the author of over 15 papers published in top international journals and conference proceedings. She served as a reviewer in IEEE-RTSS 2011, IEEE-TFS 2012, and IAJSC 2012. Her principal research interests include artificial intelligence, evolutionary computation, bioinformatics, pattern recognition, fuzzy logic, cognitive science, and human–computer interaction.

Amit Konar (SM’10) received the B.E. degree from the Bengal Engineering and Science University, Howrah, India, in 1983, and the M.E. Tel.E., M.Phil., and Ph.D. degrees from Jadavpur University, Kolkata, India, in 1985, 1988, and 1994, respectively.

In 2006, he was a Visiting Professor with the University of Missouri, St. Louis. He is currently a Professor with the Department of Electronics and Tele-communication Engineering (ETCE), Jadavpur University, where he is the Founding Coordinator of the M.Tech. program in intelligent automation and robotics. He has supervised 15 Ph.D. theses. He has over 250 publications in international journal and conference proceedings. He is the author of eight books, including two popular texts Artificial Intelligence and Soft Computing (CRC Press, 2000) and Computational Intelligence: Principles, Techniques and Applications (Springer, 2005). His research areas include the study of computational intelligence algorithms and their applications to the various domains of electrical engineering and computer science. Specifically, he worked on fuzzy sets and logic, neurocomputing, EAs, Dempster–Shafer theory, and Kalman filtering, and applied the principles of computational intelligence in image understanding, VLSI design, mobile robotics, pattern recognition, brain–computer interfacing and computational biology.

Dr. Konar serves as the Associate Editor of IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, PART A, and IEEE TRANSACTIONS ON FUZZY SYSTEMS. He was the recipient of All India Council for Technical Education (AICTE)-accredited 1997–2000 Career Award for Young Teachers for his significant contribution in teaching and research.

Pavel Bhowmik received the B.E. degree from Jadavpur University, Kolkata, India, in 2010. He is currently working toward the M.S. degree in electrical engineering at the University of Florida, Gainesville.

He is currently with the Electronics and Telecommunication Engineering Department, Jadavpur University. He has published more than 15 research articles in peer-reviewed journals and international conferences. He served as a reviewer in IEEE CEC 2011, ICCSH 2012, ISIEA 2012, and PECON 2012. His research interests include various aspects of artificial intelligence like machine learning, fuzzy logic, evolutionary algorithms, and emotion recognition. He is also interested in control and communication issues of smart grid and the application of various AI techniques in this aspect.

Indrani Goswami (Chakraborty) received the B.E. degree in electrical engineering, the M.E. degree (with specialization in Control Engineering) in electronics and communication engineering, and the Ph.D. degree in cognitive robotics from Jadavpur University, Kolkata, India, in 1993, 1988, and 2012, respectively.

She was a Visiting Research Scientist with the Robotics Laboratory, Mie University, Tsu, Mie, Japan. She has been teaching with the Calcutta Institute of Technology, Calcutta, India, for the last two years. She is currently with the Electronics and Telecommunication Engineering Department, Jadavpur University. She has published a number of interesting papers in top international journals and conference proceedings. Her current research interest includes machine intelligence and robotics, industrial control, cognitive science, brain–computer interfacing, and evolutionary algorithms.
Swagatam Das (M’10) received the B.E.Tel.E., M.E.Tel.E. (control engineering specialization), and Ph.D. degrees from Jadavpur University, Kolkata, India, in 2003, 2005, and 2009, respectively. He served as an Assistant Professor with the Department of Electronics and Telecommunication Engineering, Jadavpur University, from 2006 to 2011. He is currently serving as an Assistant Professor with the Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata. He has published more than 140 research articles in peer-reviewed journals and international conferences. He is the founding Coeditor-in-Chief of “Swarm and Evolutionary Computation,” an international journal from Elsevier. He coauthored a research monograph on metaheuristic clustering techniques from Springer in 2009. His current research interests include evolutionary computing, pattern recognition, multiagent systems, and wireless communication.

Dr. Das serves as an Associate Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, PART A, and the INFORMATION SCIENCES Journal (Elsevier). He is an editorial board member of the Progress in Artificial Intelligence (Springer), Mathematical Problems in Engineering, International Journal of Swarm Intelligence, International Journal of Artificial Intelligence and Soft Computing, and International Journal of Adaptive and Autonomous Communication Systems. He has been acting as a Regular Reviewer for journals like Pattern Recognition, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, EVOLUTIONARY COMPUTATION (MIT Press), and IEEE TRANSACTIONS ON SMC, PART A, PART B, AND PART C. He has acted as Guest Editor for special issues in journals like the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and IEEE TRANSACTIONS ON SMC, PART C. He is in the organizing and program committees of several yearly conferences like GECCO, IEEE CEC, IEEE SSCI, SEMCCO, and SEAL.

Lakhmi C. Jain is currently a Professor of knowledge-based engineering and the Director/Founder of the Knowledge-Based Intelligent Engineering Systems Centre, University of South Australia, Adelaide, Australia. His interests focus on artificial intelligence paradigms and their applications in complex systems, art-science fusion, e-education, e-healthcare, robotics, unmanned air vehicles, and intelligent agents.

Prof. Jain is a Fellow of the Institution of Engineers Australia.

Atulya K. Nagar received the B.Sc. (Hons.), M.Sc., and M.Phil. (with Distinction) degrees from the MDS University of Ajmer, India. A mathematician by training, he possesses multidisciplinary expertise in natural computing, bioinformatics, operations research, and systems engineering. He has an extensive background and experience of working in universities in the U.K. and India. He was previously was with the Department of Mathematical Sciences, and later the Department of Systems Engineering, Brunel University, London, U.K. He holds the Foundation Chair and is a Professor of computer and mathematical sciences with the Liverpool Hope University, Liverpool, U.K., and is the Head of the Department of Mathematics and Computer Science. He has been an Expert Reviewer for the Biotechnology and Biological Sciences Research Council (BBSRC), grants peer-review committee for Bioinformatics Panel, and serves on Peer-Review College of the Arts and Humanities Research Council (AHRC) as a Scientific Expert Member.

Dr. Nagar has coedited volumes on intelligent systems and applied mathematics. He is the Editor-in-Chief of the INTERNATIONAL JOURNAL OF ARTIFICIAL INTELLIGENCE AND SOFT COMPUTING (IJASIC) and serves on editorial boards for a number of prestigious journals as well as on International Program Committee (IPC) for several international conferences. He received a prestigious Commonwealth Fellowship for working toward his doctorate in applied nonlinear mathematics, which he earned from the University of York in 1996.