**Improving Autonomous Agent Search in Real-World Environments: Strengths, Shortcomings, and Potential AI Techniques**

## Introduction

In this report, the application of artificial intelligence techniques to control autonomous agents in search tasks will be explored, extending the solutions developed in previous coursework (Parts 1–3) to real-world analogues. The original tasks involved pathfinding in static grids, deducing identities by querying oracles, and multi-agent maze exploration. While these approaches demonstrated foundational AI methods, scaling them to dynamic and complex real-world environments presents challenges. This report analyses the strengths and shortcomings of the implemented approaches and discusses potential AI techniques to enhance agent performance in real-world search problems involving dynamic environments with moving obstacles.

## Strengths and Shortcomings of Implemented Approaches

***Strengths***

The solutions from Parts 1-3 demonstrate a good basis for AI techniques that could be extended for real-world applications. Namely:

* **Efficient Pathfinding (Part 1)**: The implementation of A\* allowed agents to find shortest paths to targets, while taking energy constraints into consideration. This provides a good foundation for optimal pathfinding in static, predictable environments. In real-world problems, such techniques could be leveraged in robotic systems navigating hazardous environments, such as search-and-rescue drones.
* **Route Planning (Part 2)**: This portion of the coursework extended the efficient A\* implementation to create more complex routes, taking into consideration energy expenditure. The solutions were concrete and, although not optimal, managed to successfully visit all possible oracles in the vast majority of worlds. This would be helpful in environmental monitoring tasks (e.g. surveying multiple ecological hotspots) where energy efficiency impacts operational range and coverage area.
* **Multi-Agent Exploration (Part 3)**: Here a very basic version of a multi-agent system took form, where agents cooperatively explored an unknown maze, uncovering the environment through local observations, illustrating basic multi-agent coordination. Agents could successfully leverage some very basic, simplistic shared information for more efficient exploration. Swarm robotic-like tasks in wider area search-and-rescue, or distributed monitoring of ecosystems using multiple agents could benefit from such algorithms.

***Shortcomings***

Despite the above forming a good foundation, there were many shortcomings in the implementations:

* **Inefficient Resource Management (Parts 1 and 2)**: Pathfinding was sometimes slow, especially when exploring all possible charging stations. Additionally, the pathfinding was suboptimal. The agent lacked logic to utilize nearby resources to construct more complex paths, and only “tunnel visioned” on reaching a single target at a time in the shortest path possible, ignoring additional nearby targets that could be utilised to create a longer, but more rewarding path. This resulted in extra steps and energy consumption. In the aforementioned search-and-rescue operations, this could delay locating survivors or waste critical resources.
* **Agent Conflict and Deadlocks (Part 3)**: Agents frequently clashed and caused deadlocks due to the lack of sophisticated conflict resolution mechanisms, leading to inefficient exploration and pathfinding failures. This caused reduced efficiency, and (albeit shortly lived) bottlenecks at times. In real-world multi-agent scenarios, this can lead to mission-critical delays or resource bottlenecks, compromising the system’s effectiveness.
* **Static Environment Assumptions**: Both algorithms assumed static environments with no dynamic changes, limiting their effectiveness in real-world settings where obstacles and goals may move unpredictably. Without the ability to adapt to terrain changes, real-world applications of agents risk becoming ineffective in many disaster (or otherwise) related scenarios.
* **Scalability Issues**: The computational overhead of certain operations (e.g., processing HTTP requests in Part 2) led to performance bottlenecks, which would be exacerbated in larger, real-world environments. Such delays in decision-making could result in critical time-loss under high-pressure scenarios.

## Potential AI Techniques for Enhanced Performance

Addressing the outlined shortcomings, specifically through the lens of a dynamic environment in which obstacles, as well as targets/goals are constantly changing , various AI techniques can be employed. This exploration will focus on the following two:

**1. Reinforcement Learning via the Q-Learning Algorithm:**

Reinforcement Learning allows agents to utilise trial and error to gain the necessary experience in order to learn optimal policies for a given environment. This can lend itself well to agent adaptability, as opposed to, say, supervised learning where there exists a set with predefined answers. The added ability to make decisions based on probabilistic outcomes is essential in environments where the effects of actions are not deterministic.

The Q-Learning algorithm is an efficient way to employ reinforced learning in such a system. To briefly explain its functionality, agents learn a Q-value function , representing the expected utility of taking action in state . The estimation of Q-values can be determined either through **Temporal Difference** (comparing the current state and action values with the previous ones), or via **Bellman’s Equation** (recursively calculate the value of a given state, aiming to determine its optimal position). Environmental features such as nearby obstacles, energy levels and target locations should be incorporated in the state representation, allowing agents to perform optimal actions per state. Agents should be rewarded (or punished) depending on actions in such a way as to encourage reaching goals, avoiding obstacles, and conserving energy. This would mean highly positive rewards for reaching a goal, highly negative for hitting obstacles, and likely mildly negative ones per time step to promote efficiency. Lastly, strategies like ε-greedy could be employed to balance exploration of new actions and exploitation of known good actions.

The above offers great flexibility, allowing for agents to adapt to new situations easily (provided overfitting is accounted for), coupled with scalability, enabling agents to handle large and complex state spaces. However, training time (and computational cost) will be a clear issue, especially given the sheer number of different substantial interactions with the environment agents can/should have during learning. Additionally, collecting training data for real-world applications (e.g. search-and-rescue) may be quite costly, and designing good reward functions can prove quite challenging.

**2. Dynamic Pathfinding via D\*:**

D\* Lite is an incremental heuristic algorithm, building upon A\*. It focuses on updating only the affected parts of the path whenever changes are detected. It is designed for environments where changes occur after the initial path has been planned, and it aims to ensure that the path remains (near) optimal even as the environment changes. Essentially, it reuses previous search efforts to, via consistent heuristics, efficiently replan paths when changes occur, reducing computational overhead.

An implementation of the above could build upon the existing codebase, by computing an initial path from the start to the goal using A\*, while constantly monitoring the environment in order to observe changes. When a change is detected, the cost of affected nodes can be updated, and the path from the current position can, thus, be adjusted accordingly.

|  |
| --- |
| function DStarLite(start, goal):  Initialise the priority queue with start node  while (current ≠ goal):  current = priorityQueue.pop()  if (current is changed):  Update current's cost  for each neighbour in current.neighbors:  if neighbour's cost changed:  Update neighbour's cost  priorityQueue.update(neighbor) |

*(Figure 1: Proposed Pseudocode for D\* Lite Algorithm)*

This approach is useful due to its computational efficiency and minimised processing time. It also provides a good base for real-time agent adaptations in situations where the environment changes frequently. However, there are a few drawbacks relating to the algorithm’s dependence on the agent’s ability to accurately detect changes in the environment. Additionally, frequent changes in highly dynamic environments can still, despite its efficiency, impose computational burdens.

Encompassing techniques such as the above would be useful in situations such as search-and-rescue scenarios in dynamic disaster zones. Dynamic obstacles such as rubble and debris may shift due to aftershocks or structural collapses, while survivors may be moving, searching for exits, relocating or panicking. Similar semantics apply to capturing/looking for fugitives on the run, environmental disaster monitoring, surveillance, and much more.

**Conclusion**

By properly leveraging and incorporating these advanced AI techniques, agents can overcome the limitations faced by the initial implementations. RL can provide adaptability to dynamic environments, while D\* Lite equips agents with efficient real-time pathfinding. The potential for autonomous agents to perform effectively in real-world resource-limited spatial search problems can be significantly improved using the above.