Applying Evolution to Cooperative Warehouse Robots

Noah Zemlin - C S 5073 Large Project

May 6th, 2021

Abstract

Warehouse inventory movement is a very laborious task that is being improved by using robotics. Dozens or hundreds of robots can be employed to move items quickly, efficiently, and tirelessly about a warehouse to their destination. By using machine learning techniques, we can train the robots to further optimize their movement about the warehouse and even introducing complex tasks such as passing items between each other. In this paper, we explore using evolution to train a neural network that controls the robot end-to-end from their inputs. We see that the approach does indeed work and found that cooperation can be taught to the agents without explicit reward for doing so.

I. Introduction

Warehouse robotics is not a new or novel field of robotics as robots have employed in warehouses for decades to assist workers in moving about objects efficiently. The novelty that we will look at is the ability for robots to not only assist workers but to entirely replace workers from having to perform the laborious workhouse tasks. Innovations in robotics research now allows for highly intelligent robots with near-perfect path planning and navigation. Enhancements to robotic limbs also allow robots to easily grab items from shelves and move them. Modern computer vision allows for robots to accurately describe their environment and recognize objects. While none of these problems can really be considered solved, the future of all robot warehouses is becoming less sci-fi and more modern.

One problem still facing the field, however, is inter-robot communication and work. Robots work efficiently when given a set task to do with specific static instructions. To fully replace workers in a warehouse, we need the robots to be able to evaluate their environment and make autonomous decisions. We also would like to scale up the number of robots to further increase the throughput.

As with many modern problems, an attractive approach to solving this intelligence problem is to use machine learning. Genetic algorithms tend to be highly effective in problems where there is a specific numerical goal in mind as the agents will optimize towards that goal with respect to their gene encodings. Effective approaches include algorithms like NEAT [1] which evolve neural networks using evolution.

In the paper “An Evolutionary-Cooperative Model Based on Cellular Automata and Genetic Algorithms for the Navigation of Robots under Formation Control” from G. M. B. de Oliveira, et al. [2], they implemented cooperative robots using evolution. Their robots were tasked with maintaing an equidistant formation while travelling through an environment full of obstacles, mimicing a search and rescue through debris. Using evolution allowed them to train the robots for many different environments and tune the performance of the agent to the specifications they wanted by tweaking the fitness function accordingly.

In this paper, we will cover our implementation of a simple warehouse inventory robot that is tasked with moving items about a warehouse to a delivery

point. To train the agent, we will use a neural network to decide the agent's actions based on its knowledge of its environment. Evolution is then used to train the weights of that network so that the agent can maximize the goal of moving as many items as possible. To evaluate if cooperation can be taught, we will allow the agents to pass objects between each other but not reward them for explicitly doing so.

After implementing the model, we found that the agents were able to learn to pass objects and perform better by doing so. This shows that applying evolution to training neural networks is a valid solution to the problem presented and that it can likely be applied to more complex versions of similar problems as well.

II. Approach

Broadly, our approach will consist of a population of agents which each consist of a neural network that defines the actions each agent will take based on its surroundings. To train these networks, we will encode the weights as the genes for a genetic algorithm and run standard evolution computation on the genes. Each gene will be evaluated for its fitness in a simple simulated environment and then a new population of individuals will be created, repeating this process for several generations until learning occurs.

To evaluate the problem, first we need an environment. For this project, we will use a simple 2D discretized environment that our robots will interact with. Each space is either an empty space, a wall, a robot, or an object. Robots can move from their current grid space to another adjacent space. Robots can also pick up an adjacent object and store the object as part of it (the object no longer takes up a grid space). Figure 1 shows an example screenshot of a randomly generated environment in the simulator.

To allow the robots to make decisions in the environment, we will use an end-to-end neural network that acts a function form input to actions. The inputs will be neurons that encode the type of object in adjacent spaces as well as which space the robot should move to find the nearest object or delivery space. The output will be a layer of neurons that each encode an action that the agents can take. The softmax activation function is applied to allow the network to prioritize an action and the highest activation is the action chosen by the agent. Each hidden layer uses a simple sigmoid activation function and is densely connected between the two adjacent layers. The neural network architecture will be pre-defined and static throughout the experiments. The weights will be initialized randomly using a normal distribution but will be allowed to change. Some weights will be initialized to zero so that the agents can disable some weights that are not needed.

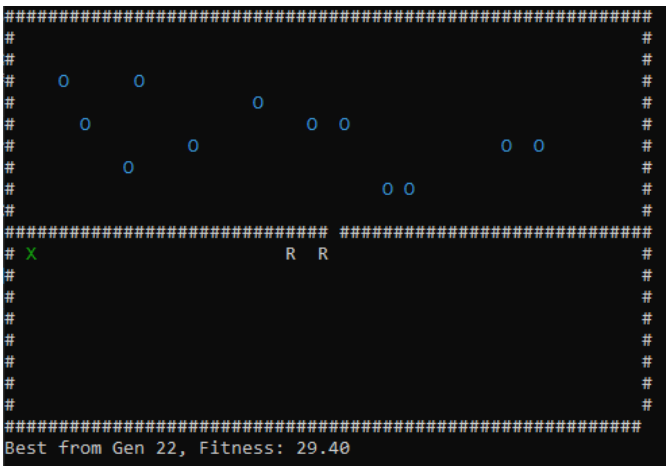

Fig. 1. Example environment in the simulation. "#" symbols are walls and "R" is robot. Note the bottleneck in the middle where only one robot can fit at a time.

To evolve our agents, we will a genetic algorithm that will optimize the weights of the neural networks. The genes that will be trained will simply encode the weights of the network by containing a list of all the weights. Thus, an individual is one set of weights for the network. We will start with an initial randomly generated population and then evaluate each agent in the environment for a set amount of time. The fitness function that we will evaluate the individuals against will simply be the number of objects they are able to move in that set amount of time.

After all the agents get a chance to run, we will then perform standard evolutionary computation procedures to generate a new generation. First, we will perform selection to choose which agents will create new children for the next generation for which we will use proportional selection. This rewards the best-performing individuals by allowing them to produce more offspring like them but also allows individuals with potential to also produce offspring. To produce the offspring, we will perform simple crossover where we will simply split each of the two parent's genes into two splits and distribute them among the children such that each child has one split of genes from each parent.

Finally, we will perform mutation on the new offspring to introduce additional variety in our individuals. For each offspring, they can receive one or more of the following mutations, each at a low (<5%) chance: set a weight to zero, reinitialize a weight, or add a small value (25% of the initialization standard deviation) to a weight. Multiple weights can be affected. These mutations allow for both local and global exploration of the fitness surface.

III. Experiments and Results

To evaluate the model, we performed 10 runs and took the average fitness and passes over the best and median agents over the generations. For each run, we trained using a population of 120 individuals and over the course of 50 generations.

Three different experiments were conducted. The first experiment consists of only one robot and the second experiment added a second robot. Both of those experiments disabled the passing action (that is the neuron corresponding to that action in the output layer was removed). For the third experiment, we used two robots and allowed them to pass between each other.

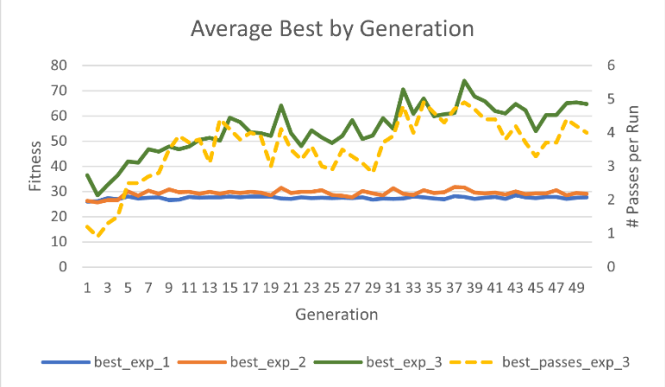

Fig. 2. Average performance of best agent by generation

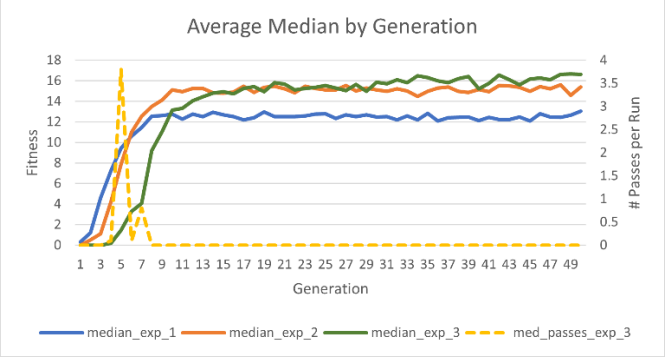

Fig. 3. Average performance of median agent by generation

The results of the experiment are shown in figures 2 and 3 above. Looking at figure 2, we can see that if we look at the average best agent each generation that the agents in experiment 3 were able to outperform the agents without passing. In fact, we can see that there is little performance improvement between one and two robots. The bottleneck is so significant that the two robots spend just as much time attempting to get through the corridor as spent grabbing objects. By allowing passing, the third agent learns to use passing to overcome this bottleneck even without direct reward for doing so. We can also see that the agents learn to pass more over training, further emphasizing that the agents are learning to cooperate.

Figure 3 shows somewhat contradictory results to what was seen in Figure 2. The two robots without passing are significantly able to outperform the single robot, despite the bottleneck. Additionally, the robot with passing is not able to perform any better than the robots without passing. In fact, the median agent seems to never learn to use passing, likely settling for just being able to score high enough to stay in the median. This may also be a result of the agents getting stuck in constantly passing to each other, leading them to learn to stop passing so they can at least gain fitness by completing the tasks sub optimally.

Our results compare similarly to the paper by G. M. B. de Oliveira, et al. [2] discussed earlier. Our agents, as well as theirs, can quickly create a few agents with high fitness while the median agent slowly catches up.

IV. **Conclusion**

From the results, our approach is a successful way to accomplish the desired task of cooperative warehouse robots. The results show that the agents were able to learn to cooperate without directly being rewarded for doing so. This shows that, given a better model with more efficient training, more complex environments and more complex cooperation abilities will also likely be able to be trained using evolution.

For future work, we would like to further scale up the experiment to include many more robots and more complex environment with moving hazards. Additionally, using NEAT [2] to evolve the network topologies may also produce more interesting and diverse results. A significant downside of this project is the lack of network topology adaptability and the agents may have been able to perform better with a smaller or larger neural architecture.

V. **Bibliography**

[1] Stanley, Kenneth O., and Risto Miikkulainen. "Evolving neural networks through augmenting topologies." Evolutionary computation 10, no. 2 (2002): 99-127.

[2] G. M. B. de Oliveira, R. G. O. Silva, L. R. do Amaral and L. G. A. Martins, "An Evolutionary-Cooperative Model Based on Cellular Automata and Genetic Algorithms for the Navigation of Robots Under Formation Control," 2018 7th Brazilian Conference on Intelligent Systems (BRACIS), Sao Paulo, 2018, pp. 426-431, doi: 10.1109/BRACIS.2018.00080.