Intel Serv Robotics (2017) 10:121–136
DOI 10.1007/s11370-017-0217-x

ORIGINAL RESEARCH PAPER

Path planning of modular robots on various terrains using Q-learning versus optimization algorithms

Sajad Haghzad Klidbary[1] · **Saeed Bagheri Shouraki**[1] · **Soroush Sheikhpour Kourabbaslou**[1]

Received: 11 August 2016 / Accepted: 10 January 2017 / Published online: 24 January 2017

Abstract Self-reconfigurable modular robots (SRMRs) have recently attracted considerable attention because of their numerous potential applications in the real world. In this paper, we draw a comprehensive comparison among five different algorithms in path planning of a novel SRMR system called ACMoD through an environment comprised of various terrains in a static condition. The contribution of this work is that the reconfiguration ability of ACMoD has been taken into account. This consideration, though raises new algorithmic challenges, equips the robot with new capability to pass difficult terrains rather than bypassing them, and consequently the robot can achieve better performance in terms of traversal time and energy consumption. In this work, four different optimization algorithms, including Adaptive Genetic Algorithm, Elitist Ant System, Dijkstra and Dynamic Weighting A*, along with a well-known reinforcement learning algorithm called Q-Learning, are proposed to solve this path planning problem. The outputs of these algorithms are the optimal path through the environment and the associated configuration on each segment of the path. The challenges involved in mapping the path planning problem to each algorithm are discussed in full details. Eventually, all algorithms are compared in terms of the quality of their solutions and convergence rate.

Keywords Self-reconfigurable modular robots (SRMRs) · Robot path planning (RPP) · Adaptive genetic algorithm (AGA) · Elitist ant system (EAS) · Dijkstra · dynamic weighting A* (DWA*) · Q-Learning

✉ Sajad Haghzad Klidbary
Sajjad.haghzad@gmail.com

[1] Sharif University of Technology, Tehran, Islamic Republic of Iran

1 Introduction

Self-reconfigurable modular robot (SRMR) system is made up of independent and simple modules which can rearrange and reassemble with other modules to form a variety of configurations [1–3]. The idea of these robots was first proposed in the late 1980s, and since then a considerable amount of research has been carried out in this field. Having the capability to adapt their shape and functionality to their appointed task or the environment, the SRMRs major advantage is their great flexibility and robustness. The objective of employing such robots is to build multi-purpose robots whose functionalities are not confined to one or a few applications. In robotic applications, including space or underwater explorations [4,5] where versatility and robustness are required and the total load to be carried is a critical factor, or in rescue missions [6] where the appointed task or the environment model may not be fully known, SRMRs outperform fixed-shape robots [1,7–9]; however, if the environment and the appointed task are known in advance, it is more efficient to build fixed-shape and specific-purpose robots. Further motivations for using SRMRs include economic and self-repair advantages.

Reconfiguration is a process in which the robot changes its current configuration to another configuration based on a predefined motion plan [8,9]. Generally, SRMRs can be viewed from two perspectives [10,11]. From the first point of view, the individual motion capability of each module is considered and the second point of view highlights the capability of group motion as a result of interconnection between modules. Having considered these viewpoints, the architecture of SRMRs can be categorized into three different types called: Lattice-type, Chain-type and Mobile-type [12]. Lattice-type architecture uses cluster-flow to move and reconfigure [11]. Crystalline [13] and ATRON [14] are examples of this type.

Chain-type architecture forms chain structures and has joints that help them move without necessarily performing reconfiguration. M-TRAN [15] and CONRO [16] are examples of this type [11]. Unlike two other types, in Mobile-type architecture, modules can move individually, also can this kind of modules connect to each other and form a variety of configurations. iMbot [17] and ACMoD [18] are examples of this type. In this paper, we narrow down our focus to Mobile-type, ACMoD robotic system in particular.

Robot path planning is defined as a problem of finding a proper collision-free path for one or more robots from a start point to a goal point with regard to different evaluation criteria [7,19]. The number of feasible paths for a mobile robot to go from a start point to the goal point is often very large. Therefore, the path planning problem is one of the most challenging tasks in mobile robotics [7,8,20,21]. This problem can be converted to a constrained optimization problem in which finding an optimal path involves searching the space of possible solutions [7].

Based on the extent to which the environment is observable by the robot, the path planning problem can be categorized into global and local modes [22]. In the global mode, the model of the whole environment is known, whereas in the local mode, the environment is locally observable by the robot. From the environmental characteristics point of view, the path planning problem can also be categorized into static and dynamic modes [22]. In the static mode, the characteristic of the environment is fixed, but in the dynamic mode, the environment is changing. In this work, global static mode has been examined.

Environment description and an appropriate search algorithm are the main prerequisites of any path planning problem. Over the last decade, different methods have been proposed for environment description which can be categorized into two general types called environment decomposition and graph description. In the former method, the whole environment is decomposed into smaller grids called cells. The adjacency of these cells is represented by an undirected graph called *Adjacency Graph*. In the graph description method, the environment is reduced to a network of 1-D curves and as a result, the path planning problem is reduced to searching a path between the start point and the goal point on this network. Grid decomposition [23], MAKLINK graph [24], Voronoi diagram [25], visibility graph [26] and also the combination of these methods are some examples of environment description methods.

Generally, there are two types of optimization algorithms known as deterministic and heuristic. In deterministic algorithms, if there exists an optimal solution, the algorithm can find it by sequentially searching the whole search space. On the other hand, heuristic algorithms use probabilistic search methods to find the optimal solution in a proper time (optimal or near optimal) while there is always a trade-off between speed and accuracy. Different algorithms have been employed for path planning, including Genetic Algorithm (GA) [7–9,18,20], neural networks [27], particle swarm optimization [2], artificial potential field method [28,29], Ant Colony Optimization (ACO) [30–32] and A* algorithm [8,33,34] which are all heuristic algorithms. Dijkstra's algorithm has also been reported for path planning [22,24,35] which is a deterministic algorithm. Q-Learning, as its name conveys, is a learning algorithm that has also been employed in path planning in which the robot finds the optimal path according to the reward that it receives through interaction with the environment [36–39].

To the best of our knowledge, almost all related works in path planning have been done with fixed-structure robots; however, having considered the growing popularity of modular robots, path planning of SRMRs in multi-terrain environments is of particular importance. In this work, we will show that the path planning of SRMRs can be simplified and solved by well-known path planning algorithms which are so far used for fixed-shape robots. However, for this purpose, the path planning problem of SRMRs need to be defined in new form to be mapped to either of these algorithms. In comparison with our previous work [18], the convergence rate and the optimal fitness value of the Genetic Algorithm have been improved by employing the adaptive version of the Genetic Algorithm (AGA), and the ACMoD robotic system is now under realization. Furthermore, to draw a more comprehensive comparison, more algorithms called Elitist Ant System (EAS), Dijkstra, Dynamic Weighting A* (DWA*) and Q-Learning algorithm have also been considered.

The rest of the paper is organized as follows: Our novel SRMR system is introduced in Sect. 2. Basic concepts in path planning are reviewed in Sect. 3. The mapping of the robot path planning problem to Adaptive Genetic Algorithm, Dijkstra, Dynamic Weighting A*, Elitist Ant System and Q-Learning is presented in Sects. 4, 5, 6, 7 and 8, respectively. The simulation results of the proposed algorithms are analyzed in Sect. 9 before Sect. 10 concludes our paper and provides suggestions for future research.

2 ACMoD robot

In this section, our novel SRMR system called ACMoD is introduced. The rest of the section briefly reviews the mechanical design as well as individual motion, group motion and reconfiguration abilities.

2.1 Mechanical design

An ACMoD robotic system is a set of independently controlled mechatronic modules. In addition to capability of individual motion on its wheels, each ACMoD module is

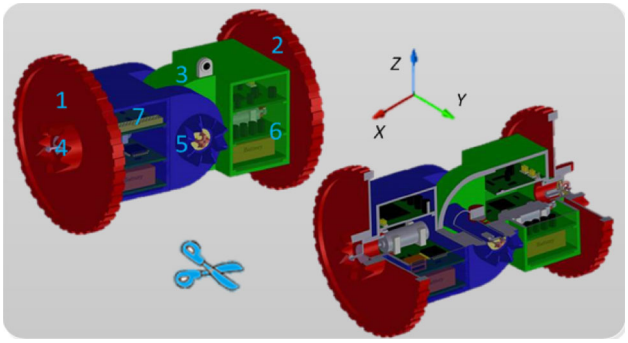

Fig. 1 An ACMoD module and its cross-sectional view. All modules are identical. Each module can perform some computations and can communicate with its immediate neighboring modules. SolidWorks software is used for its mechanical design

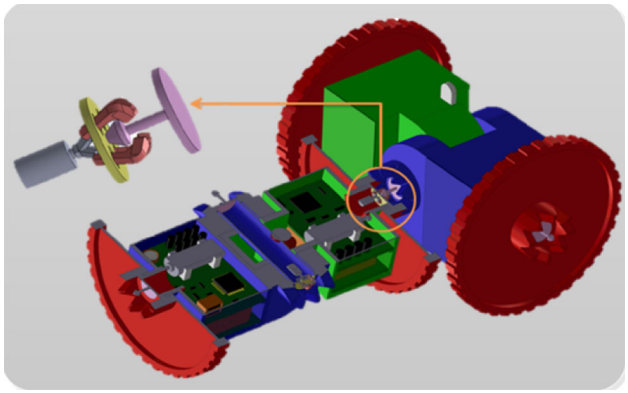

Fig. 2 The docking mechanism of ACMoD module. Clasp system in the docking mechanism provides rigid connection

also able to connect to other modules to form complex configurations [18]. ACMoD can automatically reconfigure and adapt to the dynamic environment.

The major contribution of this design includes but not limited to (1) rigid and flexible structure, (2) mobility of each module, (3) ability to reconfigure and form several 3-D configurations, (4) group motion. ACMoD modules are designed so that its center of mass is adjusted close to geometric center of the robot; thus, individual motion is acceptably stable.

Figure 1 shows an ACMoD module and its cross-sectional view. Each module is symmetric and has three degrees of freedom. As it can be seen, it is comprised of different parts. Part (1) and (2) are two lateral wheels that enable the module to move individually. Part (3) acts like a joint and add another degree of freedom in order to make the module able to bend around that point. This part plays a critical role in some configurations like legged configurations. Each module has four connecting joints overall (two on the wheels and two on the central box) which let the modules to connect to each other from different sides. One connecting joint on the wheel and one connecting joint on the central box are female joints (Active Connectors), and the two other connecting joints are male joints (Passive Connectors) as shown in Fig. 2. Part (6) and (7) are the containers of electronic and mechanical components, including servo motors, battery and electronic boards.

2.2 Docking mechanism and feasibility

The docking mechanism of modular robots is a critical feature as it plays an important role in forming complex configurations [40]. Each module can connect to at most four other modules. The docking mechanism is shown in Fig. 2. Because this mechanism should resist high forces in some configurations, it should provide a rigid and reliable connection in order to prevent unwanted separation and wobble. Therefore, in this module we used clasp system instead of electromagnetic connection.

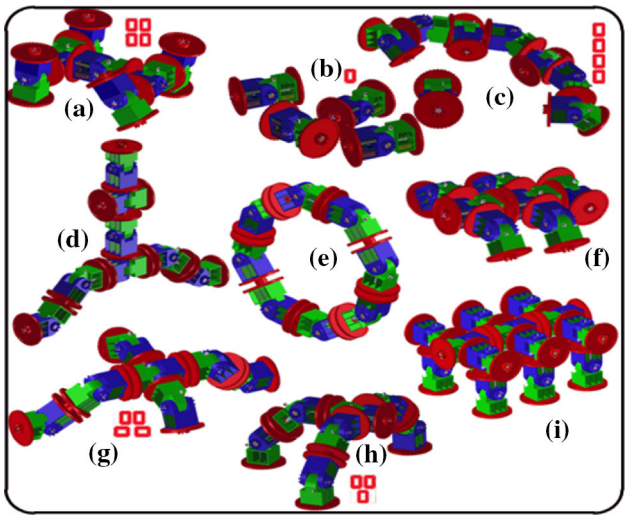

Fig. 3 Some feasible configurations of ACMoD robots and their symbolic representation. (*a*) Shows a 4-legged robot, (*b*) shows a few individual modules, (*c*) is a snake configuration, (*d*) is a Segway, (*e*) shows a ring, (*f*) is another type of 4-legged robot, (*g*) is a another type of Segway, (*h*) is a 3-legged robot, and (*i*) is a 6-legged robot. Only configurations shown in (*a*), (*b*), (*c*), (*g*) and (*h*) are used in the experiments

To form complex configurations for group motion, ACMoD modules can find other modules and then attach to each other like swarm robotic system. Each module receives the information and independently determines its motion. In this SRMR system, the navigation algorithm to help modules find each other is ERRT algorithm that was introduced in [41,42]. Various feasible configurations are shown in Fig. 3. Each configuration has its own Central Pattern Generator (CPG) that performs locomotion. These configurations can be automatically reassembled to form other configurations. Figure 4 shows some examples of different configurations on various terrains. For instance, ACMoD robots can form snake configuration to pass through narrow tunnels, as shown in Fig. 4c. Legged configurations are suitable for cobbled or hilly terrains as shown in Fig. 4b, d.

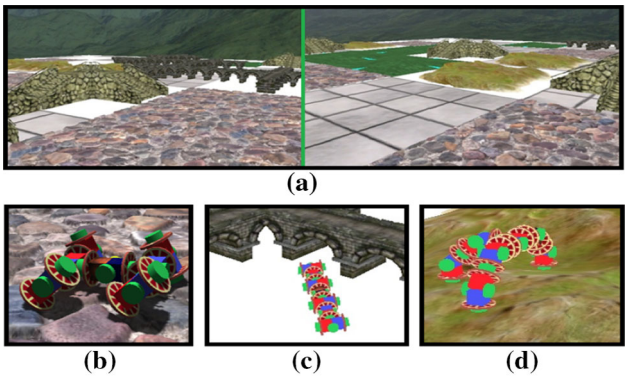

Fig. 4 **a** The side view of the simulation environment covered with various terrains. **b** 4-legged robot on the cobbled terrain. **c** Snake configuration passing through a narrow tunnel. **d** 3-legged robot climbing a hilly terrain. Modules and the environment are simulated in Microsoft DirectX software using NVidia PhysX library

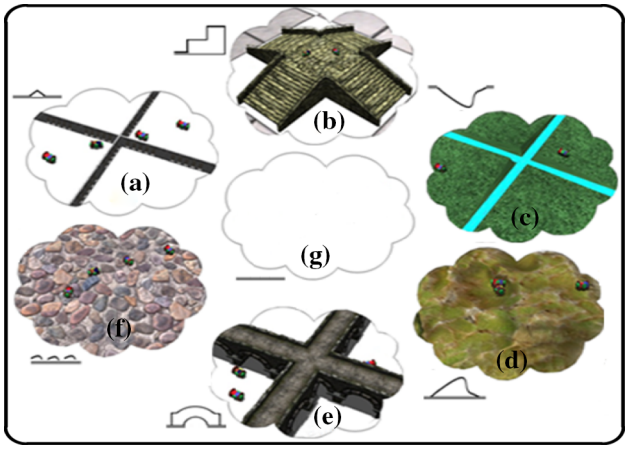

Fig. 5 Seven types of traversable terrains and their symbolic representations on their *left*. (*a*) Shows a fenced terrain, (*b*) shows a stairway, (*c*) shows a terrain with gaps, (*d*) shows a hilly terrain, (*e*) shows a bridge, (*f*) shows a cobbled terrain, and (*g*) shows a flat terrain

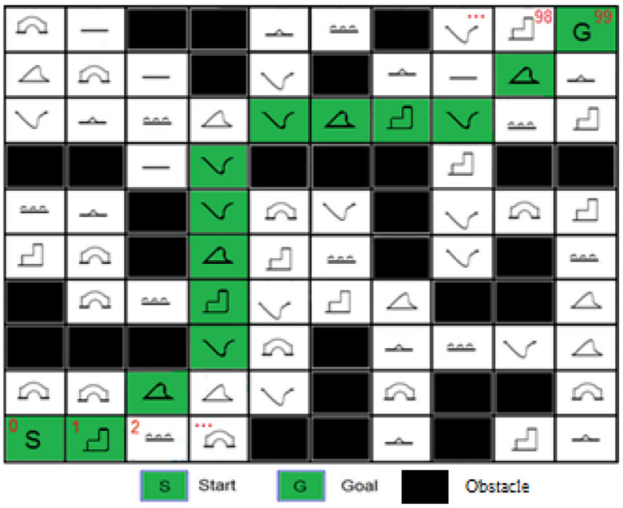

Fig. 6 Matrix representation of the environment. *Black grids* indicate the presence of an untraversable obstacle in that grid. *Other grids* are covered with one of the seven types of terrains. The *green path* is a typical valid path from the start grid to the goal grid that algorithms can find

After examining the internal structure, the potential capabilities of each module are checked. The most important constraint in designing the mechanical part of ACMoD is to choose the appropriate motors in terms of torque and power. Over design in motors leads to large and heavy modules. In ACMoD, five servomotors with different characteristics are needed, two of which are used to control wheels. The third one is for central joint. Last two motors are used for docking mechanism (female joints). In configurations, including 4-legged robots, the central joint motors play an import role because they need to be powerful enough to move legs for locomotion; thus, for this motor torque is much more important than speed.

3 Basic concepts

In this section, some basic concepts concerning environment representation and optimization criteria are reviewed before the main problem can be defined. Regardless of any algorithm chosen for path planning the SRMRs, there are some common steps involved. First, the static environment should be determined and mapped using a proper global map. The next step is to choose the start and goal states. Then one of the proposed algorithms is employed to optimize its objective function which contains terms of energy and time consumption for both traversal and reconfiguration. The output is then the optimal path through the environment and associated configuration on each path segment.

3.1 Environment determination

The first step of the path planning problem is environment determination, meaning that the layout of the environment in terms of the location of obstacles, start and goal points, and different terrains should be defined. The environment in our experiment is divided into squared grids, and each of which is covered by either an untraversable obstacle or one of the seven types of terrains as shown in Fig. 5. Matrix representation is another form of representing the environment in which a grid is colored black if there is an untraversable obstacle in that grid; otherwise, the grid is covered with one of the seven types of terrains as shown in Fig. 6. All grids are numbered in order. This way of numbering performs better than Cartesian method because of its less computational complexity and memory management benefits [7,20]. It is assumed that the robot can move in all eight directions (north, northeast, east, etc.).

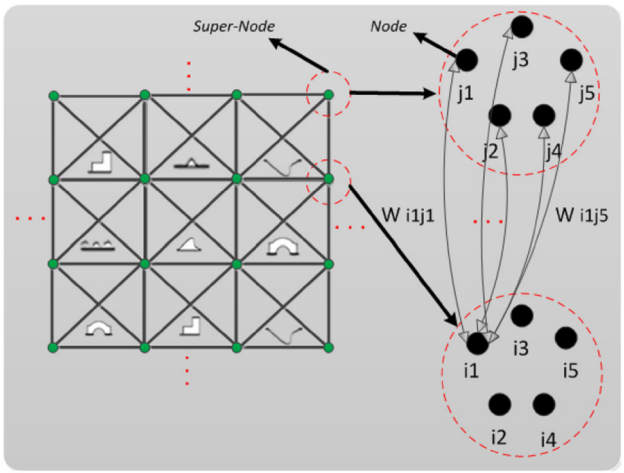

Fig. 7 Graph representation of an environment. For each super-node of the graph, five nodes are assumed which represent five different configurations available for the robot

Some of the algorithms proposed in this work including Dijkstra, DWA* and EAS employ graph representation of the environment as shown in Fig. 7. In this form of representation, we are going to use the term "super-node" to refer to the grids of the environment and we also assume that each super-node contains five nodes representing all configurations available for the robot. For instance, if there is a traversable path from ith grid to the jth grid of the environment, there would be a set of edges between ith and jth super-nodes. For instance, an edge between the first node of ith super-node to the fourth node of the jth super-node means that the robot traverses the ith grid with the first configuration and upon reaching the jth grid, it reconfigures to the fourth configuration. The weight of each edge shows the cost of passing that edge. All edges are two ways which means the robot can return its path.

3.2 Objective function

As it was mentioned before, the contribution of this work is that the reconfiguration ability of ACMoD is considered in the path planning problem. This consideration not only cause some changes in each algorithm, but also emerges in the objective function as energy and time consumption criteria. Therefore, the overall objective function would be a weighted sum of multi criteria which may have the following form:

$$\begin{aligned}\text{Objective function} = &(\theta \times E_T + \Omega \times T_T) \\ &+ (\theta \times E_C + \Omega \times T_C) \end{aligned} \quad (1)$$

where E_T and T_T terms are, respectively, the energy and time costs imposed by traversing the environment. E_C and T_C terms are, respectively, the energy and time costs imposed by reconfiguration. θ and Ω are two coefficients controlling the relative importance of energy terms to time terms in the overall objective function. These two coefficients are chosen by the user based on the priority that the user puts on time optimization over energy optimization or vice versa. It is assumed that the energy and time terms used in the objective function are known for all configurations and all types of terrains. This information is obtained by simulations. Table 1 shows the energy and time consumption of the robot with each configuration traversing each terrain. Table 2 shows the energy and time consumption of the robot for reconfiguration between any two configurations.

4 Adaptive genetic algorithm

In this section, mapping our path planning problem to Genetic Algorithm will be discussed in more details. GA is inspired by the principle of survival of the fittest in the nature, and it is one of the robust and powerful optimization algorithms for complex problems. The algorithm starts with an initial set of guesses about the solution coded in a population of chromosomes, and then the algorithm tries to improve that initial population by applying specific genetic operators, including crossover, mutation and elitism. In this algorithm, an objective function is used to evaluate the performance of

Table 1 Energy (KJ) and time (s) consumption of the robot with each configuration traversing each terrain

Terrain	Configuration									
	4-legged		3-legged		Segway		Individual		Snake	
	E_T	T_T	E_T	T_T	E_T	T_T	E_T	T_T	E_T	T_T
Flat	2.02	22	7.00	25	5.53	16	1.35	22	2.29	21
Fence	3.40	32	4.87	38	∞	∞	∞	∞	4.52	33
Bridge	∞	∞	∞	∞	∞	∞	1.47	22	2.29	21
Cobble	3.20	30	5.13	45	8.43	50	∞	∞	4.43	42
Stairway	5.00	36	6.57	42	∞	∞	∞	∞	∞	∞
Gap	3.45	31	4.20	38	∞	∞	∞	∞	4.65	33
Hill	5.40	35	8.90	34	∞	∞	∞	∞	5.50	45

These numbers are for vertical or horizontal movement of the robot in each grid

Table 2 Energy (KJ) and Time (s) consumption of the robot for reconfiguration between any two configurations

Current Configuration	Goal Configuration									
	4-legged		3-legged		Segway		Individual		Snake	
	E_C	T_C	E_C	T_C	E_C	T_C	E_C	T_C	E_C	T_C
4-legged	0	0	5.69	147	5.19	112	0.19	2	6.19	172
3-legged	5.22	115	0	0	5.22	115	0.22	5	6.22	175
Segway	5.25	117	5.72	152	0	0	0.25	7	6.25	177
Individual	5	110	5.5	145	5	110	0	0	6	170
Snake	5.1	112	5.6	147	5.1	112	0.1	2	0	0

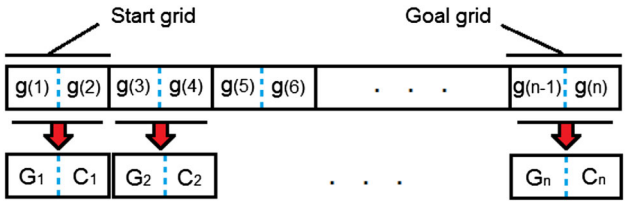

Fig. 8 Chromosome definition in our problem. Odd and even genes are, respectively, the grid location G and associated configuration C of the robot

each chromosome according to specific optimization criteria. In each generation, one of the selection methods, including roulette wheel selection, tournament selection, or rank selection is used to produce the next generation. The evaluation and production of the next generation is performed iteratively as long as the algorithm converges, and the optimal solution is achieved.

4.1 Chromosome definition

The first step in GA is Chromosome definition, meaning that the solution of the optimization problem should be encoded into a sequence of genes. In our optimization problem, the reconfiguration ability of robots should be taken into account, thus not only the path segment, but also the associated configuration on each terrain should be encoded in genes, whereas in previous works like [7,8], chromosomes only contain a set of segments of the path. The proposed chromosome has variable length to model arbitrary path length traversed by the robot in each solution. Each pair of genes $\{g(k), g(k+1)\}, k = 1, 3, 5, \ldots, \mathrm{n}-1$, is the grid location -$G$ and associated configuration C of the robot. $g(1)$ is always the start grid, and grid location genes can be any integer number from zero to the largest grid number in the matrix representation of the environment. n is the length of the chromosome. In other words, $(n+1)/2$ is the length of the path. The convergence speed of GA is dependent on the chromosome definition. An example of a chromosome is shown in Fig. 8.

4.2 Initial population

Choosing the initial population of chromosomes in GA is the other important step that affects the convergence speed of the algorithm. The initial population contains a number of possible solutions for the problem under study. If a large population size is chosen, it is more likely to find the global optima, but computational time becomes a restricting factor. There are two methods available to generate the initial population: random and heuristic [20]. In most cases, heuristic initialization results in faster convergence. In this work, we used heuristic approach and a penalty term to find feasible and infeasible paths. The advantage of this approach will be discussed in the next steps where a few GA operators will be used to decrease the computation time. This method of initialization shows notable improvement compared to the random initialization used in [18]. In this method of initialization, we first assign a very large value to border and obstacle grids and then assign a small initial value (σ) to other grids as shown in Fig. 9. The next step is to calculate the Euclidean distance between each free grid and the center of all obstacle grids (d_{ij}) of the environment. The inverse of these distances is summed up and is then added to σ to obtain a gain for that grid as follows:

$$g_i = \sigma + \frac{1}{d_{i1}} + \frac{1}{d_{i2}} + \cdots = \sigma + \sum_{j=1}^{L} \left(\frac{1}{d_{ij}} \right), \tag{2}$$

where L is the number of obstacle grids in the environment, and d_{ij} is the Euclidean distance of the ith grid from jth grid.

For each grid i, a vector denoted by P can be computed that has eight elements filled with the probability of choosing each surrounding grids by the robot if it is located in the ith grid. The probability of choosing jth grid if the robot is in the ith grid is the ratio of g_j to the sum of the gains of all eight grids surrounding the ith grid. Roulette wheel method is used to choose the next grid for a path generation from the start grid to the goal grid. After generating a path, we should optimize the path and remove loops and unnecessary grids by using specific operators designed for this purpose.

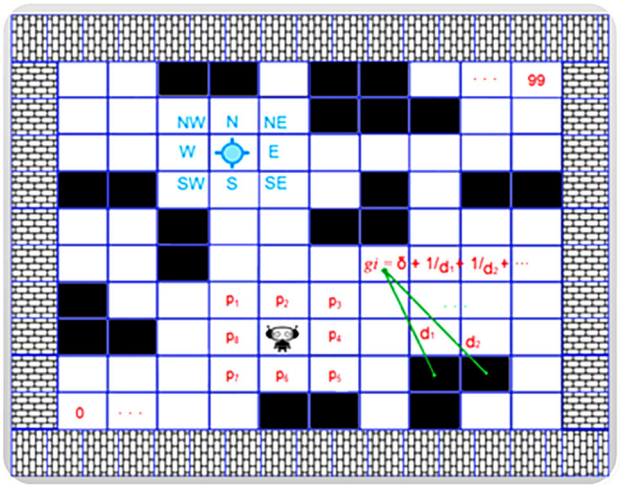

Fig. 9 The proposed heuristic path generation method for initializing the population in GA

4.3 GA operators

Optimization in GA is performed by iteratively applying some operators on each generation population. In this work, these operators are selection, crossover, mutation and two customized operators, shortcut and loop removal. These operators are explained in this subsection.

The primary objective of the selection operator is to ensure that the best chromosomes (solutions) will survive to reproduce the next generation [7]. In this operator, chromosomes are selected based on their fitness values in order to undergo the operations, including mutation, crossover. In [18], almost the whole generation was replaced by children, but in this paper, we use elitism policy to guarantee that the quality of solution will not decrease from one generation to the next generation and eventually improve the convergence rate. For this purpose, 25% of the best chromosomes (lowest fitness value) are carried over to the next generation and then the operators are applied to the whole generation to reproduce the rest of the chromosomes for the next generation.

Crossover is one of the fundamental operators of genetic algorithm that combines two parents in order to exchange their information and generate new solution that would benefit from both parents. There are various versions of crossover, namely 1-point, 2-point and uniform, etc. We use 2-point crossover in this work, in which we randomly choose two parents and search for two distinct common genes in them and then all genes between those two common genes are swapped between parents to generate two children. Crossover operator is only applicable on odd genes (grid genes) so that the resulting path remains continuous.

Mutation is an operator that increases the diversity in the population by making small alterations in a chromosome. This operator prevents immediate convergence to a local minimum. In this work, we use two different mutation operators, grid mutation and configuration mutation. After applying either of mutation operators, only those results are acceptable that the obtained pair of genes is valid, meaning that the suggested grid is traversable by the suggested configuration.

The objective of shortcut removal operator is to reduce the total length of the path. This operator searches the whole chromosome to find redundant grids and then remove them. Likewise, loop removal operator searches for loops in the path and removes them. Smoothing operator is used to smooth the paths that have considerable deviation. It should be mentioned that increasing the number of operators, slows down the convergence.

4.4 Evaluation and fitness function

Defining a proper fitness function is always a challenging step, and it is often defined according to the nature of the problem to be optimized. Appropriate selection of the fitness function will lead the algorithm toward the optimal solution [20,27]. All chromosomes are evaluated based on their fitness value. The fitness value, in fact, determines those chromosomes that will be directly transferred to the next generation and those chromosomes that will not survive or undergo operations. Our proposed fitness function is multiple criteria. Time and energy consumption terms imposed by traversal as well as time and energy consumption terms imposed by reconfiguration all take part in our proposed fitness function as follows:

$$f = \lambda \times \sum_{i=1}^{K} (\theta \times E_T(t_i, c_i) + \Omega \times T_T(t_i, c_i)) + \sum_{i=1}^{K-1} (\theta \times E_c(c_i, c_{i+1}) + \Omega \times T_c(c_i, c_{i+1}) \quad (3)$$

$$\theta + \Omega = 1 \quad (4)$$

where K denotes the path length. $E_T(t_i, c_i)$ and $T_T(t_i, c_i)$ are, respectively, the energy and time required by the robot with configuration c_i to traverse the terrain t_i. $E_c(c_i, c_{i+1})$ and $T_c(c_i, c_{i+1})$ are, respectively, energy and time required by the robot to reconfigure from configuration c_i to configuration c_{i+1}. θ and Ω are constant weights defining the relative importance of time and energy terms in the total fitness value. λ is a coefficient to distinguish between vertical/horizontal movement and diagonal movement on each grid. Hence, it is equal to 1 and $\sqrt{2}$, respectively, for the vertical/horizontal movement and diagonal movement.

In order to make the original version of GA adaptive, the probabilities of crossover, p_c, and mutation, p_c, should be adjusted adaptively [43,44]. The adaptive version of GA is called Adaptive GA (AGA). This adaptation is done by uti-

lizing the information of population in each generation as follows:

$$p_c = \begin{cases} \frac{k_1 \times \left(f' - f_{min}\right)}{(f_{avg} - f_{\min})} & f' \leq f_{avg} \\ k_2 & f' > f_{avg} \end{cases} \quad (5)$$

$$p_m = \begin{cases} \frac{K_3 \times (f - f_{min})}{(f_{avg} - f_{min})} & f \leq f_{avg} \\ k_4 & f > f_{avg} \end{cases} \quad (6)$$

where $f_{\min}$ and f_{avg} are, respectively, the best and the average of fitness value in the population. f' is the better fitness value between parents involved in crossover operation, and f is the fitness value of the solution to be undergone mutation. If the value of K_1, K_2, K_3 and K_4 are chosen in the interval [0, 1], the two probabilities are updated adaptively. In this work, p_m is used for both grid mutation and configuration mutation. Convergence criterion in this algorithm is as follows:

$$\text{Error}\,(Z) = \frac{f_Z - f_{Z-1}}{f_Z} < 0.001 \quad (7)$$

where f_Z and f_{Z-1} are the best fitness values for the Zth and $(Z-1)$th generations, respectively.

5 Dijkstra's algorithm

Dijkstra's algorithm is an optimization algorithm and also a graph search algorithm that uses deterministic search to find the shortest path from a starting node to every other node in a graph with nonnegative edge path costs [8,45]. If there is an optimal path, this algorithm can find it with high probability by searching the whole space. Dijkstra's algorithm begins from the start node and assigns a cost value to all its neighboring nodes, then visits the node with lowest cost and does the same calculation for the new node. During this process, if it finds a lower cost for any node that it has already assigned a cost value, it updates the cost of that node and goes on until it visits the goal node.

To map our problem to this algorithm, the graph model presented before in Sect. 3 has been employed. The first step in Dijkstra's algorithm is to define the adjacency matrix $G\,(N, E)$ whose elements are defined as follows:

$$g_{ij} = \begin{cases} w_{ij}, & if\ edge\left(n_i, n_j\right) \in E \\ \infty, & if\ O.W. \end{cases} \quad (8)$$

where N is the set of nodes and E is the set of edges. If there is an edge between node i and node j, then the associated element of the adjacency matrix is equal to the cost of that edge. The cost of traversing from node i to node j is obtained as follows:

$$w_{ij} = \lambda \times (\theta \times E_T\,(t_i, c_i) + \Omega \times T_T\,(t_i, c_i)) + \left(\theta \times E_c\left(c_i, c_j\right) + \Omega \times T_c\left(c_i, c_j\right)\right), \quad (9)$$

where the definition of $E_T\,(t_i, c_i)$, $T_T\,(t_i, c_i)$, $E_C\left(c_i, c_j\right)$, $T_T\left(c_i, c_j\right)$, θ, Ω and λ are the same as before. The path evaluation function C is then the sum of all partial costs imposed by each edge of the graph from start node to the final node. Despite being very time-consuming, Dijkstra's algorithm has been included in our comparison as a base line to compare the performance and execution time of other algorithms.

6 DWA* algorithm

A* is a popular search algorithm that enjoys widespread use in path planning and graph traversal because of its speed and performance [33,34]. A* Algorithm is a combination of Dijkstra's algorithm and a greedy search algorithm called Best-First-Search. Despite Dijkstra's algorithm that uniformly searches in all directions without considering the location of the goal node, A* algorithm uses a predefined heuristic function which proposes an estimation for the shortest path to the goal node. Therefore, compared to Dijkstra's algorithm, A* enjoys faster search and less memory requirements. Following the path with lowest estimated cost during traversal, A* algorithm also keeps a sorted priority queue of alternative path segments along the way toward the goal node. If, at any point, a segment of the path being traversed has a higher cost than another encountered path segment, the algorithm abandons the higher-cost path segment and traverses the lower-cost path segment instead. This process continues until the goal has been reached.

The A* algorithm uses a problem-specific heuristic function denoted by $h\,(n)$, which in our problem is defined to be the straight-line distance to the goal point in order to estimate the cost of the path from the current node to the goal node. The algorithm also evaluates the path cost $g\,(n)$ which is the cost incurred from the start node until the current node. In fact, $g\,(n)$ represents the performance of Dijkstra's algorithm and $h\,(n)$ represents the performance of Best-First-Search algorithm. One of the drawbacks of A* algorithm is that the importance of heuristic function and the cost of path already traversed equally contribute in the total cost through the search. To solve this problem, the Dynamic Weighting A* has been proposed in which the total cost function is defined as follows:

$$f\,(n) = g\,(n) + w\,(n) \times h\,(n), \quad (10)$$

where $w\,(n)$ is a weight that dynamically changes the relative importance of the heuristic function in the total cost function. In the traditional A* algorithm, this weight is constant and equal to one, but in DWA*, this weight is first chosen to be $w\,(n) \geq 1$, so that in the early steps of search, higher priority is put on finding the right direction toward the goal node more

quickly, and as we get closer to the goal node, the weight is gradually decreased in order to put higher priority on finding the optimal path [46,47].

In comparison with Dijkstra's algorithm, since A* algorithm use a heuristic function and an ordered priority queue, it cuts down on the set of nodes that must be investigated. DWA* further cuts down on such nodes by heavily weighting heuristic function in the early steps of the search, but it tries to less compromise the optimality as it get closer to the goal. However, since the heuristic function is greedy, there is no guarantee that it suggests the actual shortest path [8]. Therefore, execution time and quality of solution can be significantly influenced by the definition and the preciseness of the heuristic function

7 EAS algorithm

The Ant Colony Optimization (ACO) is a probabilistic optimization algorithm that can be used for local and global path planning. This algorithm is inspired by the behavior of biological ants when foraging for food to find the shortest paths between their colony and different sources of food [30,31,48]. When an ant finds a source of food, on its way back to the colony, it deposits pheromone trail on the ground. Other ants choose the pheromone trail that is more intense compared to other pheromone trails. However, over time, pheromones evaporate and decrease the attractiveness of paths, but shorter paths are traversed more frequently and that keeps their intensity and attractiveness at high level. Pheromone evaporation, in fact, prevents convergence to a local optima. By this natural process, ants are able to solve an optimization problem of finding the shortest path to the source of food [49].

7.1 Edge selection

In ACO, like the two other graph-based algorithms in this work, the same graph model presented before in Sect. 3 has been employed. To generate a new solution for the problem, each ant moves node to node. In the process of choosing the next node, each ant makes a probabilistic decision. For *kth* ant, the probability of moving from node i to node j is defined as follows:

$$P_{ij}^{k} = \begin{cases} \frac{\left(\tau_{ij}^{k}\right)^{\alpha} \times \left(\eta_{ij}^{k}\right)^{\beta}}{\sum_{l \in N_i^k} \left(\tau_{il}^{k}\right)^{\alpha} \times \left(\eta_{il}^{k}\right)^{\beta}} & if\ j \in N_i^k \\ 0 & if\ j \notin N_i^k \end{cases} \tag{11}$$

where τ_{ij}^{k} is the pheromone level deposited in the transition from node i to j, and η_{ij}^{k} is heuristic information that represents the attractiveness of moving from node i to j. In this work, $\eta_{ij}^{k} = 1/w_{ij}$ has been assumed, where w_{ij} is the cost of edge between node i and j and is defined the same as (9). α and β are parameters to control the influence of τ_{ij}^{k} and η_{ij}^{k}, respectively. When $\beta = 0$, then $\left(\eta_{ij}^{k}\right)^{\beta} = 1$ and the probability of choosing a path only depends on the pheromone level and if $\alpha = 0$, then this probability only depends on the attractiveness of paths. The summation in the denominator ensures sum-to-one rule and considers all possible paths in the paths set N_i^k, if the kth ant is in node i.

7.2 Pheromone update and evaporation

As mentioned before, the amount of pheromone on different paths can influence other ants in their decision of choosing among paths, and hence, shorter paths are traversed more frequently and thus keep their pheromone at high level. In this work, we used ACO with Elitism policy, Elitist Ant System (EAS), in which the deposit of pheromone and evaporation effect are modeled as follows: [50]:

$$\tau_{ij}(t+1) = (1-\rho) \times \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e \times \tau_{ij}^{e}(t), \quad \forall (i,j) \in L, 0 < \rho \leq 1 \tag{12}$$

where τ_{ij} is the amount of pheromone deposited in the transition from node i to node j, m is number of ants, and $\Delta\tau_{ij}^{k}$ is the pheromone increment left by kth ant on its transition from node i to node j. In (12), $\Delta\tau_{ij}^{k}$ can be defined as follows:

$$\Delta\tau_{ij}^{k} = \frac{1}{w_{ij}^{k}}, \tag{13}$$

where w_{ij}^{k} is the cost of transition from node i to j for kth ant, as it was defined in (9). Pheromone also undergoes evaporation, and this phenomenon is modeled by an evaporation coefficient ρ which is a number between zero and one.

The formulation of ACO system does not have the last term in the right-hand side of (12); however, in EAS, this term has been added in the pheromone update rule in order to take into account the best solution found until that iteration as well. The effect of the best solution can be even greater than the other solutions. In other words, by this updating rule, the solutions are biased toward the best solution. This modification results in faster convergence, but the algorithm might also be trapped in local optima. In (12), elitism parameter e defines the intensity of the effect of the best solution on the next solution and $\tau_{ij}^{e}(t)$ is nonzero only when edge ij is part of the best path found until that iteration. Appropriate value for e helps the algorithm to find better solution in shorter period of time; however, large value may result in algorithm concentrates early on suboptimal solution, and the

algorithm fails to find the optimal solution. In other words, by this updating rule, one can balance the exploitation and exploration. The same criterion as GA can be used for convergence criterion of EAS.

8 Q-Learning

In reinforcement learning algorithm, through interaction with the environment and in an attempt to maximize the long-term reward, the agent learns the optimal strategy. In other words, reinforcement learning is a mapping from the state space to the action space so that the reward function becomes maximum [51,52]. Q-Learning is a reinforcement learning algorithm that can be used for Markov Decision Process (MDP) in which the model of the environment is not available [53]. In the interaction with the environment, the decision-making agent iteratively takes an action $a_t = a \in A$ and as the result of that action, it receives an immediate reward r_{t+1} and is transmitted from its current state $s_t = s \in S$ to the next state $s_{t+1} = s' \in S$, where A and S are the finite set of actions and states, respectively. The quality of state-action pairs are stored in a matrix called Q-matrix denoted by Q. Before learning has started, this matrix is initialized to zero or to random numbers and during the learning process, it is updated iteratively and finally converges to the optimal Q-matrix [37,54,55].

Policy $\pi(s) \in A$ is the strategy by which the agent chooses the next action in each state. The objective of the agent is to learn the optimal policy $\pi^*(s) \in A$ for any state s so that the total reward in long term becomes maximum. Q-Learning finds the optimal policy by approximating state-action pair values or Q-matrix [21,55]. The optimal policy can then be achieved as follows:

$$\pi^*(s) = arg\max_a \left(Q^*(s,a)\right) \tag{14}$$

Q-Learning algorithm finds the optimal Q-matrix $Q^*(s,a)$ through an iterative and recursive method based on available information. The updating rule in this algorithm is as follows:

$$Q(s_t,a_t) = Q(s_t,a_t) + \propto \times \left[r_{t+1} + \gamma \times \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right] \tag{15}$$

where $0 \leq \gamma \leq 1$ is discount factor that keeps a trade-off between the importance of immediate and long-term reward and $0 \leq \alpha < 1$ is learning rate that defines the importance of recently obtained information compared to old information in updating Q-matrix.

During the learning process, the strategy of choosing the next action among possible actions in each state highly affects the convergence speed of Q-Learning algorithm. In order to increase the convergence speed, various strategies have been proposed. The simplest strategy is greedy strategy in which the agent takes the action associated with the highest quality and thus does not explore other actions. To solve this problem, ε-greedy strategy was proposed in which the agent takes the action randomly so that the action associated with the highest quality can be taken with the portability of $(1-\varepsilon)$ and other actions can also be taken, but with probability ε. This approach controls the balance between exploitation and exploration by changing the ε parameter. The larger the ε, the more exploration the agent carries out.

ε-greedy strategy is very popular. Its main drawback, however, is that it assigns equal probabilities to all actions other than the one with the highest quality. To solve this problem, soft-max selection method has been proposed. In this method, actions are taken based on different weights which are often derived from Gibbs or Boltzmann distributions. The other drawback of ε-greedy strategy is that when the optimal policy is found, in each state, all actions have the same probability of ε to be taken. One of the strategies proposed to solve this problem is Value-Different-Based Exploration (VDBE). This strategy is similar to ε-greedy strategy, except that ε is no longer a universal constant parameter, rather it is learned separately for each state. In this strategy, when the knowledge of this system is limited, at the beginning of learning process for instance, ε is chosen large. Updating rule for ε is as follows:

$$h(s,a,\sigma) = \left| \frac{e^{Q_t(s,a)/\sigma}}{e^{Q_t(s,a)/\sigma} + e^{Q_{t+1}(s,a)/\sigma}} - \frac{e^{Q_{t+1}(s,a)/\sigma}}{e^{Q_t(s,a)/\sigma} + e^{Q_{t+1}(s,a)/\sigma}} \right| \tag{16}$$

$$\varepsilon_{t+1}(s) = \delta \times h(s_t,a_t,\sigma) + (1-\delta) \times \varepsilon_t(s) \tag{17}$$

where σ is a positive constant called inverse sensitivity, and $\delta \in (0,1]$ is a parameter which defines the effect of the taken action on the exploration rate [56,57].

8.1 WorkSpace

In Q-Learning, like previous algorithms, the environment is gridded. In Q-matrix, the number of rows is equal to the overall number of environment grids and the number of columns is 8×5, where 8 is the number of all directions the robot can move, and 5 is the number of all configurations. If an action in a state is impossible, the associated reward would be a negative large value and the action that leads to the goal grid is rewarded by a positive large value and finally for other actions the agent receives following reward in terms of energy and time consumption for reconfiguration and traversal.

$$Reward = \frac{1}{\lambda \times (\theta \times E_T(t_i, c_i) + \Omega \times T_T(t_i, c_i)) + \left(\theta \times E_c\left(c_i, c_j\right) + \Omega \times T_c\left(c_i, c_j\right)\right)} \tag{18}$$

where, the definition of $E_T(t_i, c_i)$, $T_T(t_i, c_i)$, $E_C\left(c_i, c_j\right)$, $T_T\left(c_i, c_j\right)$, θ, Ω and λ are the same as before.

9 Simulation results

In this section, the simulation results of the proposed algorithms are analyzed and compared. All simulations were conducted in MATLAB 2013 on a personal computer with the following specifications: Intel Core i5, 2.4 GHz, 4 GB RAM. Algorithms were run with parameters setting as in Table 3.

To draw a more meaningful comparison among algorithms, two measures of evaluation were used: the execution time of algorithms as well as the quality of their solutions [7,8]. Simulations were conducted on different sized environments with various layouts. In reporting the results of heuristic algorithms, the average and the variance of the solutions for multiple executions have been included to show the quality of the solutions as well as the stability of the algorithms.

9.1 Small environment

In this experiment, all algorithms were tested on a 10×10 environment. All algorithms were compared in terms of the average and the variance of cost value and the average of execution time. Table 4 compares EAS, AGA and GA with different population size. Table 5 compares Dijkstra, DWA* and Q-Learning. The number of execution iterations in EAS, AGA, GA and Q-Learning was 200. The two other algorithms are deterministic, and hence, their result was consistent in multiple iterations.

Table 3 Parameters setting of the proposed algorithms

Algorithm	Parameter	Value
GA & AGA	K_1	1
	K_2	0.6
	K_3	1
	K_4	0.55
DWA*	$h(n)$ (Heuristic function)	A straight-line distance to the goal grid
	$w(n) = \frac{w_0}{1+t}$	$w_0 = 2.5$
EAS	α (Pheromone trail coefficient)	5
	β (Heuristic coefficient)	5
	ρ (Evaporation rate)	0.3
	e (elitism parameter)	0.7
Q-Learning	α (Learning rate)	0.9
	γ (discount factor)	0.5
	δ, σ	0.3
	Q (Q-matrix)	[0]
Objective Function Coefficients	Θ & Ω (constant weights)	0.5

As it was expected, Dijkstra's algorithms achieved the best cost value within reasonable amount of time in the small environment. Next, in terms of best cost value, were DWA*, EAS and then AGA. In comparison with GA, AGA showed noticeable improvement in terms of both execution time and cost value. Q-Learning minimized the cost value but not as much as others. Obviously, in EAS, GA and AGA, the larger the initial population size, the smaller the cost value and its variance and the longer the execution time. However, the fastest algorithms in this environment size were DWA* and then Dijkstra. It should also be mentioned that except Dijkstra and DWA* that are deterministic algorithms, others are heuristic and multiple execution iterations were required to achieve the best result. However, EAS had the least variance and Q-Learning performed better than AGA in terms of variance of solutions. The graphical representation of the solutions of all algorithms is shown in Fig. 10.

Figure 11 compares the convergence rate of EAS, AGA and GA proposed in [18]. As it can be seen, AGA outperformed GA in terms of both convergence rate and fitness value. EAS, however, outperformed both. In addition, EAS convergence rate was smoother and faster than that of GA and AGA. Figure 12 compares the average and the variance of fitness value among the population of consecutive generations in EAS and AGA. As it can be seen, EAS not only achieved better fitness value, but also on its way to converge to that value enjoyed considerably smaller variance because with elitism policy, the best-so-far solution has taken into account and each generation is biased toward that solution.

9.2 Large environment

The same experiment was conducted on large environment. The results of this experiment are provided in Tables 6 and 7. In this experiment, the environment contained 38% obstacle grids.

Table 8 summarizes the results of Tables 4, 5, 6 and 7. In this table, algorithms are compared with respect to the execution speed, the variance of their solutions, susceptibility to being trapped into local optimum and their specific requirements.

Table 4 Comparison of the average and the variance of cost value, and the average of execution time of EAS, AGA and GA with different population size

Environment size 10*10	Number of execution iterations = 200								
	AGA			GA in [18]			EAS		
Population size	C	Var.	T	C	Var.	T	C	Var.	T
20	8.40	4.09	0.92	8.47	5.51	0.89	8.18	1.42	0.36
30	8.29	3.52	1.30	8.40	4.25	1.34	8.11	0.93	0.87
50	8.17	3.01	1.98	8.31	3.79	2.10	7.97	0.45	1.63

The number of execution iterations was 80. The environment was 10 × 10. The unit of time is second and cost is normalized and thus, it is without unit

Table 5 Comparison of the average and the variance of cost value and the average of execution time of A*, Dijkstra and Q-Learning

Environment size	DWA*			Dijkstra			Q-Learning		
	C	Var.	T	C	Var.	T	C	Var.	T
10*10	7.96	0	0.57	7.85	0	0.78	8.69	1.95	1.87

The number of execution iterations in Q-Learning was 200. The environment was 10 × 10. The unit of time of time is second and cost is normalized and thus it is without unit

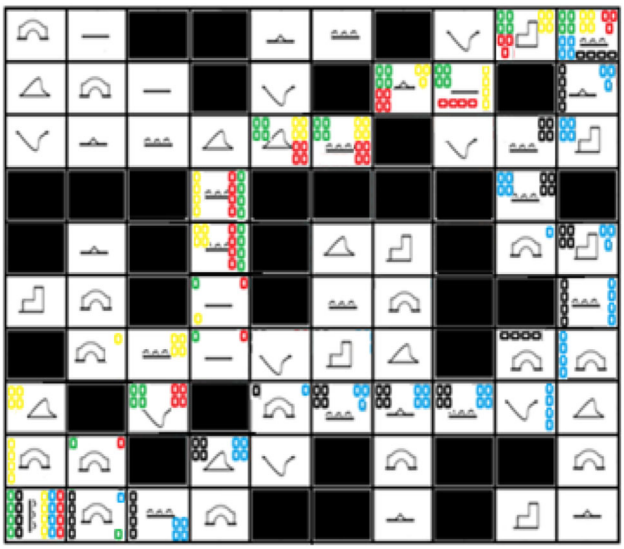

Fig. 10 Graphical representation of solutions offered by proposed algorithms. The environment was 10 × 10 and contained 34% obstacle grids. Start grid was the lower left grid, and the goal grid was the *upper right grid*. *Green* solution is the result of Dijkstra's algorithm. DWA* algorithm solution is shown in *red*. AGA found the *blue* solution. EAS and Q-Learning solutions are distinguished from other solutions by *black* and *yellow* colors, respectively. As it can be seen from the figure, a straight path from start grid to the goal grid was not available and all algorithms successfully overcame this challenge

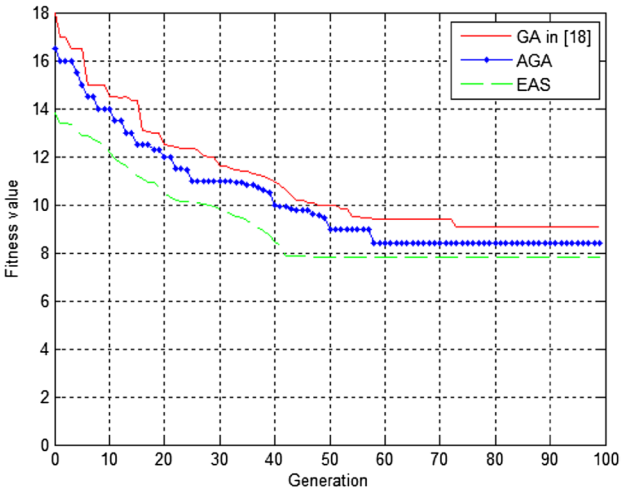

Fig. 11 The comparison of convergence rate among EAS, AGA and GA proposed in [18]. The initial population size for all algorithms was 40, and the results were the average of 100 execution iterations

As it can be concluded from Table 8, choosing among proposed algorithms is application dependent and different criteria may affect this choice, but environment size seems to be the most important criterion. Following are some key points to consider when choosing among proposed algorithms. It is worth mentioning that ACO, A* and Q-Learning with ε-greedy strategy had been simulated, but for the benefit of conciseness and because their improved versions had better performance, these basic versions were not reported in tables. However, in the following discussion all algorithms have been considered.

Dijkstra Because of its greedy search approach, Dijkstra's algorithm finds the optimal solution, but in large environment, it shows poor performance in terms of memory requirement and execution time. In this work, Dijkstra's algorithm has been reported as a base line to compare other algorithms.

DWA* Guiding the greedy search by a heuristic function, A* algorithm performs better than Dijkstra's algorithm in large environments in terms of both memory requirement and convergence speed. One of the main drawback of A* algorithm is its vulnerability to being trapped in local optima. DWA*, because of its dynamic weighting strategy, enjoys

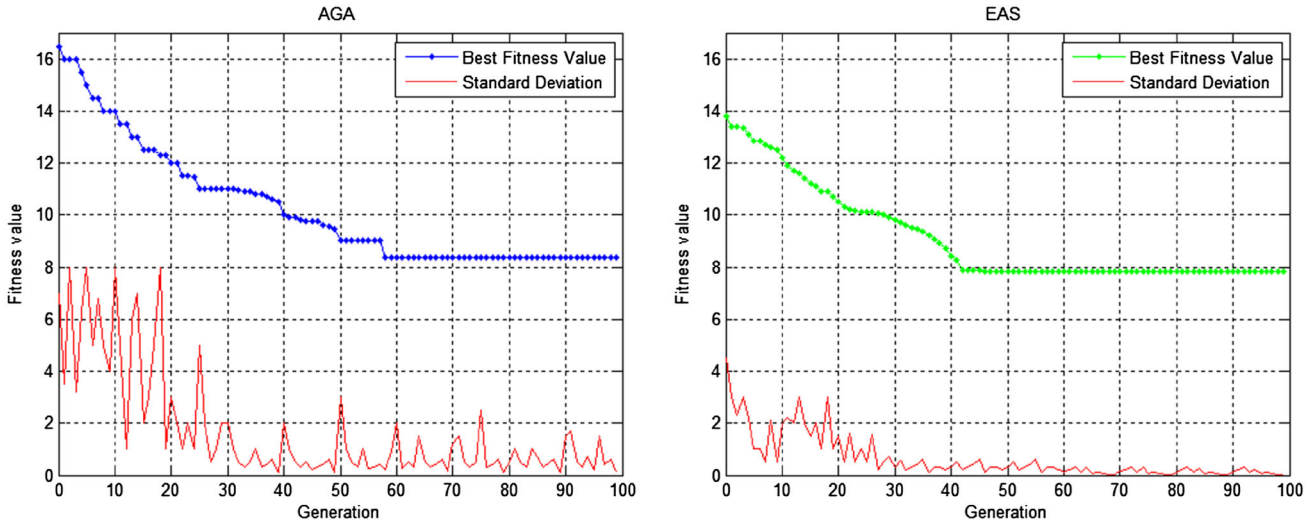

Fig. 12 Comparison of the average and the variance among the population of consecutive generations in EAS and AGA. The initial population size was 40, and the results were the average of 100 execution iterations

Table 6 Comparison of the average and the variance of cost value, and the average of execution time of EAS, GA and AGA on different environment size

Environment size	Number of execution iterations = 200 Initial Population size = 100								
	AGA			GA in [18]			EAS		
	C	Var.	T	C	Var.	T	C	Var.	T
100 * 100	65.24	20.12	70.31	69.02	22.89	73.44	61.33	7.64	49.99
160 * 160	93.44	25.69	100.22	96.91	29.48	103.59	86.23	9.57	52.66
300 * 300	233.98	45.28	248.02	240.11	51.26	249.27	198.24	13.42	170.28

The number of execution iterations was 200. The initial population size was 100. The unit of time is second, and cost is normalized and thus is without unit

Table 7 Comparison of the average and the variance of cost value and the average of execution time of DWA*, Dijkstra and Q-Learning on different environment size

Environment size	DWA*			Dijkstra			Q-Learning		
	C	Var.	T	C	Var.	T	C	Var.	T
100 * 100	62.52	0	51.24	59.04	0	60.95	64.27	12.41	62.13
160 * 160	87.32	0	89.11	85.12	0	120.28	91.29	13.88	106.37
300 * 300	218.27	0	197.71	190.22	0	450.23	227.02	27.09	275.61

The number of execution iterations in Q-Learning was 200. The unit of time is second, and cost is normalized and thus is without unit

Table 8 Comparison of all algorithms with respect to different criteria

Criterion		Algorithm				
		AGA	Dijkstra	DWA	EAS	Q-Learning
Execution speed	Small Env.	Low	High	Very high	Medium	Very low
	Large Env.	Medium	Very low	High	Very high	Medium
Cost optimization	Small Env.	Medium	Very good	Good	Good	Medium
	Large Env.	Medium	Very good	Good	Very good	Medium
Variance of solutions		Large	Zero	Zero	Small	Medium
Memory requirement		Small	Very large	Large	Medium	Medium
Local optima		Susceptible	Not susceptible	Susceptible	Susceptible	Susceptible
Initial population		Required	Not required	Not required	Required	Not required
Environment model		Required	Required	Required	Required	Not required

even faster execution time and less memory requirement compared to A*, but it may perform worse than A* in finding the optimal solution. However, in DWA*, the user can change the weighting rule in order to make a desirable trade-off between speed and optimality.

AGA and EAS AGA and ACO perform well even in large environments. This performance is mainly due to the large and appropriate initial population. This might be considered as an undesirable prerequisite of these algorithms because producing such a population may not always be an easy task. However, as far as these two algorithms are concerned, ACO performs better with random initialization and requires less memory space. Moreover, compared to GA, ACO shows better adaptability in changing environments. EAS converges even faster than ACO, but because of elitism policy, it may experience early convergence to local optima. Hence, properly setting elitism parameter plays an important role in maintaining a balance between speed and optimality. EAS also enjoys smoother convergence rate to local optima compared to ACO and AGA. In comparison with other algorithms, a crucial criterion to notice is the heuristic approach of these algorithms that necessitates multiple runs to achieve the best solution. Therefore choosing, from execution time point of view, between algorithms like Dijkstra's algorithm and EAS or AGA is not necessarily an obvious decision because although Dijkstra's algorithm is time-consuming, it finds the global optimum with only one execution, but algorithms like AGA or EAS which can be executed faster, need to be run multiple times to find the best solution. However, AGA, ACO and EAS algorithms naturally enjoy high level of parallelism and if their codes are developed to suit multi-core processors like GPUs, significantly faster execution time can be achieved.

Q-Learning Q-Learning is model-free and interactively updates its previous experience of the environment. Therefore, without any modification to the algorithm it can also be employed in dynamic situation where either the environment or the robot performance undergo changes through time. Its drawback, however, is that in large environments, the robot requires considerable amount of time interacting with the environment. In addition, making a trade-off between exploration and exploitation is always a challenging decision in this algorithm when ε-greedy strategy is used. However, by employing VDBE strategy, a fair trade-off between exploitation and exploration can be achieved. In addition, compared to ε-greedy strategy, higher reward, more stability, more robustness and faster convergence can been achieved.

10 Conclusion

In this paper, first a novel self-reconfigurable modular robot called ACMoD was introduced. The path planning problem of such a robotic system, with emphasis on its capability to reconfigure automatically, through an environment comprised of various terrains was defined. Then a multi criteria objective function containing energy and time terms for both traversal and reconfiguration was suggested, and five different algorithms including AGA, Dijkstra, DWA*, EAS and Q-Learning were proposed to optimize that objective function. It was shown that the path planning problem of SRMRs can be simplified and mapped to well-known optimization algorithms, though with some customizations in each algorithm. Simulations were conducted to compare the algorithms with respect to the quality of their solutions and their execution time. Simulations revealed that the size of the environment greatly affects algorithms execution time. A choice among different algorithms is not an obvious decision, rather it is application dependent. Some algorithms like EAS and AGA require environment model as well as initial population generation. Only Dijkstra can find the optimal solution, but it is time-consuming in large environments. DWA* has acceptable performance in all environment size. Q-Learning is the only algorithm that does not require environment model, but in large environment, the training phase becomes time-consuming. Eventually, a comprehensive comparison was drawn which helps a user in choosing among different algorithms based on desirable criteria.

Acknowledgements The authors would like to thank Sina Maleki, Salman Faraji and Hossein Davari for their generous help in software design of the robot.

References

1. Bhat P et al (2006) Hierarchical motion planning for self-reconfigurable modular robots. In: 2006 IEEE/RSJ on international conference on intelligent robots and systems
2. Liu T et al (2009) The adaptive path planning research for a shape-shifting robot using particle swarm optimization. In: Fifth international conference on natural computation, 2009
3. Sun X et al (2015) A reconfiguration approach for self-reconfigurable modular robot using assisted modules. In: 2015 IEEE international conference on mechatronics and automation (ICMA)
4. Christensen D et al (2014) Collective modular underwater robotic system for long-term autonomous operation. Science 19(1):35–40
5. Hancher MD, Hornby GS (2006) A modular robotic system with applications to space exploration. In: 2nd IEEE international conference on space mission challenges for information technology (SMC-IT'06)
6. Liu S et al (2012) A reconfigurable modular robot for detection and rescue. In: Lei J, Wang FL, Deng H, Miao D (eds) Emerging research in artificial intelligence and computational intelligence. Springer, 17–24
7. Tsai C-C, Huang H-C, Chan C-K (2011) Parallel elite genetic algorithm and its application to global path planning for autonomous robot navigation. IEEE Trans Ind Electron 58(10):4813–4821

8. Soltani AR et al (2002) Path planning in construction sites: performance evaluation of the Dijkstra, A*, and GA search algorithms. Adv Eng Inform 16(4):291–303
9. Naderan-Tahan M, Manzuri-Shalmani MT (2009) Efficient and safe path planning for a mobile robot using genetic algorithm. In: IEEE congress on evolutionary computation, 2009. CEC'09
10. Guanghua Z, Zhicheng D, Wei W (2006) Realization of a modular reconfigurable robot for rough terrain. In: Proceedings of the 2006 IEEE international conference on mechatronics and automation
11. Golestan K, Asadpour M, Moradi H (2013) A new graph signature calculation method based on power centrality for modular robots. In: Martinoli A, Mondada F, Correll N, Mermoud G, Egerstedt M, Hsieh MA, Parker LE, St∅y K (eds) Distributed autonomous robotic systems. Springer, pp 505–516
12. Yim M et al (2007) Modular self-reconfigurable robot systems [grand challenges of robotics]. IEEE Robot Autom Mag 14(1):43–52
13. Rus D, Vona M (2001) Crystalline robots: self-reconfiguration with compressible unit modules. Auton Robots 10(1):107–124
14. Jorgensen MW, Ostergaard EH, Lund HH (2004) Modular ATRON: modules for a self-reconfigurable robot. In: Proceedings of 2004 IEEE/RSJ international conference on intelligent robots and systems, 2004 (IROS 2004)
15. Murata S et al (2002) M-TRAN: self-reconfigurable modular robotic system. IEEE/ASME Trans Mechatron 7(4):431–441
16. Castano A, Shen W-M, Will P (2000) CONRO: towards deployable robots with inter-robots metamorphic capabilities. Auton Robots 8(3):309–324
17. Ryland GG, Cheng HH (2010) Design of iMobot, an intelligent reconfigurable mobile robot with novel locomotion. In: 2010 IEEE international conference on robotics and automation (ICRA)
18. Haghzad S, Bagheri S, Faraji S (2013) Finding proper configurations for modular robots by using genetic algorithm on different terrains. Int J Mater Mech Manuf 1:360–365
19. Fetanat M, Haghzad S, Shouraki SB (2015) Optimization of dynamic mobile robot path planning based on evolutionary methods. In: IEEE AI & Robotics (IRANOPEN), 2015
20. Hu Y, Yang SX (2004) A knowledge based genetic algorithm for path planning of a mobile robot. In: Proceedings of 2004 IEEE international conference on robotics and automation, 2004. ICRA'04
21. Konar A et al (2013) A deterministic improved Q-learning for path planning of a mobile robot. IEEE Trans Syst Man Cybern Syst 43(5):1141–1153
22. Wang H, Yu Y, Yuan Q (2011) Application of Dijkstra algorithm in robot path-planning. In: 2011 second international conference on mechanic automation and control engineering
23. Cai C, Ferrari S (2009) Information-driven sensor path planning by approximate cell decomposition. IEEE Trans Syst Man Cybern Part B Cybern 39(3):672–689
24. Yu-qin W, Xiao-peng Y (2012) Research for the robot path planning control strategy based on the immune particle swarm optimization algorithm. In: 2012 second international conference on intelligent system design and engineering application (ISDEA)
25. Dong H et al (2010) The path planning for mobile robot based on Voronoi diagram. In: 2010 3rd international conference on intelligent networks and intelligent systems (ICINIS)
26. Janet JA, Luo RC, Kay MG (1995) The essential visibility graph: an approach to global motion planning for autonomous mobile robots. In: Proceedings of 1995 IEEE international conference on robotics and automation, 1995
27. Xin D, Hua-hua C, Wei-kang G (2005) Neural network and genetic algorithm based global path planning in a static environment. J Zhejiang Univ Sci A 6(6):549–554
28. Rimon E, Koditschek DE (1992) Exact robot navigation using artificial potential functions. IEEE Trans Robot Autom 8(5):501–518
29. Ge SS, Cui YJ (2000) New potential functions for mobile robot path planning. IEEE Trans Robot Autom 16(5):615–620
30. Guan-Zheng T, Huan H, Sloman A (2007) Ant colony system algorithm for real-time globally optimal path planning of mobile robots. Acta Autom Sin 33(3):279–285
31. Sariff NB, Buniyamin N (2009) Comparative study of genetic algorithm and ant colony optimization algorithm performances for robot path planning in global static environments of different complexities. In: CIRA
32. Brand M et al (2010) Ant colony optimization algorithm for robot path planning. In: 2010 international conference on computer design and applications (ICCDA)
33. Kala R et al (2009) Mobile robot navigation control in moving obstacle environment using genetic algorithm, artificial neural networks and A* algorithm. In: 2009 WRI world congress on computer science and information engineering
34. Zeng C, Zhang Q, Wei X (2011) Robotic global path-planning based modified genetic algorithm and A* algorithm. In: 2011 third international conference on measuring technology and mechatronics automation (ICMTMA)
35. Kang HI, Lee B, Kim K (2008) Path planning algorithm using the particle swarm optimization and the improved Dijkstra algorithm. In: Pacific-Asia workshop on computational intelligence and industrial application, 2008 (PACIIA'08)
36. Das P, Behera H, Panigrahi B (2015) Intelligent-based multi-robot path planning inspired by improved classical Q-learning and improved particle swarm optimization with perturbed velocity. Int J Eng Sci Technol 19:651–669
37. Li Y, Li C, Zhang Z (2006) Q-learning based method of adaptive path planning for mobile robot. In: 2006 IEEE international conference on information acquisition
38. Banerjee D, et al (2012) Path-planning of mobile agent using Q-learning and real-time communication in an unfavourable situation. In: 2012 world congress on information and communication technologies (WICT)
39. Li S, Xu X, Zuo L (2015) Dynamic path planning of a mobile robot with improved Q-learning algorithm. In: 2015 IEEE international conference on information and automation
40. Meng Y et al (2011) Cross-ball: a new morphogenetic self-reconfigurable modular robot. In: 2011 IEEE international conference on robotics and automation (ICRA)
41. Kuffner JJ, LaValle SM (2000) RRT-connect: an efficient approach to single-query path planning. In: Proceedings of IEEE international conference on robotics and automation, 2000 (ICRA'00)
42. Desaraju VR, How JP (2011) Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees. In: 2011 IEEE international conference on robotics and automation (ICRA)
43. Bai W, Xue B, Sun Y (2011) Research on path planning for soccer robot based on improved genenic algorithm. In: 2011 international conference on mechatronic science, electric engineering and computer (MEC)
44. Li Q et al (2006) An improved adaptive algorithm for controlling the probabilities of crossover and mutation based on a fuzzy control strategy. In: Sixth international conference on hybrid intelligent systems, 2006 (HIS'06)
45. Parlaktuna O, Sipahioğlu A, Yazici A (2007) A VRP-based route planning for a mobile robot group. Turk J Electr Eng Comput Sci 15(2):187–197
46. Sun X, Druzdzel, MJ, Yuan C (2007) Dynamic weighting A* search-based MAP algorithm for bayesian networks. In: IJCAI
47. Rosyidi L, et al (2014) Timebase dynamic weight for Dijkstra Algorithm implementation in route planning software. In: 2014 international conference on intelligent green building and smart grid (IGBSG)

48. Bozdoğan AÖ, Yilmaz AE, Efe M (2010) Performance analysis of swarm optimization approaches for the generalized assignment problem in multi-target tracking applications. Turk J Electr Eng Comput Sci 18(6):1059–1078
49. Chia S-H et al (2010) Ant colony system based mobile robot path planning. In: 2010 fourth international conference on genetic and evolutionary computing (ICGEC)
50. Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern Part B (Cybern) 26(1):29–41
51. Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285
52. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. Bradford Book, Cambridge
53. Watkins CJ, Dayan P (1992) Q-learning. Mach Learn 8(3–4):279–292
54. Huang B-Q, Cao G-Y, Guo M (2005) Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance. In: Proceedings of 2005 international conference on machine learning and cybernetics, 2005
55. Gao Q, et al (2006) An improved q-learning algorithm based on exploration region expansion strategy. In: Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on. IEEE
56. Tokic M (2010) Adaptive ε-greedy exploration in reinforcement learning based on value differences. In: Annual conference on artificial intelligence. Springer
57. Sutton RS, Barto AG (1998) Introduction to reinforcement learning., vol 135. MIT Press, Cambridge