An Adaptive Memetic Algorithm for Multi-robot Path-Planning

Pratyusha Rakshit, Dhrubojyoti Banerjee, Amit Konar, and Ramadoss Janarthanan

ETCE Depratment, Jadavpur University, India
pratyushar1@gmail.com,
{dhrubo_jyoti_banerjee,konaramit}@yahoo.co.in,
srmjana_73@yahoo.com

Abstract. 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. 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 to two standard metrics defined in the literature.

1 Introduction

Coined by Dawkins in 1976, the word "meme" refers to the basic unit of cultural transmission or imitation. Memetic Algorithms (MAs) are population-based meta-heuristic search algorithms that combine the composite benefits of natural and cultural evolution. MA captures the power of global search by its evolutionary component and local search by its cultural component.

The class of algorithms incorporating the adaptive selection of memes is referred to as Adaptive MA (AMA). Usually, the selection of the meme for an individual member of the population is done based on its ability to perform local improvement. The variant of AMA we would use in this paper is Roulette-Choice strategy based Hyperheuristic AMA [1]. 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 Differential Evolution (DE) for global search and a reinforcement learning algorithm for local refinement. The particular version of the DE (DE/current-to-best/1) used here has two parameters called scaling factors, which are adaptively selected from a meme pool. 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

B.K. Panigrahi et al. (Eds.): SEMCCO 2012, LNCS 7677, pp. 248–258, 2012.

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 [2]. 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 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 proposed AMA has successfully been employed in multi-robot path-planning. It refers to online trajectory planning from given starting positions to fixed goal positions for each robot without hitting teammates and obstacles [3], [4]. 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. 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, which has a high order of computational complexity, now boils down to relatively simplified problem of minimization of n objective functions for n robots. In both the cases, the above problem is solved by iteratively identifying next positions of the robots until the goal position for all the robots are reached.

This paper considers an evolutionary path-planning with robots of definite size and circular cross-section and tested both in simulation and real environments, 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 [4] have been used here to compare the relative merits of the proposed AMA with respect to Genetic Algorithm (GA) based realization given in [3]. Experiments 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 five sections. Section 2 provides an overview of the classical Q-learning. In section 3, we propose the AMA realized with DE and TDQL. Section 4 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 5.

2 An Overview of the Classical Q-Learning

Let $S = \{s_1, s_2, ..., s_n\}$ be a set of states of an agent, $A = \{a_1, a_2, ..., a_n\}$ be a set of actions that the agent can select in each state $s_i \in S$, $r(s_i, a_j)$ and $Q(s_i, a_j)$ be the immediate and total 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., $s_k = \delta(s_i, a_j)$, γ be the discounting factor used to penalize the future reward after a delay of k units by scaling it by a factor γ^k for positive integer k.

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, which is approximated as $\underset{a'}{Max}\, Q(s, a')$. Thus,

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

In the classical Q-learning for deterministic state-transition, 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 δ-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.

Differential Q-learning is a modified version of 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

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

The learning rate α determines to the extent the newly acquired information will override the old information. A setting of $\alpha = 0$ makes the agent stop learning, while $\alpha = 1$ 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.

3 Proposed Adaptive Memetic Algorithm

The AMA to be proposed shortly includes a DE for global exploration and a TDQL for adaptive selection of memes. The process of adaptive selection of scaling factors F_1'

and F_2' as in (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 row indices of the Q-table represent states S_1, S_2, …, 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, …, 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(.,.). Principles used in designing the AMA are introduced below.

1. Initialization: DE-TDQL starts with a population of NP D-dimensional parameter vectors within the prescribed minimum and maximum bounds: $\vec{X}_{\min} = \{x_{\min-1}, x_{\min-2}, \ldots, x_{\min-D}\}$ and $\vec{X}_{\max} = \{x_{\max-1}, x_{\max-2}, \ldots, x_{\max-D}\}$. Hence, we may initialize the j-th component of the i-th vector at generation G=0 as

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

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 probability of selection of $F=F_j$ from the meme pool $\{F_1, F_2, \ldots, F_{10}\}$ is given by

$$P(F_j) = Q(S_i, 10F_j) / \sum_{l=1}^{10} Q(S_i, 10F_l) \tag{4}$$

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:

$$\sum_{m=1}^{j-1} P(F=F_m) < r \le \sum_{m=1}^{j} P(F=F_m) \Rightarrow \frac{\sum_{m=1}^{j-1} Q(S_i, 10F_m)}{\sum_{l=1}^{10} Q(S_i, 10F_l)} < r \le \frac{\sum_{m=1}^{j} Q(S_i, 10F_m)}{\sum_{l=1}^{10} Q(S_i, 10F_l)} \tag{5}$$

4 Differential Evolution (DE/Current-to-Best/1)

4.1 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

$$\vec{V}_i(G) = \vec{X}_i(G) + F_1'(\vec{X}_{best}(G) - \vec{X}_i(G)) + F_2'(\vec{X}_{rand-1}(G) - \vec{X}_{rrand-2}(G)) \quad (6)$$

and F'_1 and F'_2 are two scaling factors selected from the meme pool adaptively by step 2 before invoking the DE process in [0, 1].

4.2 Crossover

There are two types of crossover (recombination) schemes- binomial and exponential [2]. In the proposed realization we have used only binomial crossover. Here, a trial vector $\vec{U}_i(G)$ is generated for each pair of $\vec{V}_i(G)$ and $\vec{X}_i(G)$ by (5)

$$u_{i,j}(G) = \begin{cases} v_{i,j}(G) \; if \; rand_{ij} \le Cr \; or \; j = j_{rand} \; (j_{rand} \in [1, D]) \\ x_{i,j}(G) \; otherwise \end{cases} \quad (7)$$

where $rand_{i,j}(0,1) \in [0, 1]$ is a uniformly distributed random number lying in [0,1].

4.3 Selection

Here, for a given objective $f(\vec{x})$ to be minimized, we have

$$\begin{aligned} \vec{X}_i(G+1) &= \vec{U}_i(G) \quad if \; f(\vec{U}_i(G)) \le f(\vec{X}_i(G)) \\ &= \vec{X}_i(G) \quad if \; f(\vec{U}_i(G)) > f(\vec{X}_i(G)) \end{aligned} \quad (8)$$

5 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 for all r=1 to NP.

6 Reward/Penalty Based Q-Table Updating

If the fitness of the member increases due to transition from S_i to S_k on selection of F_j, then $Q(S_i, F_j)$ will be updated following (9) with a positive reward function: reward(S_i, $10F_j$) = increase in fitness of the member,

$$Q(S_i,10F_j) = (1-\alpha)Q(S_i,10F_j) + \alpha(reward(S_i,10F_j) + \gamma \max_{F'} Q(S_k,10F')) \quad (9)$$

else $Q(S_i, 10F_j)$ will be evaluated by (9) with a negative reward= -K, of constant value, however, small.

7 Convergence

After each evolution, we repeat from step-2 until the termination condition is satisfied.

Pseudo Code of AMA

I. Set the generation number $t = 0$ and randomly initialize a population of NP individuals $P_t = \{\vec{X}_1(t), \vec{X}_2(t),..., \vec{X}_{NP}(t)\}$ with $\vec{X}_i(t) = \{x_{i,1}(t), x_{i,2}(t),..., x_{i,D}(t)\}$ for i=[1, NP] and each individual uniformly distributed in the range $[\vec{X}_{\min}, \vec{X}_{\max}]$ as given in (3) .Set α=0.25, γ=0.8. Evaluate $f(\vec{X}_i(t))$, for i= [1, 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 $[Q(r_i(0), j)] = 1,\ j = [1,10]$ .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**

Initialize $[reward(r_i(t), j)] = 0,\ r_i = [1, NP],\ j = [1,10]$.

For i=1 to NP **do**

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 $\vec{V}_i(t)$ corresponding to $\vec{X}_i(t)$ by (6).

II.c. **Crossover:** Generate trial vector $\vec{U}_i(t)$ for $\vec{X}_i(t)$ as in (7). Evaluate $f(\vec{U}_i(t))$.

II.d. **Selection:**

If $f(\vec{U}_i(t)) < f(\vec{X}_i(t))$ **Then do**

$reward(r_i(t), 10 \times F_{r_i}) = f(\vec{X}_i(t)) - f(\vec{U}_i(t)); \vec{X}_i(t+1) = \vec{U}_i(t);$

Evaluate $f(\vec{X}_i(t+1))$;

If $f(\vec{U}_i(t)) < f(\vec{X}_{best}(t))$ **Then** $\vec{X}_{best}(t) = \vec{U}_i(t)$; Evaluate $f(\vec{X}_{best}(t))$;
End If;

Else $reward(r_i(t), 10 \times F_{r_i}) = -K; \vec{X}_i(t+1) = \vec{X}_i(t);$

End If;
End For;

II.e. Determine the ranked population be $R(t+1) = [r_1(t+1), r_2(t+1), ..., r_{NP}(t+1)]$ in the (t+1)-th generation, 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**
For F_j=0.1 to 1.0 **do**
If $reward(r_i(t), 10 \times F_j) \neq 0$ **Then**

$$Q(r_i(t), 10 \times F_j) = (1-\alpha) Q(r_i(t), 10 \times F_j) + \alpha [reward(r_i(t), 10 \times F_j) + \gamma \max_F Q(r_i(t+1), 10 \times F)];$$

End If;
End For;
End For;
Increase the counter value $t = t + 1$.
End While;

8 Realization of Multi-Robot Path-Planning Using DE-TDQL

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 grids are referred to by their distinct integer addresses. 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. 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.

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. 1 and 2).

We now propose an evaluation method to check the feasibility of a path. 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. A cost function [3] for a solution representative of a possible path for the i-th robot is given as

$$F_i = \sum_{j=1}^{N} \left(d_j + \beta_j C \right) \tag{10}$$

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. β_j is the coefficient denoting depth of collision, which is defined as

$$\beta_j = \begin{cases} 0 & if \;\; j-th \; line \;\; segment \;\; is \;\; feasible \\ \sum_{k=1}^{M} \alpha_k & if \;\; j-th \; line \;\; segment \;\; intersects \;\; with \;\; obstacles \end{cases} \tag{11}$$

Here, M is the number of obstacles the j-th line segment intersects. α_k is defined as the shortest moving distance for escaping the intersected obstacle [3]. 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.

We now illustrate the measurement of α_k and β_j in Fig. 3. In Fig. 3(a), α_k is treated as the shortest distance to move the line out of the obstacle k. Fig. 3(b) 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 α_1 and α_2 is used for calculation of β_j. For other complex configurations, the reader may consult the paper by Yang [3].

B. Experiments: The experiments were undertaken in two phases.

B.1. Experiments in Simulated Environment

Experiments were performed with n ($2 \leq n \leq 14$) similar soft-bots of circular cross section (radius=6 pixels) on a Pentium machine. 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. Extensive experiments were performed with 50 world maps of diverse configurations. We set no. of runs for each experiment, k =10.

B.2. Experiments in Real Environment on Khepera- II Platform

The experiment was undertaken with a world map of 8×6 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 robots were used to sense obstacles around them in the world map and turn wheels by motor firing for controlled movement in prescribed directions. The robots were controlled by two Pentium-IV personal computers (PCs) through wired connections. 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. Two sample runs of path-planning in the real environment is given in Fig. 4. It is observed that the robots follow the shortest paths avoiding collision with obstacles. The experiment was repeated 50 times 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. Results of the experiments performed are summarized in Table-1. Three performance metrics, namely 1) total number of steps taken to reach the goal, 2) ATPT, and 3) ATPD, defined in next sub-section, have been used here too to determine the relative merits of DE-TDQL over other algorithms. Table-1 confirms that DE-TDQL outperforms the remaining four algorithms with respect to all the three metrics.

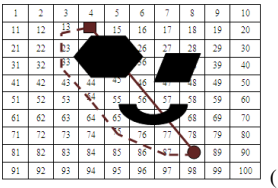


Fig. 1. The theoretical (solid line) and planned paths (dashed line) between given starting and goal positions

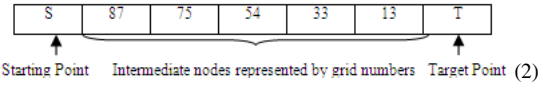


Fig. 2. An example of solution in the DE-TDQL-based multi-robot motion planning

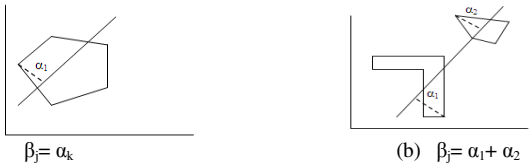


Fig. 3. Definition of the coefficient β_j

C. Performance Analysis

To determine a quantitative measure of the relative performance of different algorithms, we use two metrics suggested in [4].

Average Total Path Deviation (ATPD): 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. For n robots in the workspace the Average Total Path Deviation (ATPD) is $\sum_{i=1}^{n}(P_{i-ideal} - \sum_{j=1}^{k} P_{ij} / k)$ where $P_{i-ideal}$ is the theoretical shortest ideal path from given starting to goal position of i-th robot.

Average Uncovered Target Distance: Given a goal position G_i and the current position C_i of a robot on a 2-dimensional workspace uncovered target distance $UTD = \sum_{i=1}^{n} \|G_i - C_i\|$. Now, for k runs of the program, we evaluate the average of UTDs and call it the Average Uncovered Target Distance (AUTD).

In Fig. 6 we plot ATPT by varying no. of robots using 5 different algorithms, including GA, PSO, DE/current-to-best/1, SaDE and DE-TDQL. The second study on performance analysis was undertaken by plotting ATPD by generating paths by five

different evolutionary algorithms (as used in ATPT) with no. of robots as variable (Fig. 7). From these two figures, we observe that DE-TDQL outperforms the remaining four algorithms as both ATPT and ATPD remain 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. 8). It is apparent from Fig. 8 that DE-TDQL returns the smallest no. of steps required to reach the goal. In brief, the proposed DE-TDQL based path-planning outperforms all the four other algorithms with respect to all three metrics.

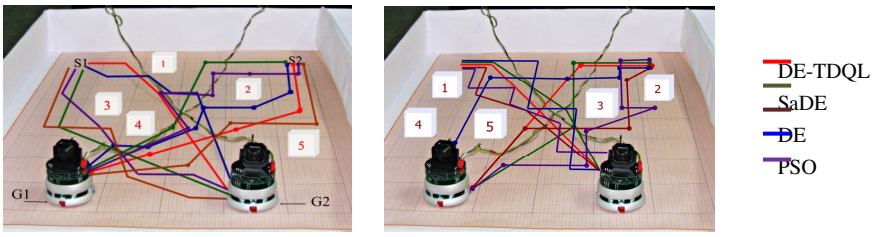


Fig. 4. Trajectories planned by execution of different algorithms in Khepera environment with five obstacles

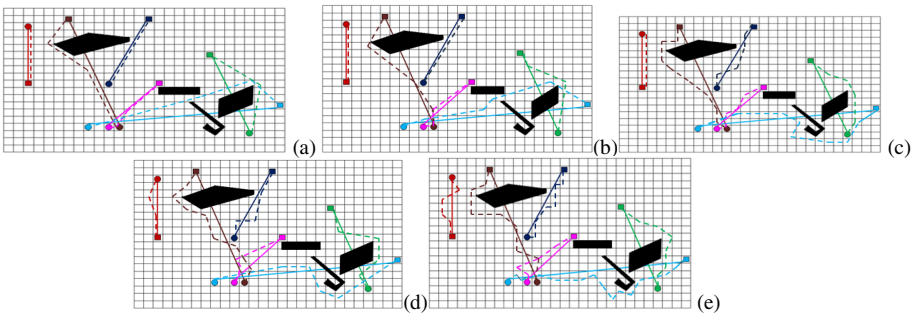
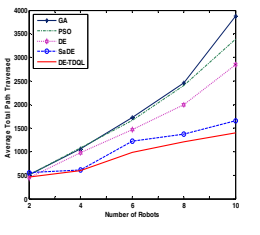
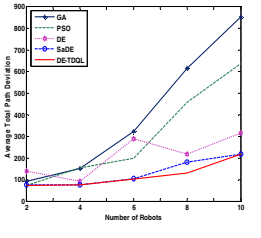


Fig. 5. Final configuration of the world map after execution of the (a) DE-TDQL (b) SaDE (c) DE (d) PSO and (e) GA based simulations with 6 robots and 2 obstacles requiring 23, 25, 29, 32 and 34 steps respectively



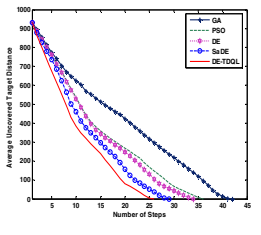
(6)

Fig. 6. Average total path traversed vs. number of robots with number of obstacles= 5



(7)

Fig. 7. Average total path deviation vs. number of robots with number of obstacles= 5



(8)

Fig. 8. Average uncovered target distance vs. number of steps with number of robots=5 and obstacles= 5

Table 1. Comparison of Number of Steps, ATPT and ATPD By the Robots

	World-map-I			World-map-II		
Algorithms	Number of Steps	ATPT (inch.)	ATPD (inch.)	Number of Steps	ATPT (inch.)	ATPD (inch.)
DE-TDQL	**10**	**42.2**	**7.2**	**12**	**43.2**	**8.2**
SaDE	12	44.9	9.9	15	46.1	11.1
DE	17	46.4	11.4	18	46.8	11.8
PSO	19	47.1	12.1	21	48.6	13.6
GA	23	50.0	15.0	23	50.3	15.3

9 Conclusion

The paper introduced a new technique for efficiently employing DE and Q-learning together to develop an adaptive memetic algorithm. A case study on multi-robot path-planning problem has been undertaken to demonstrate the importance of the proposed DE-TDQL algorithm. A formulation of the objective function for the problem has been given following [3], 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 in real environment with Khepera-II mobile robots give the same results as in simulated environment, thereby justifying the efficacy of the algorithm free from system-level implementation.

References

1. Cowling, P., Kendall, G., Soubeiga, E.: A Hyperheuristic Approach to Scheduling a Sales Summit. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 176–190. Springer, Heidelberg (2001)
2. Storn, R., Price, K.V.: Differential Evolution–a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optimization 11(4), 341–359 (1997)
3. Yang, S.X., Hu, Y., Meng, M.Q.H.: A Knowledge Based GA for Path Planning of Multiple Mobile Robots in Dynamic Environments. In: IEEE Conference on Robotics, Automation and Mechatronics, pp. 1–6 (June 2006)
4. Chakraborty, J., Saha, S.: Co-operative Multi-robot Path Planning Using Particle Swarm Optimization. In: Proc. of IEEE WIE National Symposium on Emerging Technologies (2007)
5. Lin, C.C., Chen, K.C., Chuang, W.J.: Motion Planning using a Memetic Evolution Algorithm for Swarm Robots. International Journal of Advanced Robotic Systems 9, 1–9 (2012)
6. Gayle, R., Moss, W., Lin, M.C., Manocha, D.: Multi-Robot Coordination using Generalized Social Potential Fields. In: IEEE International Conference on Robotics and Automation (2009)
7. Bhattacharjee, P., Rakshit, P., Goswami, I., Konar, A., Nagar, A.K.: Multi-Robot Path-Planning Using Artificial Bee Colony Optimization Algorithm. In: NaBIC 2011, pp. 219–224 (2011)