Report

Beatrice Jiang

December 6, 2023

1 Strengths and shortcomings

For part 1, an A* search and a breadth first search algorithm have been implemented in such a way that if the destination is known to the agent, then an A* would be applied, otherwise we search with breadth first search. Instead of the default depth first search, which is not a complete search, this strategy makes sure that the shortest path is executed with lowest cost of energy. The agent goes to the charging station by making a comparison of the cost and energy level. Additionally, a dynamic charging station strategy has been incorporated to optimize energy consumption.

In Part 2, a suboptimal strategy has been implemented, where the agent recursively seeks the nearest oracle, visits it, and stops after visiting all oracles or determining their identities. The approach significantly reduces the time and energy required compared to brute force, with fewer visits to charging stations. An attempt to store the charging station have been made but were not utilized due to performance concerns. This stems from the fact that when there is only one charging station that has been visited, the agent would go there even when there is a charging station just nearby. Also, when the agent has done an A* search to the charging station it stored, finding there is not sufficient energy, then it searches by BFS, wasting the search. Hence, caching the station delays the decision-making process. To boost a faster performance, what could be done is to scan the whole map at the start of program and have an internal representation of the map to reduce the number of http requests.

In Part 3, an advanced technique involves maintaining the state of each agent, utilizing a tree-like maze structure, and storing paths, visited nodes, and agendas. Additionally, strategies such as reordering agents to prevent collisions and implementing synchronized movements were applied to enhance overall performance and coordination. This facilitates efficient multi-agent coordination, preventing redundant exploration and ensuring agents are aware of each other's states. When an agent reaches the maze exit, all the other agents stops exploration, calculate and move along their shortest paths. By employing a stochastic heuristic in scenarios with multiple paths, the agents can effectively disperse and avoid consistently traversing already explored branches.

These techniques generalize well to real world problems with suitable conditions. With A* , it requires a consistent heuristic function. For scanning and storing the map, this requires a reasonably small map, otherwise, the overhead for scanning though might take longer than searching for the problem itself.

2 Real world applications

In the context of real-world search problems, such as finding a hidden object in an unexplored environment, subject to the little information of the problem, a genetic algorithm is appropriate in this situation. The uncertainty of agent is making progress through generations pose challenges in designing an effective fitness function. To address these challenges, coevolutionary approaches can be applied to multiple agents across a variety of mazes.

Under coevolution, each agent is awarded for exploring an unvisited node. Moreover, when the agent reached the goal first, it will be rewarded with a highest mark. In cases where no agent reaches the goal, the maze itself is awarded. In this way, agents can behave more sophisticated and potentially leading to more efficient exploration and faster convergence to solutions.

The problem of part 2 in real world might be search and rescue, the goal is to maximize the rescues in a limited amount of time. Compared to a hill climbing algorithm, a genetic algorithm can perform better as it is less likely to get stuck in a local optimum. To address this problem, a genetic algorithm

can be applied with a fitness function based on the number of rescues, using the move as it's genotype. Running this for many generations is expected to produce a strategy that maximized the number of rescues.

Even though genetic algorithms are proved to balance exploit and exploration at the same time [1], this is based on a few assumptions that is not usually the case in practice, These assumptions are

1. the population size is infinite
2. the fitness function accurately reflects the utility of a solution
3. that the genes in a chromosome do not interact significantly

Not to mention that in reality rarely population is infinite, despite applying a dynamic fitness function, there is no way to verify it actually reflects the utility of a solution. On top of that, if moves were used as genotypes, then it would pose a dependence on multiple genes, which is a defect in the accuracy of the algorithm. In addition, genetic algorithms are much slower compared to deterministic ones because of its inherent nature of small mutation in thousands of generations. Hence, using a genetic algorithm might not be the best choice in a time sensitive task. And generally, for real world problems, there is some partial decomposability, some epistasis in there, so an algorithm exploiting that might be the best.

As much as we would like to optimise our algorithms, there is no free lunch for optimisation [2]. When algorithm performance is averaged across all possible problems: No optimization algorithm can do better than blind random search. In real world cases, a random search might just be the best option.

References

[1] Jeffrey R Sampson. Adaptation in natural and artificial systems (john h. holland), 1976.

[2] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.