Investigating an A-star Algorithm-based Fitness Function for Mobile Robot Evolution

Shanker G R Prabhu[§], Peter Kyberd[†], Jodie Wetherall[‡]
Department of Engineering Science
University of Greenwich
Chatham, Kent, UK, ME4 4TB
s.prabhu[§], p.j.kyberd[†], j.c.wetherall[‡]@gre.ac.uk

***Abstract*— One of the factors that affect the success of Evolutionary Robotics (ER) is the way fitness functions are designed to operate. While needs-based custom fitness functions have been developed, most of the time they have been defined in simpler mathematical functions to reduce the computation time. In this paper, we hypothesize that an incremental fitness function based on established techniques in specific task domains in robotics will aid the evolution process. An A-star algorithm-based fitness function for path planning is designed and implemented for evolving the body plans and controllers of robots for navigation and obstacle avoidance tasks. It has been shown that using this concept, fitter robots have evolved in most cases when compared to simple distance-only based fitness functions. However, due to variable performance of the evolver with the A-star fitness function, the results are inconclusive. We also identify problems associated with the fitness function and make recommendations for designing future fitness functions based on observations of the experiments.**

Keywords— Evolutionary Robotics, Co-evolution, Fitness Function, A-star Algorithm

I. Introduction

Evolutionary Robotics is a branch of Evolutionary Computation that deals with the use of Evolutionary Algorithms (EA) to evolve robots. The evolution can be performed on just the robot body plan, controller or both simultaneously. During the process, each of the evolved robots or candidate in the solution populations are tested mostly in a simulated environment with a fitness function that calculates how well it is able to accomplish the task. Using this metric, the evolver later decides whether each individual is worthy to be moved to the next phase of evolution [1]. Normally, the average fitness of the population increases gradually. If the fitness function is not an effective indicator of task attainment, the evolution process can deviate from the shortest path to the solution or even not generate a solution.

The literature shows that fitness functions developed are application specific and rely on simple calculations to arrive at the final fitness of the robot. The question to be asked is: *Does using sophisticated fitness functions make an actual difference to the speed of evolution?* In an attempt to answer this with a specific emphasis on the co-evolution process, we design multiple experiments to evolve mobile robot body plan and Artificial Neural Network controller to perform navigation and obstacle avoidance in a virtual arena. The fitness function design is based around the well-known A-star algorithm used in path planning [2].

II. Background

There are several studies that evolve mobile robots or creature body and controller with locomotion capabilities as a primary requirement, or as a part of an activity. In one of the early works, Lee et al. evolved robots for obstacle avoidance by designing a fitness function to penalize every time a robot was close to an obstacle. Here, navigation was an indirect result of the obstacle avoidance behavior [3]. In a similar strategy, to evolve line following robots, the fitness function was comprised of factors that checked if robot was on the line and whether it was moving in the right direction [4]. Authors in [5, 6] based fitness solely on the distance travelled during the simulation time, combat based fitness function were designed in [7] and fitness was proportional to food consumed in [8]. In a different approach, [9] based fitness on distance travelled and coordinated movements of joints. Distance travelled along with the suitability of body to solve a particular problem was the fitness function in [10]. A multiple point-based fitness function where fitness was allocated for reaching individual points or distance towards these points was employed in [11].

In the limited number of studies that report co-evolution, (when compared to majority of work reported in evolution of morphology or controller), to the best of our knowledge, none of them report how different fitness functions affect evolution for a particular application, nor seem to report how using established robotics algorithms in respective application affect evolution. Through this work we attempt to address these gaps in knowledge.

The paper uses the open source robot evolution software *RoboGen* [12] to perform the experiments. A Genetic Algorithm (GA) evolves the phenotype tree and an Oscillatory Neural Network based controller for tasks defined in the fitness function. A fixed set of parts which

include a controller, sensors, actuator and parametric parts participate during the process. Each of the parts except the parametric part are around *5* cm *x 5* cm *x 5* cm in size. Deterministic two-member tournaments are conducted for parent selection and (μ + λ) replacement strategy is used for updating the population.

III. Proposed Algorithm

A-star is a commonly used algorithm for performing path planning in mobile robotics. The algorithm decomposes the environment into nodes and searches for a path to the goal with the least cost. This is accomplished through an iterative calculation of a cost function at every node. The cost function is a combination of cost to goal and cost incurred to reach the current node [2].

To test the effectiveness of the fitness function, a range of obstacle avoidance tasks are designed. In each of these tasks, the evolver needs to generate a virtual robot that can move from the centre of the arena to a specified location. The task complexity is increased by gradually adding obstacles to the arena (*4 m* by *4 m*). For each task, attempts are made to produce mobile robots which are evolved with two different fitness functions. The first fitness function allocates fitness solely based on the Euclidian distance to destination at end of simulation. On a scale of *0* to *10* with *10* being maximum fitness if the robot reaches the goal. The other fitness function uses the A-star algorithm at its core to allocate fitness.

While using the proposed A-star based fitness function, the process of evaluating the fitness of each of the evolved robots are as follows: At the beginning of the fitness evaluation simulation, the map of the environment is converted into *5 cm*(the smallest size of an obstacle) grids. Then the grids that overlap with obstacles are marked to be excluded from the route calculation. Since the grid size is smaller than the size of the actual robot, to avoid the problem of the A-star algorithm finding a path that cannot be traced by the robot, the robot size is also fed to the algorithm. This is accomplished by generating a bubble of imaginary obstacles with the same size as the robot, around the actual obstacles. The result is a map with an area that the robot can transverse. Based on this information, the A-star algorithm solves for a possible grid to grid solution to reach the goal.

If a solution is found by the algorithm, the fitness function moves to the next step or the process is terminated and zero is returned as fitness. Robots that are not able to move freely to the goal are removed. The fitness function also limits the number of parts for the robot to *15* to avoid evolving long robots that reach the goal without moving. In the next step, the behaviour of the evolved robot is compared against the ideal route developed by the A-star algorithm. At every time step, the candidate robot position is converted into grid positions and a corresponding position is identified from the A-star solution. This is performed by dynamically matching the instantaneous speed of candidate robot with the imaginary robot that uses the A-star solution. The distance (d) between these robots are then accumulated over the period of simulation (*100 s*). So, the accumulated distance will be lower if the robot is able to track the A-star solution closely. Along with allocating a fixed fitness for reaching the goal, for evolving faster robots, the time taken to reach goal is also considered through the function f_{gt}. To discourage robot getting stuck at a point, for every instance of robot remaining stationary, a penalty function f_t is also included. The time taken for the robot to reach the goal is t, the final distance between robot and goal is d_f, a is the length of the arena and p is the total time robot is stationary.

At the end of simulation, a final fitness is calculated using the following formula:

$$f = f_a + f_g + f_{gt} + f_d + f_t \quad (1)$$

where,

$$f_a = (100 - \Sigma d)/10 \quad (2)$$

$$f_g = 50 \text{ if goal is reached or } 0 \quad (3)$$

$$f_{gt} = (100 - t)/10 \text{ if goal is reached or } 0 \quad (4)$$

$$f_d = 5(2a - d_f)/a \quad (5)$$

$$f_t = (100 - p)/10 \quad (6)$$

The individual components of the fitness function (except f_g) are normalised between 0 to 10. Therefore, the maximum possible fitness of any robot would be less than, but close to *90*. The fitness for reaching the goal is fixed to be *50* and since the combined fitness of the rest of the weights is less than *50*, it becomes possible to immediately differentiate a robot that has reached the destination.

TABLE I. Evolution Parameters

Parameter	Value
Population size	40
Number of evolved children	40
Probability of brain mutation	0.3
Sigma value of brain	0.7
Brain Bounds	3 : 3
Probability of node insertion	0.3
Probability of sub-tree removal	0.3
Probability of duplicating sub-tree	0.1
Probability of swapping sub-tree	0.3
Probability of node removal	0.3
Probability of modifying parameters	0.3

IV. EXPERIMENTS

In the experiments, during fitness evaluation, every simulated robot is allowed *100* seconds to solve the map and arrive at the goal and the rest of the simulator parameters are set to default values. Complete list of evolution parameters used is shown in Table 1. To speed

up the process, the experiments are performed on a High-Performance Computing System comprising of 50 nodes with 20 computing cores in each node. Ten sets of experiments are conducted for each scenario with different seeds for the random number generator in every iteration.

A. *Obstacle-less Navigation*

In this experiment, a scenario without obstacles is chosen. Each robot is placed in the middle of the arena and is expected to reach the goal *C* at (*-1.5, 1.5*). At the end of evolution, both fitness functions were able to evolve robots to reach the goal. Red and green lines to *C* in Fig. 1 plot the route traced by chosen evolved robot with the basic and A-star fitness functions (the obstacles in Fig. 1 should be neglected).

B. *Navigation with obstacles*

An arena with obstacles is designed (shown in Fig.1) and robots are evolved to reach *A* at (*1.4, -1.4*) and *B* at (*0, -1.4*) in separate experiments. It can be seen from Fig.1 that the robots were able to reach the destination except when applying A-star fitness function for point *B* (solid red colored vertical line in middle of the Fig. 1).

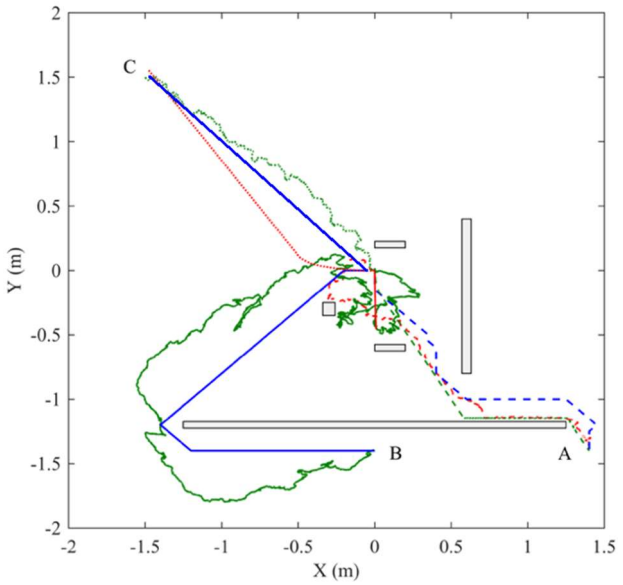

Fig. 1. Robot trajectory and ideal A-star solutions for multiple scenarios. Ideal A-star solutions are in blue, robot trajectory with A-star based fitness function are in red and basic fitness functions are in green. Routes to *A* are dashed lines, to *B* are solid lines and to *C* are dotted lines. Short vertical solid red line from centre shows that the robot is unable to reach *B*.

V. RESULTS

Despite performing experiments with multiple goal scenarios, experiments above only report three different cases as they were adequate to demonstrate the observed patterns. The fitness versus generation curves for the three different destinations are presented in Fig. 2 and Fig. 3.

Generally, the A-star based fitness function was able to evolve successful robots in all except one case when the goal was at point *B*. Even though in first experiment (to reach *C*) the basic fitness calculation was sufficient to evolve satisfactory robots, the A-star based evolver was able to evolve smaller and faster robots. Both fitness functions needed almost the same number of generations to reach the goal for the first time. But it has to be noted that the computation requirements for the A-star fitness function is significantly higher than the basic fitness function. This results in each generation taking longer to evolve when using the former fitness function.

The effect of initial seed on the evolution process is unclear. While applying the basic fitness function, results from all ten experiments showed similar variations and the output matched closely with each other (Fig. 2(a), (d), (g)). In contrast, the progress of evolution has a much wider spread with the A-star fitness function (Fig. 3(a), (d)) except when the evolver is unable to arrive at a satisfactory solution (Fig. 3(g)).

The standard deviation of each population is highest when there is a rapid change of fitness (Fig. 2 and Fig. 3). Later, the value reducing back to zero shows that the average fitness of the population is the same as the best fitness in the population. The scenario and fitness function specific rate of increase or decrease of standard deviations are identical imply that there is minimal effect of seed once the evolver stumbles at the first possible solution. After this, similar generations were sufficient to allow every individual of the population to reach peak fitness. As evident form Table 2, the standard deviation is higher for higher distance between goal and origin.

TABLE II. COMPARISON OF RESULTS

Criteria	**Fitness Function Type**					
	A-star			***Basic***		
	A	*B*	*C*	*A*	*B*	*C*
Success (%)	90	0	100	100	50	100
Worst standard deviation	22	5	26	0.45	0.25	0.515
First generation to reach goal	21	-	20	19	500	18
Last generation to reach goal	5000	-	500	90	-	90
Best fitness	72	18.8	74	10	10	9.98

When no robots were able to reach point *B* with the A-star fitness function, five experiments using the basic fitness function generated successful robots (Table 2). The detrimental effect of using a displacement-based fitness function is clear as in the five non-successful cases, the long plateaus with almost zero standard deviation in Fig. 2(g) show that robots are extremely close but unable to reach the goal which is on the other side of the obstacle (Fig. 1).

The steps in Fig. 3(a) and (d) and sudden variation of standard deviation indicate that one experiment at a time

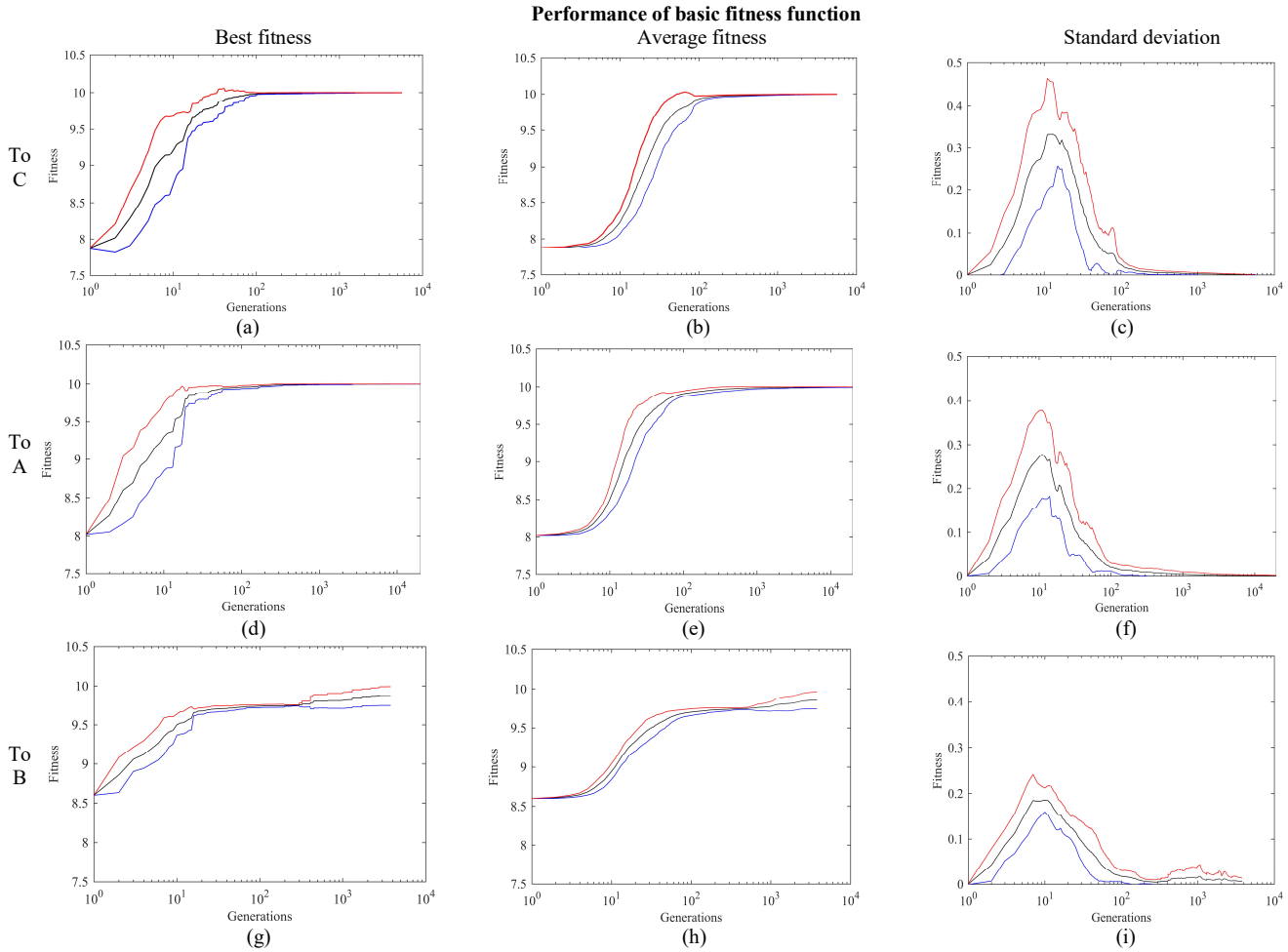

Fig. 2. The progress of evolution over generations (shown in logarithmic scale) while using basic fitness function for ten experiments in each scenario. Black line is mean for 10 experiments and red and blue lines are one standard deviation above and below average respectively. (a, d g): Mean best fitness measured while reaching points C (a), A (d) and B (g). (b, e, h): Mean of average fitness of entire population over generations for reaching points C (b), A (e) and B (h). (c, f, i): Mean of standard deviation of fitness of entire population for reaching points C (c), A(f) and B (i). Fitness of 10 shows that the robot has reached the goal. Observations from all three scenarios show similar time taken to reach the goal despite exhibiting variation between overall maximum and minimum fitness in each case. An exception is while reaching B when half of the experiments did not converge to a solution in the given timeframe. In these cases, the robots moved to the farthest point along displacement until an obstacle blocked the path.

converged to a satisfactory solution. The horizontal overlapping line segments in Fig. 3(c), (f) and (i) suggest that all ten experiments behaved similarly during that period of evolution.

While evolving robots for obstacle avoidance, the two key findings were that the controller was unable to perform large direction changes after the first quarter of the fitness evaluation. Even when obstacle sensors were present, despite running the evolution process for 100,000 generations, their suitable utilisation was not observed. The change of direction was either the result of collision with obstacles (Fig. 1) or open-loop control without using obstacle sensors. When an obstacle sensor was present, it was not placed in the direction of motion. All these suggest the need for improving the evolution of controller during the evolution process. The evolver generated long robots when it was not restricted in terms of the number of parts a robot could have. This allowed robots to reach the goal with minimal set of movements.

A surprising finding was how the evolver became stuck by not evolving a robot to move past an obstacle in several cases particularly while trying to reach *B* (short vertical red line in Fig. 1). The initial hypothesis to explain this was that it did not receive sufficient encouragement from the fitness function to move past that point. However, step-by-step analysis of the fitness allocation showed that this was not the case. As a result, the A-star based evolution was unable to generate robots to reach *B* despite running the evolution for around 37,000 generations after which maximum computation time (7 days) exceeded.

In contrast, the basic fitness function needed just 500 generations in the best case and close to 8,000 generations in the worst case to solve the same problem (Fig. 2(g)). But as seen from Fig. 1 (green line), it can be safely stated that the route taken was not close of the optimal solution, and A-star solutions were better in this regard.

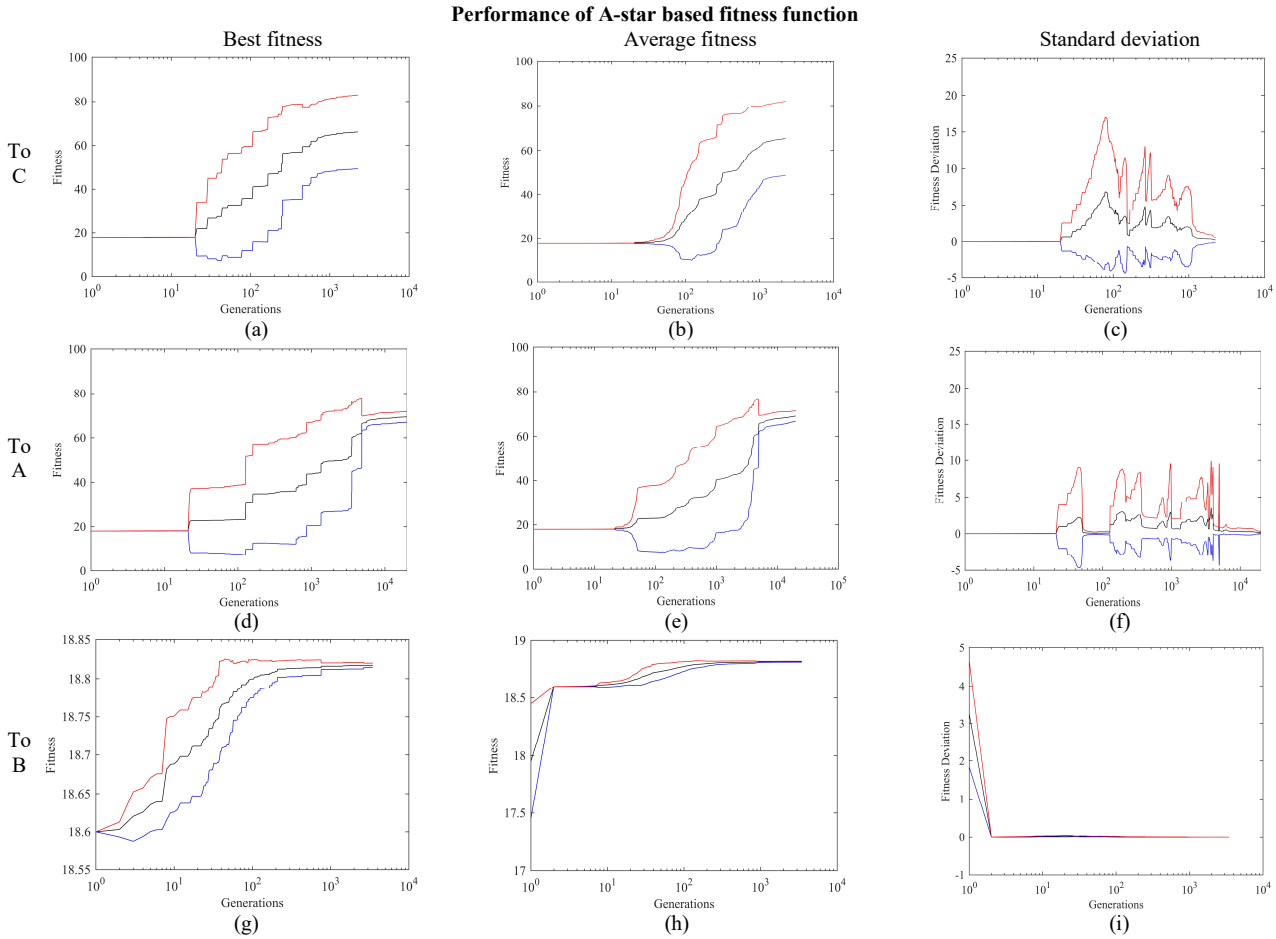

Fig. 3. The progress of evolution over generations (shown in logarithmic scale) while using basic fitness function for ten experiments in each scenario. Black line is mean and red and blue lines are one standard deviation above and below average respectively for ten experiments. (a, d g): Mean best fitness measured while reaching points C (a), A (d) and B (g). (b, e, h): Mean of average fitness of entire population over generations for reaching points C (b), A (e) and B (h). (c, f, i): Mean of standard deviation of fitness of entire population for reaching points C (c), A(f) and B (i). Fitness of 50 shows that the robot has reached the goal. The experiments converged to solutions at different times (as shown by standard deviation curves) except while reaching B depending on the chosen seed for the random number generator.

VI. Discussion

Since the fitness function is designed to help a robot stay on or close to the A-star route, ideally the robot should have been able to make turns when necessary. But this was not observed which questions the effectiveness of the fitness function.

To reduce computation time, the A-star fitness function only solves the map once at the beginning of evaluation (depending on the robot size). In the next generation when the controller gets reused, the A-star solution could generate an entirely different path despite when the old path might still be a valid solution (but not the optimal solution). Moreover, since the fitness function is fixated on one ideal solution, instead of allowing the robot to focus on reaching the goal, it ends up trying to reach the ideal route and ultimately failing both. From another perspective, the fitness calculation does not consider the robot's capability to reach goal from its instantaneous position. An A-star solution can exist from the real-time position and this could have been considered.

After solving the map, a robot is discarded if it would not be able to reach the goal because of its large size. From an evolutionary perspective, the robot could have been allowed to survive and allocated a non-zero fitness based on how close it was to the goal.

The fitness function only considered the trajectory of the robot to allocate fitness. A proper mating of morphology and controller is essential in the co-evolution process for a robot to succeed. This suggests the need for incorporating fitness functions which perform an in-depth analysis of both morphology and controller before making a decision. On the other hand, this new method would apply immense strain on the computational resources and might question the need for using EAs.

In the experiments, the robot can navigate to a solution via blindly remembering a path to the goal by constantly searching for the best possible fitness at every time step or

have a controller react to the environment based on real time feedback from sensors or by using a combination of both. For simplicity if we assume that the controller is adopting the first option, at the end of simulation, the robot could take 10 correct steps out of the 100 possible steps in the best solution only owing to the randomly generated oscillatory neural network. In the next step, if the robot is chosen as a parent, during variation operations, the evolver does not know which part of the neural network is to be modified through mutation and crossover as the fitness function does not shed any light to aid in this regard. The long plateaus during evolution and one of the reasons why more than 95% of the works in ER are in evolving controllers rather than in co-evolution could be explained with this.

A common observation in all the experiments above are that the standard deviation of populations reaches zero for the majority of the time during long runs of evolution. Even though this is a fundamental problem associated with ER, fitness functions are also to blame. Rather than evolving diverse robots which excel in different aspects (a robot might have an efficient morphology or superior controller), because every robot is given a single fitness value, differently skilled robots do not progress further. Therefore, parent selection should consider fitness as a group of individual fitness corresponding to each skill and the same information needs to be used while mating. The counter argument here could be that evolution would then be subject to bias and thereby moving away from true random nature.

Further, the fitness function is not testing the capability of the controller to achieve the task at hand. We think that with the help of fitness functions, the evolution can be speeded by prima facie eliminating the robots whose controller does not show any promises of solving the problem. For instance, if the task is obstacle-less point-to-point navigation, the robot just has to make a single turn to the direction of goal and move towards it in a straight line. This can be accomplished by a simple neural network with a handful of connections. On the other hand, if the task is to handle complex obstacle avoidance, the simple neural network may not be sufficient. A similar approach can also be applied on the morphology by discouraging robots which does not have the potential to solve the problem.

The comparison of results from both fitness algorithms are from the perspective of evolving successful solutions. Due to the massive amount of post-processing involved, in-depth statistical analysis of evolved robots is not performed above. We believe deductions could be further improved by analyzing the other multiple factors such as trajectories, size, shape, power consumption and by physically testing the best individual from each generation. Another possible solution could be changing the linear time independent allocation of fitness to adaptive fitness functions which dynamically change of over the lifetime of each robot and at different stages of evolution.

The use of EAs in robotics to find solutions to deterministically solvable problems is an ongoing debate. For instance, it could be argued that instead of using an A-star based fitness function, a robot controller could easily be designed with the A-star algorithm for path planning. However, it has to be stated that such an approach only works if the robot body is fixed during evolution as different morphologies warrant different controllers and a single control approach will not be appropriate. Further, the main premise of ER is to arrive at non-obvious solutions to problems or automatically generate solutions from scratch without human intervention as in this case.

VII. Conclusion

In an effort to co-evolve morphology and controller of robots for navigation and obstacle avoidance, our fitness function allocated fitness based on the behaviour of the robot in comparison with a possible A-star solution to the problem. Results, when compared with a basic fitness function that just uses Euclidean distance to allocate fitness, showed mixed results thereby not giving conclusive evidence to validate our initial hypothesis which said that using established task specific algorithms can indeed aid evolution. Since the focus was on applying algorithms just for path planning and as co-evolution depends on several other factors which were not considered, the full benefit of the A-star based fitness function were not observed. This points to the need for developing fitness functions derived by combining functionality-based algorithms in the future. Finally, for the benefit of future researchers, we also discussed several recommendations for designing fitness functions.

VIII. Future Work

Both set of experiments indicated that distance or goal only based fitness function is not sufficient to evolve fitter robots. Therefore, the first step could be to gradually add factors discussed in *Section 6* during fitness function development. They could be through applying the morphological suitability, controller suitability and behavioural assessment factors in fitness function. Simultaneously, the fitness function structure could also be changed to vary over the period of evolution for allowing a gradual addition of skills over the evolution. For comparison, the factors neglected due to high computation cost can also be incorporated.

References

[1] S. Nolfi, and D. Floreano, "*Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*", MIT Press, Cambridge, MA, USA, 2000. doi: 10.1162/106454601317297031

[2] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. doi: 10.1109/tssc.1968.300136

[3] Wei-Po Lee, J. Hallam and H. H. Lund, "A hybrid GP/GA approach for co-evolving controllers and robot bodies to achieve fitness-specified tasks," *Proceedings of IEEE International Conference on Evolutionary Computation*, Nagoya, 1996, pp. 384-389. doi: 10.1109/ICEC.1996.542394

[4] H. Lund, "Co-evolving Control and Morphology with LEGO Robots," *Morpho-functional Machines: The New Species*. Springer Japan, Tokyo, 59–79, 2003. doi:10.1007/978-4-431-67869-4_4

[5] J. Pollack and H. Lipson, "The GOLEM project: Evolving hardware bodies and brains," *Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pp. 37–42, 2000. doi: 10.1109/EH.2000.869340

[6] A. Faiña, F. Bellas, F. Orjales, D. Souto, and R. Duro. "An evolution friendly modular architecture to produce feasible robots," *Robotics and Autonomous Systems,* vol. 63, no. 2, pp. 195–205, Jan 2015. doi: 10.1016/j.robot.2014.07.014

[7] T. Miconi, "In Silicon No One Can Hear You Scream: Evolving Fighting Creatures," *Genetic Programming*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 25–36, 2008. doi: 10.1007/978-3-540-78671-9_3

[8] M. Pilat, T. Ito, R. Suzuki, and T. Arita, "Evolution of virtual creature foraging in a physical environment," *Proceedings of the 13th International Conference on the Simulation and Synthesis of Living Systems (Artificial Life XIII),* MIT Press, pp. 423–430, 2012. doi: 10.7551/978-0-262-31050-5-ch056

[9] M. Mazzapioda, A. Cangelosi, and S. Nolfi, "Evolving morphology and control: A distributed approach," *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009).* pp. 2217-2224, 2009. doi: 10.1109/cec.2009.4983216

[10] J. Auerbach and J. Bongard "Evolving complete robots with CPPN-NEAT: the utility of recurrent connections," *Proceedings of the 13th annual conference on Genetic and evolutionary computation (GECCO '11),* Natalio Krasnogor (Ed.). ACM, New York, USA, pp. 1475-1482, 2011. doi: 10.1145/2001576.2001775

[11] E. Samuelsen, K. Glette, and J. Torresen, "A hox gene inspired generative approach to evolving robot morphology," *Proceedings of the 15th annual conference on Genetic and evolutionary computation (GECCO '13).* ACM Press, pp. 751–758, 2013. doi: 10.1145/2463372.2463464

[12] J. Auerbach, D. Aydin, A. Maesani, et al., "RoboGen: Robot Generation through Artificial Evolution," *Proceedings of International Conference on the Synthesis and Simulation of Living Systems (Artificial Life XIV).* MIT Press, pp. 136–137, 2012. doi: 10.7551/978-0-262-32621-6-ch022