# Realization of an Adaptive Memetic Algorithm Using Differential Evolution and Q-learning: A Case Study in Multi-Robot Path-Planning

Pratyusha Rakshit[1], Amit Konar[1], Pavel Bhowmik[1], Indrani Goswami[1], Swagatam Das[1], Lakhmi C. Jain[2], Atulya K. Nagar[3]

[1]ETCE Department, Jadavpur University, Kolkata-32, India

[2]Dept. of Electrical and Electronics Engineering, Univ. of South Australia, Adelaide

[3]Dept. of Math and Computer Science, Liverpool Hope University

Contact Author: Amit Konar (konaramit@yahoo.co.in)

***Abstract—* Memetic algorithms are population-based meta-heuristic search algorithms that combine the composite benefits of natural and cultural evolution. An adaptive memetic algorithm 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 provides a novel approach to design an adaptive memetic algorithm by utilizing the composite benefits of Differential Evolution for global search and Q-learning for local refinement. Four variants of Differential Evolution including the currently best Self-Adaptive Differential Evolution algorithm have been used here to study the relative performance of the proposed adaptive memetic algorithm with respect to runtime, cost function evaluation and accuracy (offset in cost function from the theoretical optimum after termination of the algorithm). Computer simulations undertaken on a well-known set of 25 benchmark functions reveals that incorporation of Q-learning in one popular and one outstanding variants of Differential Evolution makes the corresponding algorithm more efficient in both runtime and accuracy. The performance of the proposed adaptive memetic algorithm has been studied on a real-time multi-robot path-planning problem. Experimental results obtained for both simulation and real frameworks indicate that the proposed algorithm based path-planning scheme outperforms real coded Genetic Algorithm, Particle Swarm Optimization and Differential Evolution, particularly its currently best version with respect two standard metrics defined in the literature.**

***Index terms-* Adaptive memetic algorithm; Q-learning; Differential evolution; Multi-robot path-planning.**

## I. INTRODUCTION

Coined 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 evolution. 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 local search by its cultural component.

The early research on MA was confined in 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 favors 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], [4], [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 Differential Evolution (DE) algorithm for its proven merits in global optimization [20], [26], [10]. 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/evolutionary algorithms. 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 member of the population in a DE algorithm should not be equal for the 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, while 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 run time. Experiments have also been undertaken to compare the performance of a classical Self-adaptive Differential Evolution (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 online trajectory planning of mobile robots from given starting positions to fixed goal positions without hitting teammates and obstacles [25], [49]. There exist two alternative approaches, centralized and distributed, to handle the problem. In a centralized approach, the next position of all the robots are determined from their current positions, by minimizing an objective function concerning total path of traversal by the robots, satisfying the necessary constraints on collision avoidance of individual robots with teammates and obstacles. The above problem is solved by iteratively identifying next positions of the robots until the goal position for all the robots are 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 distributed environment boils down to relatively simplified problem of minimization of n objective functions for n robots. The dynamic changes in the current and the next positions for each robot are taken care of by an iterative manner as in case of centralized approach.

There exists extensive literature on path-planning by single robotic agent employing graphs [50], [51], [52], neural nets [53], [54], fuzzy logic [55], [56] and evolutionary algorithms [57], [45], [58]. Recently researchers are taking keen interest to consider multi-robot 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 multi-robot path-planning in comparison to its single robot counterpart. The existing literature on multi-robot path-planning employs graphs [64], Voronoi diagrams [65], and potential field techniques [66] to handle the problem. Only in the last few years, traces of evolutionary algorithms in multi-robot environment [24], [60], [68], [69] could be detected. The evolutionary multi-robot path-planners found in the literature usually consider point-mass robots [59], [60] and thus are not amenable for real world applications. Although 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 [68], [70], [71] only.

This paper considers an evolutionary path-planning with robots of definite size and circular cross-section and tested both in simulation and real environments. The workspace here is partitioned into square grids of equal size. The starting and the 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 Genetic Algorithm (GA) based realization given in [49]. Experiments undertaken further to compare the relative performance of the AMA based path-planner with other swarm/evolutionary algorithm 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.

The paper is divided into eight sections. Section II provides an overview of the classical Q-learning. Section III introduces the Differential Evolution algorithm. In section IV, we propose the AMA realized with DE and TDQL. Section V reports the results of performance analysis of the reported AMA. Section VI provides the formulation of the multi-robot motion planning problem and experiment with Khepera II mobile robots and computer simulation. Conclusions are given in section VII.

## II. An Overview of the Classical Q-Learning

Learning helps an agent performing 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 in attempting 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 in order 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 [61], 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

$S=\{s_1,s_2,...,s_n\}$ be a set of states of an agent in a given environment,

$A=\{a_1,a_2,...,a_n\}$ 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 $s_i$,

$\delta(s_i,a_j)$ be the transition function that returns the next state $s_k$ 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), and

$Q(s_i,a_j)$ 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 from its next state. Let $\overset{*}{V}(s)$ 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, $\overset{*}{V}(s)$ is approximated as $\underset{a'}{Max}\,Q(s,a')$. Thus $Q(s,a)$ evaluation becomes simplified by (1).

$$Q(s,a) = r(s,a) + \gamma \overset{*}{V} Q(\delta(s,a)) \qquad \text{(by definition)}$$
$$= r(s,a) + \gamma \underset{a'}{Max} Q(\delta(s,a),a') \qquad (1)$$

The classical Q-learning for deterministic state-transition is given below. Here, the algorithm begins with a randomly selected initial state. An action $a \in A$ is randomly selected, and the agent because of this action receives an immediate reward *r*, and moves to the new state using $\delta$-transition rule, provided in a table. The Q-value of the previous state s due to selected action $a$ is updated in a two-dimensional 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.

**Pseudo-Code 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 $a \in A=\{a_1,a_2,...,a_n\}$ 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
    $Q(s,a) \leftarrow r(s,a) + \gamma \underset{a'}{Max} Q(\delta(s,a),a')$ ;
    $s \leftarrow s^{/}$;
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

$$Q(s,a) \leftarrow (1-\alpha)\times Q(s,a) + \alpha \times (r(s,a) + \gamma \underset{a'}{Max} Q(\delta(s,a),a')) \ (2)$$

The formula has the effect, that the Q-value $Q(s,a)$ is incremented, when the action $a$ led 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 plus 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 to the extent the newly acquired information will override the old information. A setting of $\alpha$= 0 makes the agent stop learning, while α=1 would make the agent consider only the most recent information. The discount factor γ determines the importance of future rewards. A factor of 0 will make the agent "opportunist" by only considering current rewards, while 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. An Overview of Differential Evolution Algorithm

DE is a population based meta-heuristic algorithm, which 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], [73]. DE involves a population of NP parameter vectors, which at generation G is denoted by $P_G = \{\vec{X}_1(G), \vec{X}_2(G), ..., \vec{X}_{NP}(G)\}$.

**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 $\vec{X}_{\min} = \{x_{\min-1}, x_{\min-2}, ..., x_{\min-D}\}$ and $\vec{X}_{\max} = \{x_{\max-1}, x_{\max-2}, ..., x_{\max-D}\}$, and thus the j-th component of the i-th member at G=0 is given by

$$x_{i,j}(0) = x_{j-\min} + rand_{i,j}(0,1) \times (x_{j-\max} - x_{j-\min}) \quad (3)$$

where $rand_{i,j}(0,1)$ is a uniformly distributed random number lying between 0 and 1. Initialize crossover rate Cr randomly in [0, 1].

**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 $\vec{X}_{rand-1}(G)$ and $\vec{X}_{rand-2}(G)$ from the current population $P_G$, where

$$\vec{V}_i(G) = \vec{X}_i(G) + F_1'(\vec{X}_{best}(G) - \vec{X}_i(G)) + F_2'(\vec{X}_{rand-1}(G) - \vec{X}_{rrand-2}(G)) \quad (4)$$

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], [43].

**3. Crossover:** There are two types of crossover (recombination) schemes- binomial and exponential [10, [11]. We outline both binomial and exponential crossover here, which we would use in AMA algorithm of section-IV.

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

$$u_{i,j}(G) = \begin{cases} v_{i,j}(G) \text{ if } rand_{ij} \le Cr \text{ or } j = j_{rand} \\ x_{i,j}(G) \text{ otherwise} \end{cases} \quad (5)$$

where $rand_{i,j}(0,1) \in [0, 1]$ is a uniformly distributed random number lying in [0,1] and is instantiated independently for each j-th component of the i-th vector. $j_{rand} \in [1, D]$ 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, we randomly select an integer n from [1, D] and use it as a starting point in the target vector to represent the beginning of the crossover or exchange of components with the donor vector. We also select another integer L 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

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

where the angular brackets $\langle \ \rangle_D$ denote a modulo function with modulus D. The integer L is selected from [1, D] by the following pseudo-code:

```
L = 0;
DO
{
    L = L + 1;
} WHILE (((rand (0, 1) < Cr) AND (L<D)).
```

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

$$\begin{aligned} \vec{X}_i(G+1) &= \vec{U}_i(G) \quad \text{if } f(\vec{U}_i(G)) \le f(\vec{X}_i(G)) \\ &= \vec{X}_i(G) \quad \text{if } f(\vec{U}_i(G)) > f(\vec{X}_i(G)) \end{aligned} \quad (7)$$

The steps 2 to 4 are repeated until a stopping criterion is reached.

## IV. Proposed Adaptive Memetic Algorithm

The AMA to be proposed shortly includes a DE for global exploration and a 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/penalty given to a member is stored in the Q-table using the TDQL learning rule. A meme pool of a 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/penalty updating in the Q-table is continued until the condition for convergence of the AMA is satisfied. In this paper, F1' and F2' are set equal to F to save complexity to avoid maintenance of two Q-tables for each individual scaling factors.

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 evolutionary algorithm. Let 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 the $Q(S_i, 10F_j)$, j=1, 2, …, 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 evolutionary algorithm, 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.

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

$$\vec{X}_{\min} = [x_{1-\min}, x_{2-\min}, \ldots\ldots\ldots, x_{D-\min}] \quad \text{and}$$
$$\vec{X}_{\max} = [x_{1-\max}, x_{2-\max}, \ldots\ldots\ldots, x_{D-\max}]$$

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

$$x_{i,j}(0) = x_{j-\min} + rand_{i,j}(0,1) \times (x_{j-\max} - x_{j-\min}) \qquad (8)$$

where $rand_{i,j}(0,1)$ 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.

**2. 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, …, 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 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

$$P(F_j) = Q(S_i, 10\,F_j) \,/\, \sum_{l=1}^{10} Q(S_i, 10\,F_l) \qquad (9)$$

To maintain adaptation and learning in all Q's in each row, we select a particular F from the meme pool by a 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 hold:

$$\sum_{m=1}^{j-1} P(F = F_m) < r \le \sum_{m=1}^{j} P(F = F_m) \qquad (10)$$

$$\Rightarrow \frac{\sum_{m=1}^{j-1} Q(S_i, 10F_m)}{\sum_{l=1}^{10} Q(S_i, 10F_l)} < r \le \frac{\sum_{m=1}^{j} Q(S_i, 10F_m)}{\sum_{l=1}^{10} Q(S_i, 10F_l)} \qquad (11)$$

**3. Differential Evolution:** 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 $F_1^{/}$ and $F_2^{/}$ from the meme pool adaptively by step 2 before invoking the DE process.

**4. 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 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.

**5. Reward/Penalty based Q-table updating**: Let a member at state $S_i$ on selection of $F_j$ moves 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$(S_i, 10F_j)$ = increase in fitness of the member, where

$$\begin{aligned} Q(S_i, 10F_j) &= (1-\alpha) Q(S_i, 10F_j) \\ &\quad + \alpha(reward\ (S_i, 10F_j) + \gamma \max_{F^{/}} Q(S_k, 10F^{/})) \end{aligned} \qquad (12)$$

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

**6. Convergence:** After each evolution, we repeat from step 2 until one of the following conditions for convergence is satisfied. The conditions include restricting the number of iterations, maintaining error limits, or the both, whichever occurs earlier.

```
Pseudo Code of AMA
I.  Set the generation number $t=0$ and randomly initialize a population of NP individuals
    $P_t=\left\{\vec{X}_1(t),\vec{X}_2(t),...,\vec{X}_{NP}(t)\right\}$ with
    $\vec{X}_i(t)=\{x_{i,1}(t),x_{i,2}(t),...,x_{i,D}(t)\}$ for $i=[1,2,...,NP]$ and each
    individual uniformly distributed in the range $[\vec{X}_{\min},\vec{X}_{\max}]$, where
    $\vec{X}_{\min}=\{x_{\min-1},x_{\min-2},...,x_{\min-D}\}$ and
    $\vec{X}_{\max}=\{x_{\max-1},x_{\max-2},...,x_{\max-D}\}$. Set α=0.25, γ=0.8.
    Evaluate $f(\vec{X}_i(t))$, for target vector $\vec{X}_i(t)$, $i=[1,2,...,NP]$.
    Rank each vector according ascending order of cost function f(.).
    Let the ranked population be $R(0)=[r_1(0),r_2(0),...,r_{NP}(0)]$,
    where $r_i(0)$ denotes a target vector of rank i in t-th generation.
    Initialize $[Q(r_i(0),j)]=1,\ ,\ j=[1,2,...,10]$ .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 $[reward(r_i(t),j)]=0,\ r_i=[1,2,...,NP]$,
    $j=[1,2,...,10]$.
    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 $P(j)=Q(r_i(t),j)\Big/\sum_{l=1}^{10}Q(r_i(t),l)$.
      II.b. Mutation:
        Generate a donor vector $\vec{V}_i(t)=\{v_{i,1}(t),v_{i,2}(t),...,v_{i,D}(t)\}$
        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 $\vec{X}_{best}(t)$ 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 $\vec{U}_i(t)=\{u_{i,1}(t),u_{i,2}(t),...,u_{i,D}(t)\}$
        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
            $reward(r_i(t),10\times F_{r_i})=f(\vec{X}_i(t))-f(\vec{U}_i(t));$
            $\vec{X}_i(t+1)=\vec{U}_i(t);$
            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;
          Evaluate $f(\vec{X}_i(t+1))$ and save it for future.
    End For;
    II.e. Rank each vector according ascending order of cost function
          f(.). Let the ranked population
          be $R(t+1)=[r_1(t+1),r_2(t+1),...,r_{NP}(t+1)]$, 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
          $Q(r_i(t),10\times F_j)=(1-\alpha)Q(r_i(t),10\times F_j)$
            $+\alpha[reward(r_i(t),10\times F_j)+\gamma\max_F Q(r_i(t+1),10\times F)];$
        Else $Q(r_i(t),10\times F_j)=Q(r_i(t),10\times F_j)$ ;
        End If;
      End For;
    End For;
    Increase the counter value $t=t+1$.
  End While;
```

## V. Simulation Results

In this section, we compare DE-TDQL with 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 [43]: (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 evolutionary algorithm is to identify the right benchmark functions with diverse characteristics of the functions, such as uni-modality, multimodality and particularly location of the optima on the surface. Traditional benchmark functions usually have the global optimum at the centre, surrounded by several local optima along the axes. Naturally, these benchmark functions are inadequate to exhaustively test the performance of an optimization algorithm. In order to overcome the above problem, a set of recommended benchmark functions [48] was proposed in the Congress of Evolutionary Computation (CEC'2005) 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.

Here, we test the relative performance of our algorithm with other variants of DE using 25 benchmark functions, recommended in [48], of 10, 30 and 50 dimensions. The list of the 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. Maximum number of fitness evaluation (Max_FEs) is set at 100,000 for 10-D, 300,000 for 30-D and 500,000 for 50-D. For lack of space, the experimental results for 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 [43]. In this paper, we have adopted an asymmetric initialization procedure following the works reported in [74].

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

To judge the accuracy of different DE variants, we first let each of them run for a very long time 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 Cr=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 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, say 25. In the last columns of Table-I, we report the statistical significance level of the difference of the means of 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, while '-' 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.

### *D. Performance Analysis*

In order 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. Then we note the number of FEs the algorithm takes. 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. One may note from Table I, that for a few relative 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 best average accuracy beating DE-TDQL, which remained the second best algorithm.

Table-II indicates that not only does DE-TDQL yield the most accurate results for nearly all the benchmark problems, but the number of runs that converge below a pre-specified cut-off 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 Self-adaptive Differential Evolution (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 dependant. 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. Additionally, 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 the 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: f13, f14, and f20. 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, SaDE-TDQL, we in Fig. 1 plotted the mean value of the objective function (mean best fitness) taken over 25 runs versus function evaluations (FEs), and note that DE-TDQL outperforms all other variants of DE considered above, while SaDE-TDQL outperforms all algorithms including DE-TDQL. The above 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 in performance of the six algorithms including DE-TDQL can be performed 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, while in Fig 2(b) we provide a plot of accuracy versus run-time 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 the y-coordinates (in order 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 ‘>=’ symbol to represent the relative performance of two algorithms. Using this convention, we note from Fig. 2(a) and (b) that the performance of the six algorithms respectively is

DE-TDQL >= SaDE-TDQL>= SaDE>= DE/current-to-best/1 >= DE/rand/either-or>= DE/rand/1, and

SaDE-TDQL>= DE-TDQL>=SaDE>=DE/current-to-best/1 >= DE/rand/either-or>= DE/rand/1 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 run-time 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 evolutionary algorithm 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, multi-objective 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 individual member of the population usually should not work under dynamic environment. For example, in stock prediction problem [62], [63] undertaken by evolutionary algorithms, 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 (say 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. Similar situation is expected to occur in path-planning problem of mobile robots with noisy sensory measurements. We have performed some experiments with the above two problems, but the details of this are outside the purview of the present paper.

TABLE-I

PERFORMANCE OF THE PROPOSED DE-TDQL BASED AMA WITH OTHER VARIANTS OF DE (FOR D=50 AND CR=0.9 )

| Function No. | DE/rand/1 | DE/rand/either-or | DE/current-to-best-1 | DE-TDQL | Statistical Significance |
|---|---|---|---|---|---|
| f 01 | 7.62981e-009 (5.76298e-009) | 1.22946e-006 (7.71943e-007) | 0.00000e+000 (0.00000e+000) | 0.00000e+000 (0.00000e+000) | NA |
| f 02 | 1.06345e-002 (8.7392e-003) | 1.46318e-006 (7.19554e-007) | 0.00000e+000 (0.00000e+000) | 0.00000e+000 (0.00000e+000) | NA |
| f 03 | 1.46740e+004 (7.5233e+003) | 5.94211e-003 (2.37703e-003) | 9.8472e-011 (2.94160e-011) | **0.00000e+000 (0.00000e+000)** | + |
| f 04 | 9.38707e-002 (7.25890e-002) | 79104e-005 (1.53892e-005) | 0.00000e+000 (0.00000e+000) | 0.00000e+000 (0.00000e+000) | NA |
| f 05 | 2.42849e+002 (5.08024e+001) | 2.85470e+000 (1.21284e+000) | 9.5635e-005 (1.2491e-005) | **2.27374e-012 (1.31602e-012)** | + |
| f 06 | 1.11454e+002 (2.60841e+002) | 1.64746e+002 (2.76433e+002) | 1.35054e+000 (5.97550e+000) | **4.78389e-001 (1.32220e+000)** | + |
| f 13 | 2.82947e+000 (4.29449e-001) | 2.90694e+000 (5.35088e-001) | 2.15193e+000 (4.3762e-001) | **1.18067e+000 (2.70485e-001)** | + |
| f 15 | 6.53924e+002 (1.7351e+001) | 6.80948e+002 (9.3627e+001) | **4.28239e+002 (7.51763e+001)** | 4.29234e+002 (1.12549e+001) | - |
| f 16 | 1.80043e+002 (1.31896e+001) | 1.77648e+002 (1.09782e+001) | 1.65812e+002 (2.06795e+001) | **1.12813e+002 (1.31497e+001)** | + |
| f 17 | 2.03029e+002 (1.51955e+001) | 2.01405e+002 (1.74092e+001) | 1.78097e+002 (1.91219e+001) | **1.34020e+002 (1.84988e+001)** | + |
| f 18 | 8.40003e+002 (4.35765e+001) | 8.48183e+002 (4.92894e+001) | **7.49505e+002 (2.34925e+002)** | 7.63328e+002 (1.77011e+002) | + |
| f 19 | 8.30678e+002 (22685e+001) | 8.46269e+002 (4.60059e+001) | 8.23536e+002 (1.18894e+002) | **8.19489e+002 (1.20251e+002)** | + |
| f 20 | 8.75395e+002 (5.77017e+001) | 8.51891e+002 (5.06452e+001) | 8.16640e+002 (1.64708e+002) | **6.84876e+002 (2.51560e+002)** | + |
| f 21 | 1.10406e+003 (8.25569e+000) | 1.07624e+003 (1.11391e+001) | 1.03193e+003 (1.53154e+002) | **7.08847e+002 (1.3880e+002)** | + |
| f 22 | 5.28406e+002 (4.13118e-001) | **5.27321e+002 (4.55182e-001)** | 6.06447e+002 (1.30617e+002) | 7.63716e+002 (1.88575e+001) | - |
| f 23 | 1.10932e+003 (5.24023e+000) | 1.09531e+003 (9.43862e+000) | 1.02561e+003 (1.56514e+002) | **8.88592e+002 (2.21409e+002)** | + |
| f 24 | 8.4588e+002 (2.98500e+000) | 8.0102e+002 (8.3361e+000) | 7.5841e+002 (1.8062e+000) | **7.5238e+002 (4.34601e+000)** | + |
| f 25 | 8.3175e+002 (4.01108e+000) | 7.9250e+002 (3.4899e+000) | **7.6823e+002 (2.99705e+000)** | 9.9675e+002 (5.12610e+001) | + |

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

| Function No. | Tolerance | DE/rand/1 | DE/rand/either-or | DE/current-to-best-1 | DE-TDQL |
|---|---|---|---|---|---|
| f 01 | 1.00e-04 | 25 (1.64190e+004) | 25 (2.00650e+004) | 25 (4.51000e+003) | **25 (2.63200e+003)** |
| f 02 | 1.00e-02 | 13 (2.43519e+004) | 25 (1.53580e+004) | 25 (5.56200e+003) | **25 (2.80000e+003)** |
| f 03 | 1.00e+05 | 25 (1.90340e+004) | 25 (6.96700e+003) | 25 (2.4300e+003) | **25 (1.59600e+003)** |
| f 04 | 1.00e+00 | 25 (1.98130e+004) | 25 (1.24390e+004) | 25 (4.21900e+003) | **25 (2.09400e+003)** |
| f 05 | 1.00e+03 | 25 (1.41600e+004) | 25 (6.95300e+003) | 25 (2.06400e+003) | **25 (7.21000e+002)** |
| f 06 | 5.00e+02 | 24 (1.61698e+004) | 22 (1.66045e+004) | 25 (2.88100e+003) | **25 (1.39100e+003)** |
| f 07 | 2.00e+00 | 25 (7.18900e+003) | 25 (6.88300e+003) | 25 (1.68000e+003) | **25 (9.31000e+002)** |
| f 08 | 2.04e+01 | 15 (1.08550e+004) | 18 (1.23597e+004) | 18 (9.07500e+003) | **25 (8.21800e+003)** |
| f 09 | 5.00e+01 | 25 (4.53800e+003) | 25 (9.69300e+003) | 25 (1.74600e+003) | **25 (5.13000e+002)** |
| f 10 | 5.00e+01 | 25 (9.78600e+003) | 25 (9.36200e+003) | 25 (2.22500e+003) | **25 (8.75000e+002)** |
| f 11 | 9.20e+00 | 19 (9.39079e+003) | 13 (7.97308e+003) | 17 (9.55294e+003) | **25 (6.18400e+003)** |
| f 12 | 4.00e+04 | 25 (4.21100e+003) | 25 (7.22700e+003) | 25 (5.44900e+003) | **25 (2.65300e+003)** |
| f 13 | 00e+00 | 16 (1.25391e+004) | 13 (1.71327e+004) | 25 (7.53300e+003) | **25 (2.15400e+003)** |

TABLE-II-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

| Function No. | Tolerance | DE/rand/1 | DE/rand/either-or | DE/current-to-best-1 | DE-TDQL |
|---|---|---|---|---|---|
| f 14 | 4.15e+00 | 14 (1.30357e+004) | 23 (8.75217e+003) | 25 (6.40800e+003) | **25 (7.38000e+002)** |
| f 15 | 7.08e+02 | 25 (4.74000e+003) | 19 (1.01908e+004) | 25 (5.7000e+002) | **25 (1.66000e+002)** |
| f 16 | 2.24e+02 | 25 (6.45000e+003) | 25 (7.20100e+003) | 25 (5.8800e+003) | **25 (1.28000e+003)** |
| f 17 | 2.50e+02 | 25 (6.91100e+003) | 25 (8.22500e+003) | 25 (5.8900e+003) | **25 (1.28400e+003)** |
| f 18 | 8.50e+02 | 21 (5.96667e+003) | 19 (2.53158e+003) | 18 (**1.01111e+003**) | **22** (2.36250e+003) |
| f 19 | 8.50e+02 | 23 (5.43913e+003) | 19 (2.86184e+003) | **20 (8.66250e+002)** | 16 (1.54688e+003) |
| f 20 | 8.75e+02 | 13 (4.81538e+003) | 18 (2.19444e+003) | 17 (**6.85294e+002**) | **20** (2.17250e+003) |
| f 21 | 1.10e+03 | 6 (1.96667e+004) | 24 (1.77708e+004) | 25 (3.8100e+003) | **25 (1.14200e+003)** |
| f 22 | 8.00e+02 | 25 (5.45000e+002) | **25 (4.46000e+002)** | 20 (5.8500e+003) | 22 (8.60227e+002) |
| f 23 | 1.15e+03 | 25 (4.6900e+003) | 25 (6.5100e+003) | 25 (1.02500e+003) | 24 (**8.35417e+002**) |
| f 24 | 4.10e+02 | 25 (4.54500e+003) | 25 (1.0000e+003) | 25 (8.86000e+002) | **25 (5.40000e+002)** |
| f 25 | 4.50e+02 | 25 (1.64800e+003) | 25 (1.49400e+003) | 25 (9.4000e+002) | 20 (**7.3750e +002**) |

TABLE-III

COMPARISON OF PERFORMANCE OF SADE AND SADE-TDQL WITH LP=50

| | D=10 | | | D=30 | | | D=50 | | |
|---|---|---|---|---|---|---|---|---|---|
| Functi on No. | SaDE | SaDE-TDQL | Stat. sig. | SaDE | SaDE-TDQL | Stat. sig. | SaDE | SaDE-TDQL | Stat. sig. |
| f01 | 3.02e-27<br>(2.13e-21) | **0.00e+00<br>(0.00e+00)** | + | 6.24e-29<br>(1.65e-10) | **0.00e+00<br>(0.00e+00)** | + | 2.33e-10<br>(1.65e-10) | **0.00e+00<br>(0.00e+00)** | + |
| f02 | 1.56e-09<br>(2.52e-05) | **3.34e-16<br>(0.00e+00)** | + | 1.66e-05<br>(1.12e-04) | **7.17e-10<br>(5.06e-07)** | + | 1.59e-02<br>(1.12e-02) | **7.17e-07<br>(5.07e-07)** | + |
| f03 | 8.02e-04<br>(5.36e-04) | **2.37e-13<br>(1.29e-19)** | + | 4.75e-04<br>(3.35e-05) | **1.57e-10<br>(7.21e-05)** | + | 4.75e+01<br>(3.35e-01) | **1.02e-04<br>(7.21e-05)** | + |
| f04 | 5.40e-03<br>(3.17e-05) | **1.23e-05<br>(0.00e+00)** | + | 5.22e-01<br>(1.92e-05) | **9.69e-02<br>(0.00e+00)** | + | 2.77e+01<br>(1.95e-01) | **9.69e-02<br>(6.85e-02)** | + |
| f05 | 6.74e-05<br>(5.05e-09) | **3.20e-10<br>(4.19e-09)** | + | 8.00e-03<br>(0.00e+00) | **4.72e-07<br>(3.02e-07)** | + | 8.00e-03<br>(5.60e-03) | **4.49e-08<br>(3.15e-08)** | + |
| f06 | 2.05e+01<br>(8.69e-06) | **6.38e+00<br>(3.14e-05)** | + | 3.33e+01<br>(1.93e-03) | **5.97e+00<br>(4.80e-03)** | + | 5.97e+00<br>(1.02e-01) | **5.29e+00<br>(7.70e-01)** | + |
| f07 | 1.25e-03<br>(0.00e+00) | **1.45e-05<br>(0.00e+00)** | + | 1.27e-03<br>(0.00e+00) | 1.27e-03<br>(0.00e+00) | NA | 1.27e-03<br>(0.00e+00) | 1.27e-03<br>(0.00e+00) | NA |
| f08 | 2.06e+01<br>(1.41e-17) | **2.05e+01<br>(1.44e-19)** | + | **2.04e+01<br>(7.07e-10)** | 2.05e+01<br>(4.10e-15) | - | 2.05e+01<br>(7.07e-12) | **2.04e+01<br>(7.07e-02)** | + |
| f09 | 2.71e-10<br>(0.00e+00) | **0.00e+00<br>(8.62e-28)** | + | 8.35e-08<br>(5.73e-08) | **2.21e-11<br>(0.00e+00)** | + | 2.37e-09<br>(1.48e-09) | **2.21e-11<br>(1.56e-11)** | + |
| f10 | 2.45e+01<br>(1.18e-04) | **1.13e+01<br>(7.07e-06)** | + | 3.38e+01<br>(3.53e-02) | **1.06e+01<br>(0.00e+00)** | + | 2.88e+01<br>(3.04e+00) | **1.06e+01<br>(4.94e-01)** | + |
| f11 | 7.29e+00<br>(1.41e-09) | **6.27e+00<br>(6.64e-13)** | + | 8.40e+00<br>(1.19e-11) | **6.71e+00<br>(4.10e-17)** | + | 6.71e+00<br>(4.10e-08) | **6.13e+00<br>(9.89e-02)** | + |
| f12 | **2.45e+03<br>(4.31e-04)** | 2.47e+03<br>(1.26e-05) | - | 1.80e+03<br>(3.74e-04) | **1.27e+03<br>(2.82e-01)** | + | 1.27e+03<br>(8.34e-02) | **1.23e+03<br>(8.76e-02)** | + |
| f13 | **6.38e-01<br>(3.11e-11)** | 9.01e-01<br>(0.00e+00) | - | **5.64e-01<br>(1.10e-10)** | 1.30e+00<br>(5.20e-01) | - | 7.20e-01<br>(5.79e-07) | **5.64e-01<br>(2.38e-01)** | + |
| f14 | **3.53e+00<br>(1.34e+00)** | 3.86e+00<br>(2.12e-01) | - | **2.59e+00<br>(1.05e-05)** | 3.83e+00<br>(8.76e-01) | - | 4.08e+00<br>(3.88e-01) | **2.59e+00<br>(8.98e-01)** | + |
| f15 | 3.32e+00<br>(4.46e-01) | **2.96e-01<br>(6.25e-02)** | + | 4.00e+00<br>(0.00e+00) | **2.28e+00<br>(1.21e+00)** | + | 4.00e+00<br>(4.80e-01) | 4.00e+00<br>(2.61e+00) | NA |
| f16 | 1.32e+02<br>(1.27e-04) | **1.11e+02<br>(4.95e-04)** | + | 1.43e+02<br>(1.06e-04) | **1.10e+02<br>(0.00e+00)** | + | 1.28e+02<br>(2.82e+00) | **1.10e+02<br>(7.07e-01)** | + |
| f17 | 1.85e+02<br>(0.00e+00) | **1.57e+02<br>(6.36e-07)** | + | 1.73e+02<br>(2.40e-01) | **1.53e+02<br>(0.00e+00)** | + | 2.07e+02<br>(1.55e+01) | **1.53e+02<br>(2.82e+00)** | + |
| f18 | 8.00e+02<br>(1.63e-04) | **3.28e+02<br>(1.97e-05)** | + | 3.98e+02<br>(6.36e-07) | **3.89e+02<br>(5.79e-02)** | + | 3.89e+02<br>(2.90e-03) | **3.07e+02<br>(1.48e-01)** | + |
| f19 | 5.31e+02<br>(2.00e-03) | **3.00e+02<br>(2.62e-06)** | + | 8.00e+02<br>(2.54e-02) | **4.40e+02<br>(8.83e-02)** | + | 4.40e+02<br>(6.43e+00) | **3.15e+02<br>(1.06e+00)** | + |
| f20 | 8.00e+02<br>(2.00e+01) | **3.00e+02<br>(2.02e-19)** | + | 8.00e+02<br>(0.00e+00) | **3.74e+02<br>(2.82e+00)** | + | 8.00e+02<br>(0.00e+00) | **3.70e+02<br>(4.94e-01)** | + |
| f21 | 5.00e+02<br>(0.00e+00) | 5.00e+02<br>(0.00e+00) | NA | 8.00e+02<br>(0.00e+00) | **5.00e+02<br>(0.00e+00)** | + | 8.00e+02<br>(2.12e+01) | **5.00e+02<br>(0.00e+00)** | + |
| f22 | 7.68e+02<br>(0.00e+00) | **8.20e+02<br>(2.40e+00)** | - | 7.82e+02<br>(1.41e+01) | **7.62e+02<br>(1.90e+01)** | + | 7.62e+02<br>(4.24e+00) | **7.89e+02<br>(2.19e+01)** | - |
| f23 | 9.29e+02<br>(1.47e-03) | **5.59e+02<br>(0.00e+00)** | + | 1.18e+03<br>(0.00e+00) | **5.59e+02<br>(0.00e+00)** | + | 1.18e+03<br>(1.77e-02) | **5.59e+02<br>(0.00e+00)** | + |
| f24 | 2.00e+02<br>(0.00e+00) | 2.00e+02<br>(2.33e-17) | NA | 7.54e+02<br>(3.91e-05) | **2.00e+02<br>(0.00e+00)** | + | 2.00e+02<br>(0.00e+00) | 2.00e+02<br>(0.00e+00) | NA |
| f25 | 2.00e+02<br>(1.13e-07) | 2.00e+02<br>(0.00e+00) | NA | 2.00e+02<br>(0.00e+00) | 2.00e+02<br>(0.00e+00) | NA | 2.00e+02<br>(0.00e+00) | 2.00e+02<br>(0.00e+00) | NA |

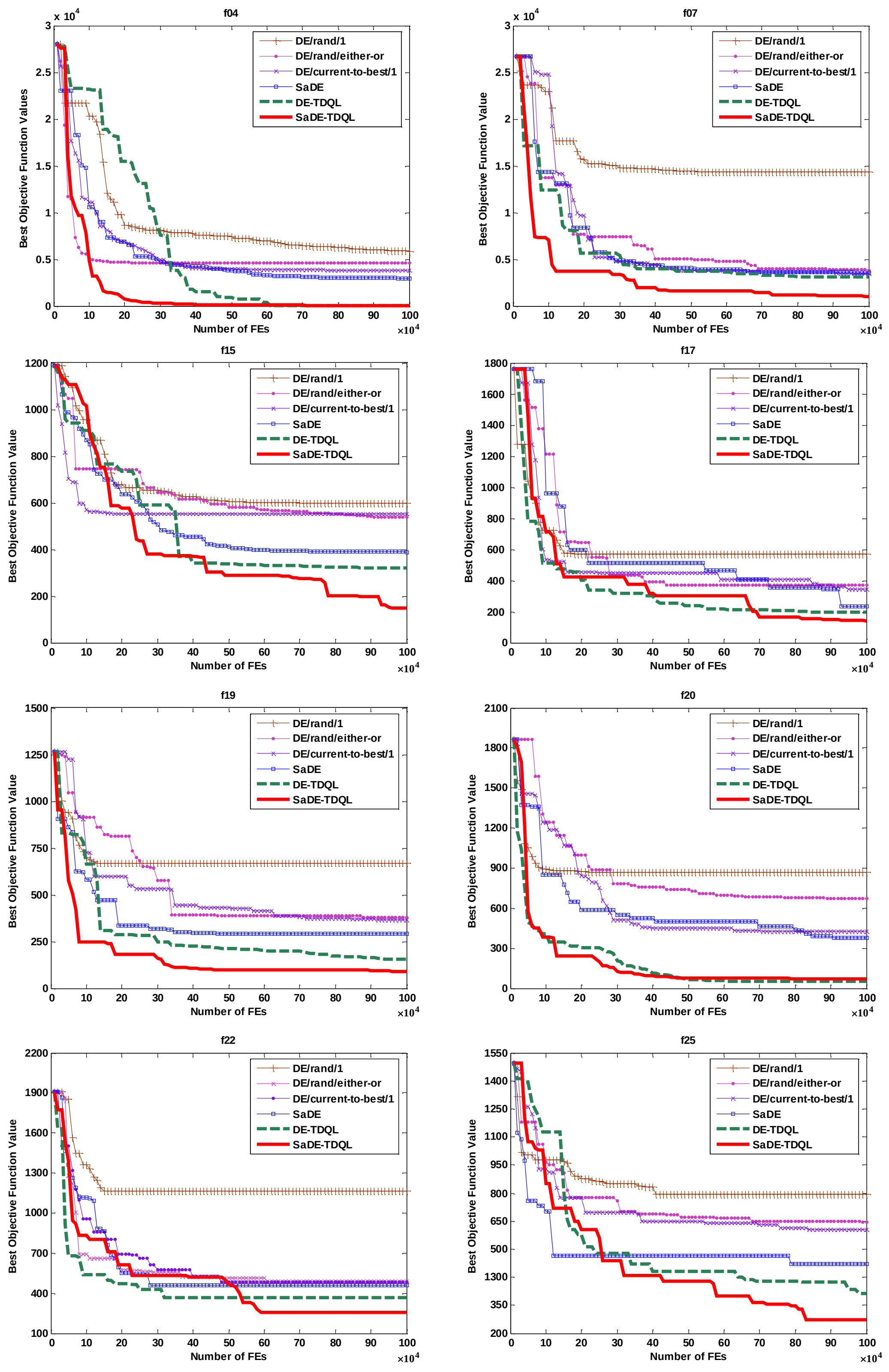


**Fig. 1:** Relative performance in mean best fitness function versus function evaluation for DE-TDQL and SaDE-TDQL over other competitive algorithms: SaDE, DE, PSO, DE for f04, f07, f15, f17, f19, f20, f22 and f25

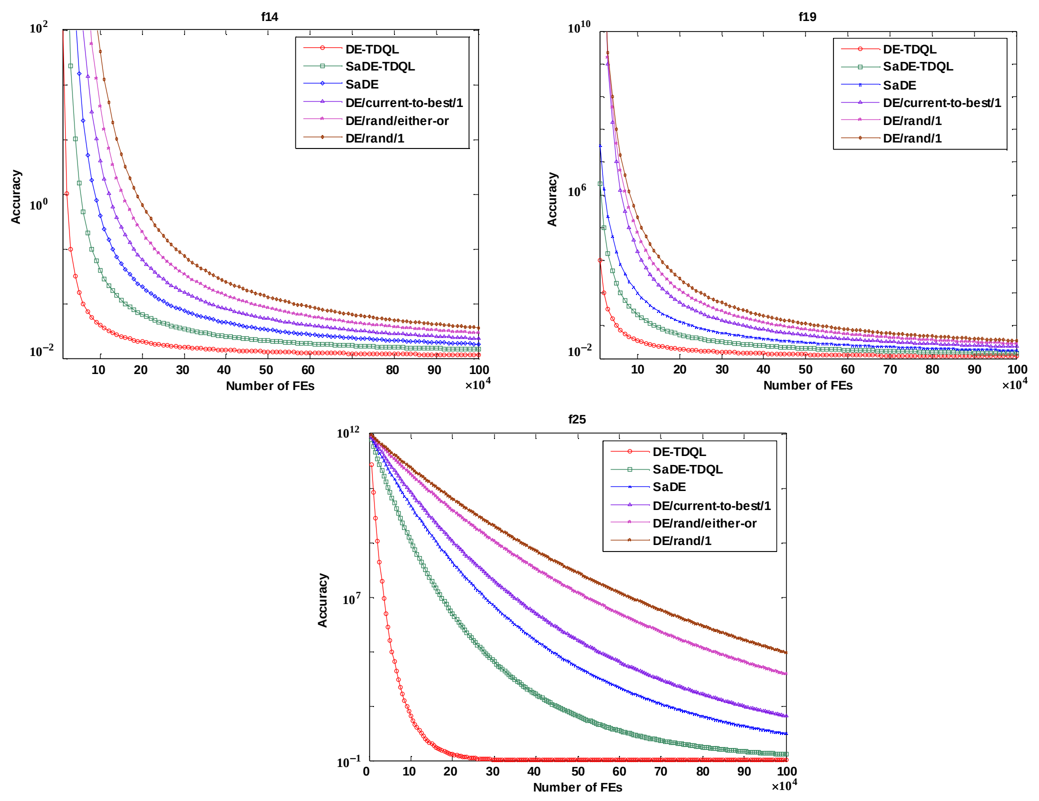


**Fig. 2(a):** Relative performance in accuracy versus function evaluation for DE-TDQL and SaDE-TDQL over other competitive algorithms: SaDE, DE, PSO, DE for f14, f19 and f25 with Max_FEs=$10^6$

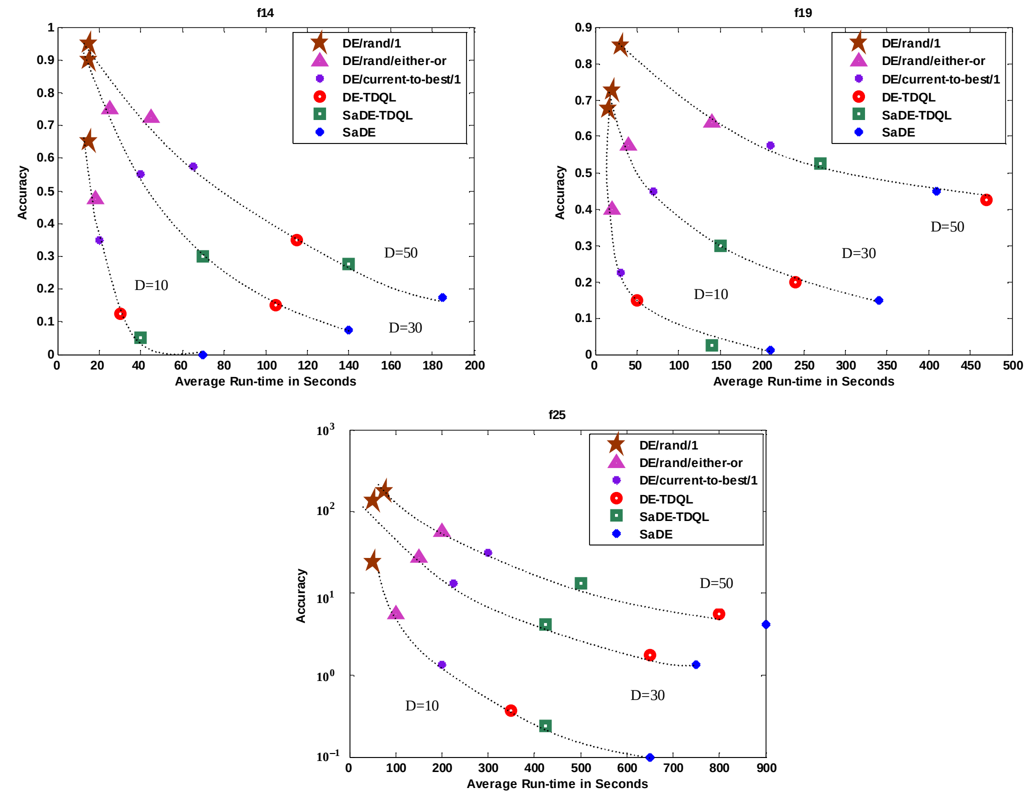


**Fig. 2(b):** Relative performance in accuracy versus average run-time for DE-TDQL and SaDE-TDQL over other competitive algorithms: SaDE, DE, PSO, DE for f14, f19 and f25 with total no. of runs=25 and Max_FEs=$10^6$

## VI. A Case Study In Multi-Robot Path-Planning

Multi-robot path-planning refers to determining the trajectory of motion of robots between pre-defined 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 multi-robot path-planning problem using evolutionary algorithm. Here, we formulated multi-robot 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 multi-robot path-planning are found in the literature [24], [25], [49]. Here, we attempt to solve the problem using distributed approach, particularly for its good time-efficiency [72]. 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-dimensional work-space, 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 the goal coordinates for each robot are fixed in the work-space, 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 the last fields (T) indicate the starting and the goal positions of the mobile robot. The second onwards successive (n-2) fields represent the intermediate nodes. Fig. 4 gives a pictorial representation of a solution encoding a possible path for a mobile robot within the workspace as shown in Fig. 3.

We now propose an evaluation method to first check the feasibility of a path, by detecting 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 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 [49] for a solution representative of a possible path for the i-th robot is given in (13).

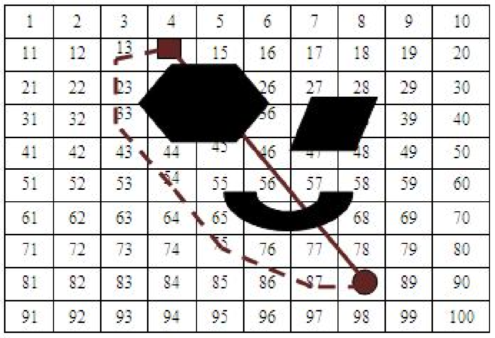


**Fig. 3**: The theoretical and planned paths denoted by solid and dashed line between a given starting and a goal position

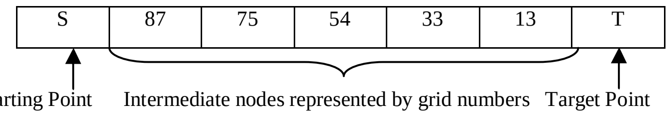


**Fig. 4**: An example of solution in the DE-TDQL-based multi-robot motion planning

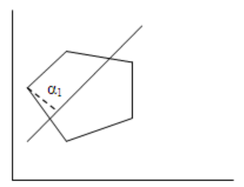
$$F_i = \sum_{j=1}^{N} \left( d_j + \beta_j C \right) \tag{13}$$

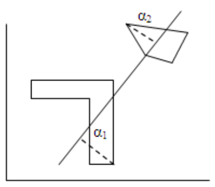
where N is the number of line segments in a path, $d_j$ is the Euclidean distance of the two successive nodes forming the j-th line segment. The factor C is used to maintain uniformity in order of magnitude of the two summations. $\beta_j$ is the coefficient denoting depth of collision, which is defined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

$$\beta_j = \begin{cases} 0 & \text{if j – th line segment is feasible} \\ \sum_{k=1}^{M} \alpha_k & \text{if j – th line segment intersects with obstacles} \end{cases} \tag{14}$$

Here, M is the number of obstacles 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 [49]. 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. So the sum of $\alpha_1$ and $\alpha_2$ is used for calculation of $\beta_j$. For other complex configurations, the reader may consult the paper by Yang [49].





(a) $\beta_j = \alpha_k$  (b) $\beta_j = \alpha_1 + \alpha_2$

Fig. 5: Definition of the coefficient $\beta_j$

## 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.

### B.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 the goal points are pre-defined 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 2 dark obstacles, given starting and goal positions of 6 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.

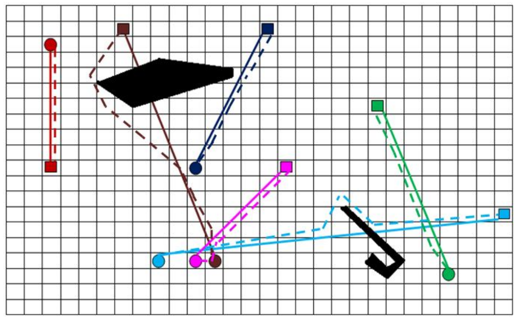


**Fig. 6**: The theoretical and planned paths denoted by solid and dashed line between given starting and goal positions for 6 robots and 2 obstacles

### B.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 8 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 5cm, 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, 25MHz 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.

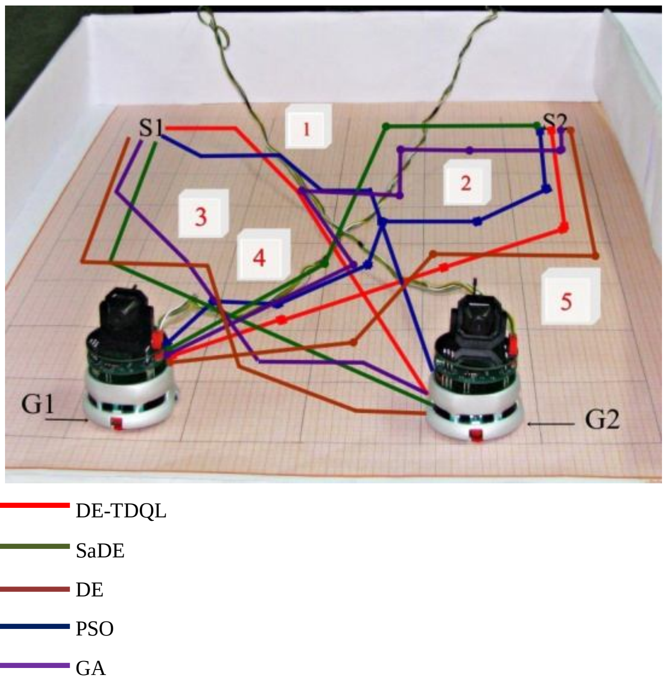


**Fig. 7:** Trajectories planned by execution of different algorithms in Khepera environment with five obstacles

## C. Performance Analysis

To determine a quantitative measure of the relative performance of different algorithms, we use two metrics suggested in [24]–[26]. We here reproduce below the definitions of the two performance metrics for the sake of completeness of the paper.

**Average total path deviation (ATPD) [**24]: Let $P_{ik}$ be a path from the starting point Si to the goal point Gi 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 $\sum_{j=1}^{k} P_{ij} / k$ and the average path deviation for this robot is evaluated by measuring the difference between APT and the ideal shortest path between $S_i$

to $G_i$ (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 $P_{i-ideal} - \sum_{j=1}^{k} P_{ij} / k$ Therefore for n robots in the workspace the Average Total Path Deviation (ATPD) is $\sum_{i=1}^{n} P_{i-ideal} - \sum_{j=1}^{k} P_{ij} / k$ .

**Average Uncovered Target Distance** [26]: Given a goal position $G_i$ and the current position $C_i$ of a robot on a 2-dimensional workspace, where $G_i$ and $C_i$ are 2-dimensional vectors, the uncovered distance of robot i is $\| G_i - C_i \|$, where $\|.\|$ denotes Euclidean norm. For n robots, uncovered target distance (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 Uncovered Target Distance (AUTD). In all our experiments, we set k =10.

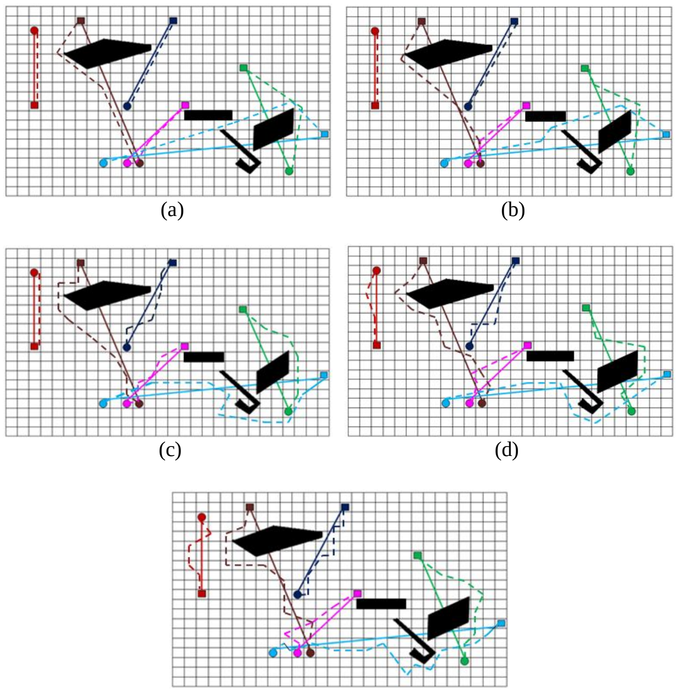


**Fig. 8**: Final configuration of the world map after execution of the (a) DE-TDQL- (b) SaDE- (c) DE- (d) PSO- and (e) GA- based simulations with 6 robots and 2 obstacles requiring 23, 25, 29, 32 and 34 steps respectively.

The performance analysis was undertaken on simulation environment. First we plot APT for n robots, called Average Total Path Traversed (ATPT) by varying n from 2 to 10 by generating paths using 5 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 to the number of robots.

The second study on performance analysis was undertaken by plotting ATPD by generating paths by five different evolutionary algorithms (as used in APT) with number of robots as variable. Fig. 10 provides the results of ATPD computation when number of robots varies between 2 to 10. Here too we observe that DE-TDQL outperforms the remaining four algorithms as ATPD remains the smallest for DE-TDQL irrespective to the no. of robots.

The last analysis on performance was undertaken by comparing AUTD over the no. 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 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.

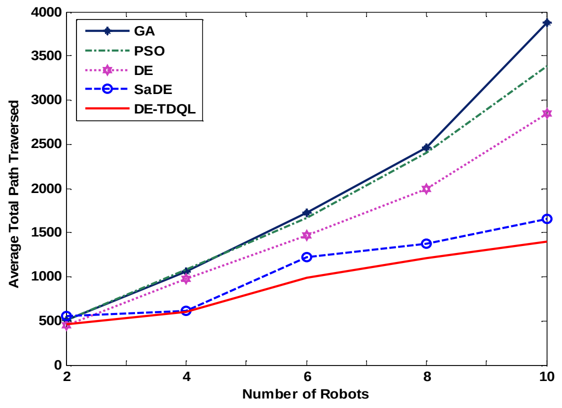


**Fig. 9:** Average total path traversed vs. number of robots with number of obstacles= 5 (constant)

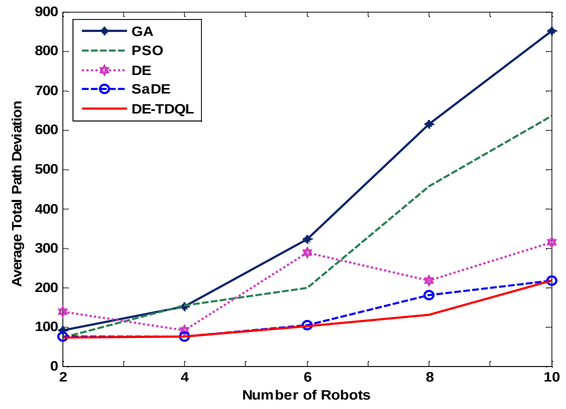


**Fig. 10:** Average total path deviation vs. number of robots with number of obstacles= 5 (constant)

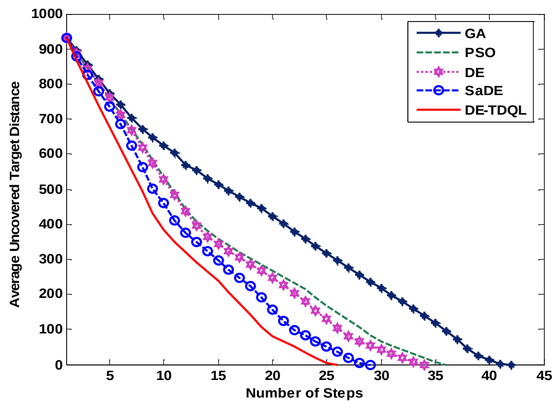


**Fig. 11:** Average uncovered target distance vs. number of steps with number of robots=5 (constant) and obstacles= 5 (constant)

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 other algorithms. Table-IV confirms that DE-TDQL outperforms the remaining four algorithms with respect to all the three metrics.

TABLE-IV
COMPARISON OF NUMBER OF STEPS, AVERAGE PATH TRAVERSED AND AVERAGE TOTAL PATH DEVIATION BY THE ROBOTS

| Algorithms | Total Number of Steps | ATPT (inch.) | ATPD (inch.) |
|---|---|---|---|
| DE-TDQL | **10** | **42.2** | **7.2** |
| SaDE | 12 | 44.9 | 9.9 |
| DE | 17 | 46.4 | 11.4 |
| PSO | 19 | 47.1 | 12.1 |
| GA | 23 | 50.0 | 15.0 |

## VII. CONCLUSION

The paper introduced a new technique for efficiently employing DE and Q-learning together to develop an adaptive memetic algorithm. 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 jointly. A set of 25 CEC 2005 benchmark functions proposed by Suganthan has been used to arrive at the above conclusions.

Besides the above, 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 both in 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 above 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 multi-robot 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 [49], and the DE-TDQL algorithm is employed to minimize the objective function in order 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 here too outperforms 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 real environment, thereby justifying the efficacy of the proposed algorithm.

## REFERENCES

[1]. Richard Dawkins, "The Selfish Gene", *Oxford University Press*, 1976.

[2]. Yew-Soon Ong, Meng-Hiot Lim and Xianshun Chen, "Memetic Computation – Past, present and future." *IEEE Computational Intelligence*, May 2010.

[3]. Y. S. Ong, M. H. Lim, Ning Zhu and Kok-Wai Wong "Classification of Adaptive Memetic Algorithms: A Comparative Study" in *IEEE Trans. on Systems, Man and Cybernetics*, Vol.36, No.1, February 2006.

[4]. P. Cowling, G. Kendall and E. Soubeiga, "A hyperheuristic approach to scheduling a sales Summit", in *PATAT 2000, Springer Lecture Notes in Computer Science*, Konstanz, Germany. Aug 2000, pp 176-190.

[5]. G. Kendall, P. Cowling and E. Soubeiga, "Choice function and random hyperheuristics," in *Proc.4th Asia-Pacific Conference on simulated Evolution and Learning*, Singapore, Nov.2002, pp.667-671.

[6]. Sridhar Mahadevan, "Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results", *Machine Learning , Special Issue on Reinforcement Learning* (edited by Leslie Kaebling), Vol. 22, pp. 159-196, 1996.

[7]. Watkins, C., "Learning from delayed rewards", *PhD dissertation*, King's College, Cambridge, England, 1989.

[8]. Watkins, C. and Dayan, P., "Q-learning", *Machine Learning*, Vol. 8, pp. 279- 292, 1992.

[9]. T. Dean, K. Basye, and J. Shewchuk,."Reinforcement learning for planning and control". In: *Minton, S(ed) Machine Learning Methods for Planning and Scheduling*: Morgan Kaufmann199

[10]. R. Storn and K. V. Price, "Differential Evolution–a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optimization*, Vol. 11, no. 4, pp. 341–359, 1997.

[11]. R. Storn, K. V. Price, and J. Lampinen, "Differential Evolution–A Practical Approach to Global Optimization", *Berlin, Germany: Springer- Verlag*, 2005.

[12]. T. Rogalsky, R. W. Derksen, and S. Kocabiyik, "Differential evolution in aerodynamic optimization," in *Proc. 46th Annu. Conf. Can. Aeronautics Space Inst.*, 1999, pp. 29–36.

[13]. S. Das and A. Konar, "Design of two dimensional IIR filters with modern search heuristics: A comparative study," *Int. J. Comput. Intell. Applicat.* Vol. 6, no. 3, pp. 329–355, 2006.

[14]. J. Lampinen. (1999), "A bibliography of differential evolution algorithm", *Lappeenranta University of Technology. Department of Information Technology, Laboratory of Information Processing, Tech. Report [Online]*. Available: http://www.lut.fi/jlampine/debiblio.htm

[15]. M. Omran, A. P. Engelbrecht, and A. Salman, "Differential evolution methods for unsupervised image classification," in *Proc. 7th Congr. Evol. Comput. (CEC-200*5), Vol. 2. Piscataway, NJ: IEEE Press, pp. 966–97

[16]. S. Das, A. Abraham, and A. Konar, "Adaptive clustering using improved differential evolution algorithm," *IEEE Trans. Syst., Man, Cybern. A*, Vol. 38, no. 1, pp. 218–237, Jan. 2008.

[17]. J. Lampinen, and I. Zelinka, "On stagnation of the differential evolution algorithm," in Proc. MENDEL 2000", *6th Int.Mendel Conf. Soft Computing*, Brno, Czech Republic, Jun. 2000, pp. 76–8

[18]. J. Ronkkonen, S. Kukkonen, and K. V. Price, "Real parameter optimization with differential evolution," in *Proc. IEEE Congr. Evol. Comput. (CEC-2005)*, Vol. 1. Piscataway, NJ: IEEE Press, pp. 506–51

[19]. E. Mezura-Montes, J. Velázquez-Reyes, and C. A. C. Coello, "A comparative study of differential evolution variants for global optimization," in *Proc. Genetic Evol. Comput. Conf. (GECCO 2006)*, pp. 485–492.

[20]. U. K. Chakraborty, S. Das, and A. Konar, "Differential evolution with local neighborhood," in *Proc. IEEE Congr. Evol. Comput. (CEC-2006)*, Piscataway, NJ: IEEE Press, pp. 7395–7402.

[21]. K. V. Price, "An introduction to differential evolution," in *New Ideas Optimization. London*, U.K.: McGraw-Hill, 1999, pp. 293–298.

[22]. P.N. Suganthan, N. Hansen , J.J. Liang, K. Deb, Y. P. Chen A. Auger, S. Tiwari, "Problem Definitions and Evalution Criteria for The CEC 2005".

[23]. J. Chakraborty., A. Konar., L. C. Jain, and U. Chakraborty, "Co-operative Multi Robot Path Planning Using Differential Evolution" *Journal of Intelligent & Fuzzy systems*, Vol 20, pp.13-27, 2009.

[24]. J. Chakraborty and S. Saha, "Co-operative Multi-robot Path Planning Using Particle Swarm Optimization," in Proc. of *IEEE WIE National Symposium on Emerging Technologies*, 2007.

[25]. J. Chakraborty and A. Konar, "A Distributed Multi –Robot Path Planning Using Particle Swarm Optimization," in *2nd National Conference on Recent Trends in Information Systems*, pp216-221, 2008.

[26]. J. Chakraborty, A. Konar, U. K. Chakraborty and L. C. Jain, "Distributed Co-operative Multi Robot Path Planning Using Differential Evolution" in *IEEE Congress on Evolutionary Computation*, 2009.

[27]. D. Fogel and H-G. Beyer, "A Note on the Empirical Evaluation of Intermediate Recombination", *Evolutionary Computations*, 3(4), pp-491-495.

[28]. A. K. Qin and P. N. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization", *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, Vol. 2, pp. 1785–1791, 2005.

[29]. J. Brest, B. Boˇskovi´c, S. Greiner, V. ˇZumer, and M. Mauˇcec, "Performance comparison of self-adaptive and adaptive differential evolution algorithms", *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, Vol. 11, no. 7, pp. 617–629, 2007.

[30]. Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman, "Pac model-free reinforcement learning", in *Proc. 23nd ICML 2006*, pp. 881–888, 2006.

[31]. D. Pandey, and P. Pandey, "Approximate Q-Learning: An Introduction", in *Machine Learning and Computing (ICMLC), Second International Conference*, 2010, pp. 317 – 320.

[32]. T. Murata, and M. Yamaguchi, "Multi-Legged Robot Control Using GA-Based QLearning Method With Neighboring Crossover", in *Frontiers in Evolutionary Robotics*, Iba (Ed.), pp. 341-352, I-Tech Education and Publishing, ISBN 9783902613196, Vienna, Austria, 2008.

[33]. T. Murata, and Y. Aoki, "GA-Based Q-Learning to Develop Compact Control Table for Multiple Agents", in *New Achievements in Evolutionary Computation*, InTech, February 2010.

[34]. Z. Zhu, Z. Ji, and S. Jia, "Memetic Ant Colony Optimization for Band Selection of Hyperspectral Imagery Classification", in *CCPR 2010*, pp. 1012-1017, October 21-23, Chongqing, China.

[35]. L. Shu, and H. Iba, "A study on the computational efficiency of Baldwinian evolution", in *Nature and Biologically Inspired Computing (NaBIC), Second World Congress*, pp. 467 – 472, February 2011.

[36]. A. M. Farahmand, M. N. Ahmadabadi, C. Lucas, and B. N. Araabi, "Interaction of Culture-Based Learning and Cooperative Co-Evolution and its Application to Automatic Behavior-Based System Design", in *Evolutionary Computation, IEEE Transactions*, Vol. 14, pp. 23-57, January, 2010.

[37]. S. Wei, Y. Wang, Y. Yang, F. Yin, W. Cao, and Y. Tang, "Applying Q-Learning Algorithm to Study Line-Grasping Control Policy for Transmission Line Deicing Robot", in *Intelligent System Design and Engineering Application (ISDEA)*, pp. 382 – 387, 2010.

[38]. M. Simsek, A. Czylwik, A. Galindo-Serrano, and L. Giupponi, "Improved decentralized Q-learning algorithm for interference reduction in LTE-femtocells", in *Wireless Advanced (WiAd)*, pp. 138 – 143, 2011.

[39]. D. Osmankovic, and S. Konjicija, "Implementation of Q-Learning algorithm for solving maze problem", in *MIPRO, Proceedings of the 34th International Convention*, pp. 1619 – 1622, July, 2011.

[40]. W. Liu, Y. Tan, and Q. Qiu, "Enhanced Q-learning algorithm for dynamic power management with performance constraint", in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 602 – 605, April, 2010.

[41]. Y. Ong, M. Lim, N. Zhu, and K. Wong, "Classification of adaptive memetic algorithms: a comparative study", in S*ystems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions*, February, 2006.

[42]. N. Shahidi, H. Esmaeilzadeh, M. Abdollahi, E. Ebrahimi, and C. Lucas, "Self-adaptive memetic algorithm: an adaptive conjugate gradient approach", in *Cybernetics and Intelligent Systems, IEEE Conference*, Vol. 1, pp. 6 – 11, July, 2005.

[43]. S. Das, A. Abraham, U. K. Chakraborty, and A. Konar, "Differential Evolution Using a Neighborhood-Based Mutation Operator" by *IEEE Transactions on Evolutionary Computation*, Vol. 13, No. 3, pp. 526-553, June 2009.

[44]. I. Goswami (Chakraborty), P. Kumar Das, A. Konar and R. Janarthanan, "Conditional Q-learning Algorithm for Path-Planning of a Mobile Robot," *2010 International Conference on Industrial Electronics, Control and Robotics*, pp. 23 – 27, Dec. 27-29, 2010.

[45]. P. Bhattacharjee, P. Rakshit, I. Goswami, A. Konar, A. K. Nagar, "Multi-Robot Path-Planning Using Artificial Bee Colony Optimization Algorithm". *NaBIC 2011*: 219-224.

[46]. J. J. Liang, P. N. Suganthan, and K. Deb, "Novel composition test functions for numerical global optimization," in *Proc. IEEE Swarm Intell. Symp*. Pasadena, CA, Jun. 2005, pp. 68–75.

[47]. A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization", in *IEEE Transactions on Evolutionary Computation*, Vol. 13, NO. 2, APRIL 2009, pp- 398 – 417.

[48]. P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. -P. Chen, A. Auger, and S. Tiwari, "Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization", in *Technical Report. 2005. Nanyang Technological University*, Singapore, May 2005 AND KanGAL Report #2005005, IIT Kanpur, India.

[49]. S. X. Yang, Y. Hu, and M. Q. H. Meng, "A Knowledge Based GA for Path Planning of Multiple Mobile Robots in Dynamic Environments", *IEEE Conference on Robotics, Automation and Mechatronics*, June 2006, pp. 1-6.

[50]. C. H. Papadimitriou, P. Raghavan, M. Sudan, and H. Tamaki, "Motion Planning on a Graph", Proceedings of STOC, 1994.

[51]. G. Swaminathan, "Robot Motion Planning", February 2006.

[52]. C. L. Shih, T. T. Lee, and W. A. Gruver, "Motion-Planning with Time-Varying Polyhedral Obstacles based on Graph Search and Mathematical Programming", *IEEE International Conference on Robotics and Automation*, 1990, pp. 331-337.

[53]. R. Glasius, A. Komoda, and S. Gielen, “Neural Network Dynamics for Path Planning and Obstacle Avoidance”, *Neural Networks*, March 1994.

[54]. P. Payeur, H. L. Huy, and C. Gosselin, “Robot Path Planning using Neural Networks and Fuzzy Logic”, *International Conference on Industrial Electronics, Control and Instrumentation, September*, 1994.

[55]. I. Gupta, and D. Riordan, “Path Planning for Mobile Robots using Fuzzy Logic”,*www.unbsj.ca/conferences/apics/2004/documents/GuptaRiordan NEW.doc*, March 3, 2008.

[56]. I. Hassanzadeh, and S. M. Sadigh, “Path Planning for a Mobile Robot using Fuzzy Logic Controller Tuned by GA”, *International Symposium on Mechatronics and its Applications*, March 2009.

[57]. Z. Michalewicz, L. Zhang, and K. Trojanowsky, “Adaptive Evolutionary Planner/ Navigator for Mobile Robots”, *IEEE Transactions on Evolutionary Computation*, April, 1997.

[58]. T. Shibata, and T. Fukuda, "Intelligent Motion Planning by Genetic Algorithm with Fuzzy Critic", *Proc. 8th IEEE Int. Symp. Intelligent Control*, 1993, pp.565 -570.

[59]. C. C. Lin, K. C. Chen, and W. J. Chuang, “Motion Planning using a Memetic Evolution Algorithm for Swarm Robots”, *International Journal of Advanced Robotic Systems*, 2012, Vol. 9, pp. 1-9.

[60]. T. Balch, and R. C. Arkin, “Behavior-based Formation Control for Multirobot Teams”, *IEEE Transactions on Robotics and Automations*, 1998, Vol. 14, pp. 926-939.

[61]. T. Mitchell, “Machine Learning”, *McGraw Hill*, 1997.

[62]. A. Ghandar, Z. Michalewicz, R. Zurbruegg, “Intelligent Decision Support: A Fuzzy Stock Ranking System”, *Aspects of Natural Language Processing*, 2009, pp. 379-410.

[63]. J. Coche, “An Evolutionary Approach to the Examination of Capital Market Efficiency”, *Evolutionary Economics*, Vol. 8, pp. 357-382.

[64]. R. Luna, and K. E. Bekris, “Efficient and Complete Centralized Multi-Robot Path Planning”, *Proceedings of the Fourth International Symposium on Combinatorial Search*, 2011.

[65]. P. Bhattacharya, and M. L. Gavrilova, “Roadmap-Based Path Planning Using the Voronoi Diagram for a Clearance-Based Shortest Path”, *IEEE Robotics and Automation Magazine*, June, 2008.

[66]. R. Gayle, W. Moss, M. C. Lin, and D. Manocha, “Multi-Robot Coordination using Generalized Social Potential Fields”, *IEEE International Conference on Robotics and Automation*, 2009.

[67]. M. Garber and M. Lin, “Constraint-based Motion Planning Using Voronoi Diagrams,” *Proc. Fifth International Workshop on Algorithmic Foundations of Robotics*, 2002.

[68]. K. S. Senhilkumar, and K. K. Bharadwaj, “A Evolutionary Approach for Multi-Robot Path Exploration Problem”, *Proceedings of the International MultiConference of Engineers and Computer Scientists*, Hong Kong, 19-21 March, 2008, Vol. 2.

[69]. S. Senhilkumar, “An Efficient Global Optimization Approach to Multi-Robot Path Exploration Problem using Hybrid Genetic Algorithm”, *4th International Conference on Information and Automation for Sustainability*, 2008.

[70]. X. Ma, Q. Zhang, and Y. Li, “Genetic Algorithm-based Multi-robot Cooperative Exploration”, *IEEE International Conference on Control and Automation 2007*, Vol. 00, pp. 1018-1023.

[71]. X. Ma, Q. Zhang, W. Chen, and Y. Li, “Immunity-Based Adaptive Genetic Algorithm for Multi-robot Cooperative Exploration”, *Advanced Intelligent Computing Theories and Applications With Aspects of Artificial Intelligence, Springer*, 2007, pp. 605–616.

[72]. L. E. Parker, “Path planning and motion coordination in multiple mobile robot teams”, in *R. A. Meyers (ed.), Encyclopedia of Complexity and System Science, Springer*, Knoxville, Tennessee, USA, Chapter 13, pp. 5783–5800.

[73]. U.K. Chakraborty, “Advances in Differential Evolution”, *Springer, Heidelberg*, New York, 2008.

[74]. P. J. Angeline, “Evolutionary optimization versus particle swarm optimization: Philosophy and the performance difference,” in *Proc. 7th Int. Conf. Evol. Programming- Evol. Programming* VII, LNCS, vol. 1447, 1998, pp. 84-89.