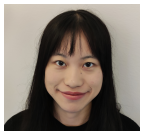
Coordination and Collaboration of Autonomous vehicles

GROUP 11

Marco Schouten
1998-05-22
schouten@kth.se

Xuecong Liu
1997-11-17
xuecongl@kth.se



Abstract

Multi-agent systems can solve problems that are difficult or impossible for an individual agent to tackle, and the coordination and collaboration between autonomous agents is the key to bring such systems to their full potential. In this project, a set of autonomous vehicles are coordinated to collaborate on a variety of problems, in a simulated UNITY environment with obstacles to avoid. We use Minimum Spanning Tree (MST) to perform Vacuum Cleaner Planning within a short time, and then apply DFS and BFS to solve Vehicle Routing Problem, which we combined with Minimum Set Cover algorithm to solve Indoor UGV search. We implemented V-shape formation for formation sweep, and Dijkstra algorithm for planning shooter coordination of three vehicles. In this report, we are going to discuss these algorithms and their rooms for improvement in detail.

1 Introduction

A Multi-agent system is a system composed of multiple agents which interact with each other to solve a task. Smooth cooperation between the agents usually yields better performance than a single agent system, opening up new possibilities. In this research, we addressed five different problems regarding multi-agent interactions: Vacuum Cleaner Planning (short-range), Indoor UGV Search (long-range), Vehicle Routing Problem, Formation Sweep, and Shooter Coordination.

Vacuum Cleaner Planning is a problem suitable for short-range agents. The problem is to find a path for each agent in such a way that every point of the space in which they navigate has been within the range of action of at least one robot.

Indoor Unmanned ground vehicles (UGV) Search predestines extensive use for autonomous agents which are well suited to perform with excellent accuracy repetitive tasks. A relevant application is about long-range surveillance robots [1] which scan a closed area repeatedly to find intruders.

Vehicle Routing Problem is the problem in which several agents have to reach a list of destinations in the shortest possible distance [2]. This can be addressed as an extension of the Travelling Salesman Problem for multiple agents.

Formation Sweep is about coordinating the movement of multiple agents such that a particular formation was kept. A key property was to adapt the shape of the formation for narrow roads.

Shooter coordination is about strategically coordinating a platoon of agents, such that it was capable of defeating a set of enemies. This problem is divided into several sub-problems. Firstly, to form a strategic formation, then to find the optimal location for the platoon, lastly to find the optimal path that led the platoon to that formation.

1.1 Contribution

In this project, we implemented a variety of ad-hoc algorithms to deal with each of the five problems optimally.

In P1 (Vacuum Cleaner Planning problem), we used an MST which outlined the edges in such a way that if the vehicles circumnavigated those edges, every point of space was within the firing range of the vehicle.

In P2 (Indoor Unmanned Ground Vehicles Search problem), we implemented a Minimum Set Cover algorithm to find a set of Guards, then we performed a randomised search for each cluster of guards, lastly, we used

Breadth-First Search to find the path connecting each guard in the right order.

In P3 (Vehicle Routing problem), we performed a randomised search for each cluster of the checkpoint that we needed to reach.

In P4 (Formation Sweep problem), we built symmetric V shape formation.

In P5 (Shooter Coordination problem), we used Dijkstra's algorithm to find the shortest path on a weighted graph combined with heuristics that assigned weights to each node and edge according to the configuration of allies, enemies and obstacles.

1.2 Outline

In Section 2 we explore the state-of-the-art techniques and available literature for each of the five coordination and collaboration problems to provide an overview of the pros and cons of each strategy to choose critically the most suitable method. In Section 3 we explain the details of our implementations. In Section 4 we analyse the results of our findings.

2 Related work

Here we investigate the available literature for each of the five problems.

2.1 Vacuum Cleaner Planning

Prim's algorithm was used in this project to find the MST. The algorithm was first developed by mathematician Vojtěch Jarník [3] and then republished by computer scientist Robert C. Prim [4]. Given a graph with weighted edges, the key idea of the algorithm is to initialise a tree with one vertex and then grow the tree by iteratively choosing one edge that connects the tree to a vertex outside the tree with the minimum weight. The MST found by Prim's algorithm can be used to search continuous space with Spanning Tree Covering [5].

2.2 Indoor UGV Search

Indoor UGV Search is a flourishing area of research. A recent application study is conducted by Lakas et all [6] where they implemented A* for path planning for each agent between checkpoints. However, to generate the checkpoints current literature categorises two different approaches. The first approach is to discretise the map into convex spaces. Common examples are

Triangulation as [7], or a more advanced implementation of Hertel-Mehlhorn algorithm [8]. The second approach is to generate Points of interests, and to find strategic positioning of Guards that are in a line of sight with each point. As for example [9]. Lastly, once the points are determined, the problem switches to Vehicle Routing Problem.

2.3 Vehicle Routing Problem

The basic vehicle route problem (VRP) is to find a route that reaches different coordinates on a map. Most research efforts are concerned with finding better heuristics. The most famous techniques are the tabu search implementations of Taillard [10] and simulated annealing [11]. Though efficient, those implementations are harder to implement and the heuristics need a long fine-tuning. Another approach, by which our solution is inspired regards genetic algorithm [2].

2.4 Formation Sweep

To avoid obstacles during formation keeping, Petter Ogren and Naomi Ehrich Leonard combined a classic formation scheme with Dynamic Window Approach [12]. Dynamic-Window Approach was first introduced by Fox et al. to prevent robots from collision [13]. It approaches the search for robot control variables directly in the space of velocities. In our project, we took a simpler approach and only checked the obstacles within a window in the coordinate space.

2.5 Shooter Coordination

In the dynamic procedural combat tactics AI designed for the game Killzone, Straatman et al. assign values for each waypoint vertex by combining a set of tactics metrics (distance, attack range, and so on), and assign weights for the edges between vertices taking into consideration the distance and how exposed the route is to the enemies [14]. This way, the combat agent can be sent to the most valuable attacking position through the least costly route. We use Dijkstra's algorithm to find such a least costly route through the edges from one vertex to another. This algorithm finds the shortest paths between nodes in a graph, by picking the unvisited vertex with the lowest cost to the current path at each iteration [15].

3 Proposed method

In this section, we are going to explain in details the algorithms we used to solve the five problems.

3.1 Vacuum Cleaner Planning

To utilise Spanning Tree Coverage for this problem, we need to find an MST given the terrain. First, we sampled the entire continuous space with certain intervals in both coordinate dimensions and save the obstacle-free points as vertices. We divide the vertices equally into groups for different car, as shown in Fig. 1. There is an edge between every pair of adjacent collision-free vertices, whose weight is the distance between the connected vertices times cross-group penalties. With this graph, we used Prim's algorithm to find the MST in Fig. 1.

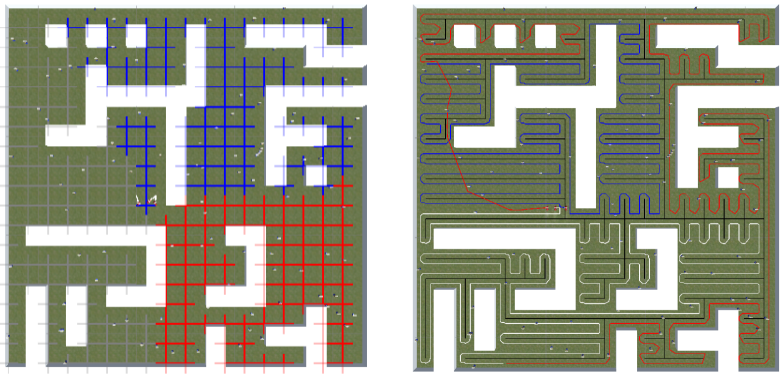


Figure 1: The figure on the left shows the obstacle-less vertices with the interval of 20f in the given UNITY terrain, where colours indicate grouping. The black lines in the right figure are edges of the MST, and the red, blue, and white lines are the paths found for three autonomous vehicles.

We then generate paths for the autonomous vehicles from the MST. When sampling the whole continuous space by 1f interval, we save the waypoints that are a certain distance away from the MST and adjust them to be in the configuration space to form a path. We divide this path into three equal continuous sub-paths, starting from the closest vertex to the vehicle starting positions. For sub-paths whose starting and ending points are both distant from the car initial position, we use RRT* [16] to connect the initial position to the closer end of the sub-path. To help the vehicles drive better, we use Chaikin algorithm [17] to smooth and populate the paths, and then use a

path straightener to remove unnecessary curves and overly close waypoints, which connects nearby waypoints directly if their distance is significantly shorter than the planned trajectory between them and there is no obstacle in between. The driving-friendly paths obtained are shown in Fig. 1.

3.2 Indoor UGV Search

Our solution is structured into two sub-problems and is inspired by the Minumum Set Cover algorithm and travelling salesman problem.

The first problem was to generate a sequence of checkpoints such by travelling there the robot cover every obstacle-free point on the map. The idea is to generate a long list of "Points of Interest" (POI), and locating as few Guards as possible to cover each POI (i.e. each POI is in line of sight with at least one Guard); then by travelling once to each Guard, the long-range robot sensors cover all the obstacle-free space. This approximation gets better the more POI we generate over the map.

The second was to find the optimal sequence to visit each checkpoint distributed to each available vehicle.

The overall solution consists of the following sequence of steps:

1. **Generate a Graph that spans the obstacle-free space**
   As we did in problem 1, first we overlapped a grid over the maze, then we constructed a graph such that each intersection is a vertex of the graph and each segment that connects two vertices becomes an edge. We removed intersections and edges that had collisions with some obstacle. As a result, we obtained a graph extended over every free area and such that each edge has the same length.

2. **Generate a list of Points of Interest (POI)**
   We considered each vertex of the Graph as a Points of Interest. Experimental tuning needs to be performed such that the graph has high enough precision to cover the whole space. In other words, the number of POI depends on the number of Vertices of the graph, to have higher precision, we need to decrease the length of each edge.

3. **Generate Guards such that every Point Of interest is covered**
   First, we raffled off an uncovered vertex of the graph and promoted it to "Guard", then we checked which POIs were in a line of sight with that "Guard". We repeatedly draw Guards until every POI was in a line of sight with at least one Guard. Lastly, we added some redundant guards: we continued to draw new guards for a parameterized number of times.

4. **Generate Min set of Guards that covers all the points of interest** As our set of guards contained more guards than necessary, we used a Minimum set cover algorithm to find the minimum (optimal) set of guards. To this end we implemented a greedy heuristic which sorted the Guards by counting how many uncovered POI it was covering, then we selected the Guard which covered them most and finally removed the covered points for the selection of the followings guards.

5. **Find a path that visits all the guards with shortest path: Vehicle Routing Problem** First, we partitioned all the guards into as many groups as the number of vehicles available. To achieve this, we generated a set of poles equidistant from the centre, then we deployed the guards to the nearest pole. We used a Euclidean distance to approximated the distance between guard and pole. However, to get a more accurate heuristic, an effective distance would be much better. Nevertheless, we preferred to use a Euclidean distance as it orders of magnitude faster.

   Secondly, we performed a random search to find the best path according to our heuristic. More specifically we shuffled the guards and evaluated the overall distance that each vehicle needs to run. Though naive, it yielded good results. Arguably the core strengths of this solution are that each component of the algorithm was as lightweight as possible in terms of CPU complexity, therefore we could have reached very deep searches in the configuration space in a matter of 1-2 seconds.

   Lastly, we run Breadth-first search (BFS) to find the path on the graph connecting each checkpoint.

The paths generated by this algorithm are shown in Fig. 2.

3.3 Vehicle Routing Problem

This is the same problem as the last step for P2 in 3.2. The only difference is that we already have the Checkpoints that the cars need to drive to. Therefore, to find the best path connecting each of the guards and distributing it to the different vehicle we used the same solution.

1. Cluster the Goals to as many equidistant poles according to a euclidean distance heuristic.

2. Use a Random search to find the best permutation of the checkpoint list. Below are given further details about the evaluation of a permutation:

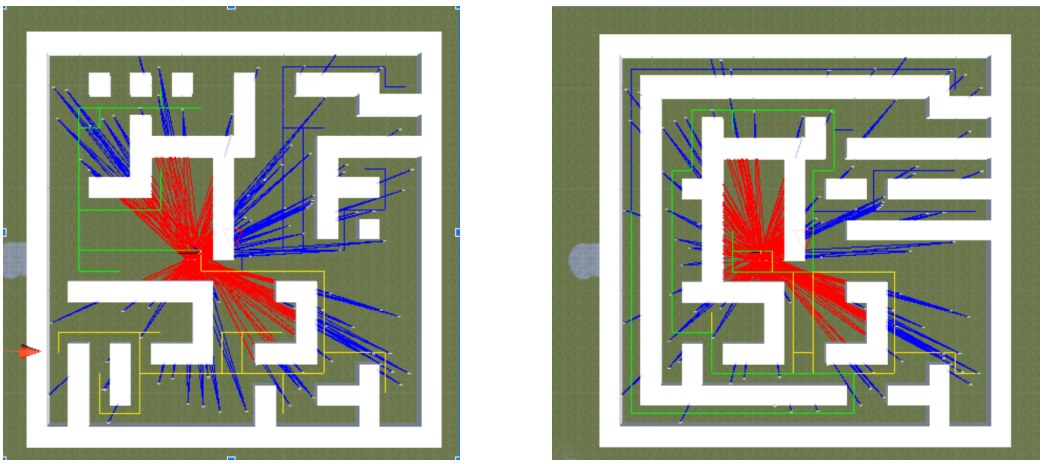


Figure 2: Paths for UGV Search Problem, different colours for different vehicles. Left figure is for training map and the right one is for test map.

- We divide the guards into as many sets as the number of vehicles. For each set, we compute the distance from the start position of the car to the first element of the set, then we add the element from the first element of the set to the next.
- Then we compute the average of the distances of each set. In this way, we prefer a permutation that aims to have an equal path length for each car.

The paths found for this problem are in Fig. 3.

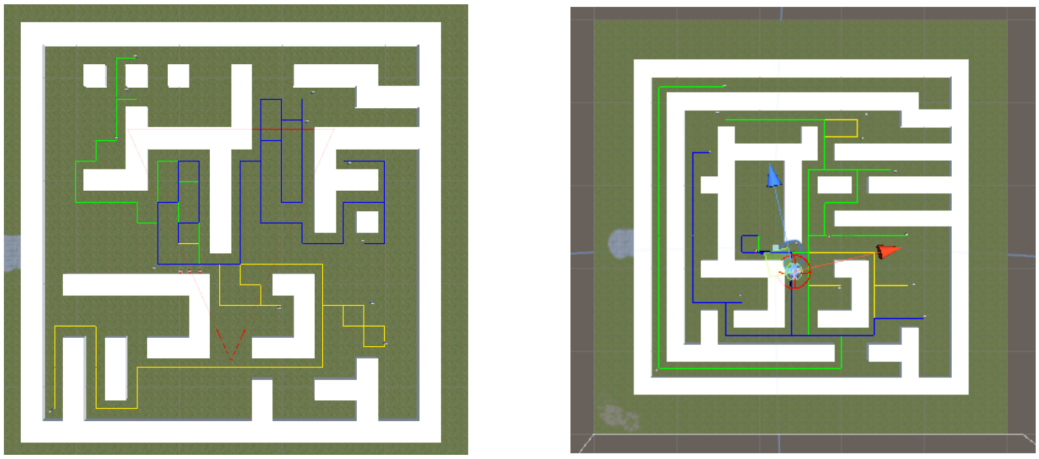


Figure 3: Paths for VRP problem, different colours for different vehicles. Left figure is for training map and the right one is for test map.

3.4 Formation Sweep

We use symmetric V-shape formation for the five cars given in the problem. Given the leader car's position $\mathbf{p_0}$ and velocity $\mathbf{v_0}$, we detect obstacles in a rectangle window whose sides are in the same direction or perpendicular to the car's velocity. This window is 20 times the car length long, and we increase its width from 2f until it reaches any obstacles. And then, we distribute the vehicles evenly along the width and into rows of two cars, with each row of cars one car length distance from the previous row/leader car, in the opposite direction of the leader car's velocity. The formation is shown in black lines in Fig. 4. After computing the planned positions for the follower cars, we generate all combinations of cars to the positions, and pick the combination that allows the shortest total distance from the cars to their assigned positions. The assignment of positions can be seen in Fig. 4 as the cyan lines.

When driving the vehicles, we let the cars keep the leader's speed, but slow down when they are close to the planned positions, and use full brake when they are over them.

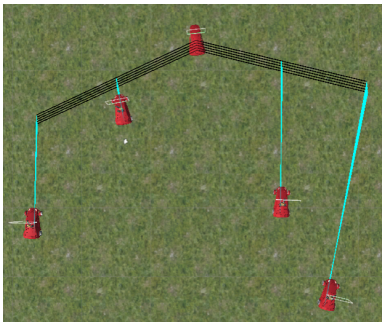


Figure 4: V-shape formation sweep

3.5 Shooter Coordination

The code idea for this strategic shooter combination problem is to attack in numerical advantage, shoot from most advantaged points, and move along the least costly route.

1. Build a graph that spanned the obstacle-free space as in the previous problems.

2. Cluster the allied units in a formation. To find the ideal position for the cluster to attack from, we evaluated each vertex according to a

heuristic that penalised exposure to more than one enemy, the lack of enemy in firing range, and the distance from the cluster's current position to the vertex.

3. Move the cluster from its start position to its desired attack position using Dijkstra's Algorithm, which finds the least costly path given weighted edges. The weights reflect the edge's length and how exposed to enemies it is.

4 Experimental results

In this section, we will comment on the best results for each problem, and discuss how our methods can be improved.

4.1 Experimental setup

The five different experiments was simulated into different UNITY 3D Scenes. For all problem except for Formation Sweep, we have three cars, whereas in Formation Sweep we have four cars to follow one leader car. The positions of turrets are randomised for all the problems.

4.2 Analysis of Outcome

The top three results of the test terrains for problem 1 (Vacuum Cleaner Problem), 2 (Indoor UGV Search), 3 (Vehicle Routing Problem), and 5 (Shooter Coordination) are in Table 1, along with our results.

Problem	P1 (sec)	P2 (sec)	P3 (sec)	P5 (health)
Top 1	290 (G10)	136.55 (G11)	143 (G1)	161 (G3)
Top2	337 (G11)	245.39 (G3)	154.23 (G10)	147 (G12)
Top 3	360 (G1)	248.05 (G10)	175 (G9)	85 (G1)
Ours	337	136.55	244.07	0

Table 1: Results of the test terrains for Problem 1, 2, 3, and 5

For Vacuum Cleaner Problem, Group 10 had the best results, and we're following them in the second place. They took a different approach from us, by greedily visiting all vertices. In comparison, even though we have the best results among the teams using Spanning Tree Coverage, we double traversed all the edges in the MST, which results in unnecessary paths. For improvement, we can take inspiration from Group 10 and try to do the

following. Grid sample the whole terrain by intervals of $2r_{firing}/\sqrt{2}$ and take obstacle-free points as vertices, and then visit all vertices at least once with revisit allowed. This way, all the terrain would be covered and the path should be the shortest. We think this might be achieved by bipartite maximum matching, which can be solved in polynomial time.

For Problem 2, Group 11 had the best results. The solution was built around the Minimum Set Cover Algorithm. Group 10 was the second-best, and solved it using by dividing the space into a set of convex areas. We think the reason why our implementation with Guards and Points of Interest was performing better is that it was easier to find an optimal and smaller set of way-points to reach. For Problem 3 Group 1 had the best results. The solution was built around MST with 3 sub-branches, then using A* for actual traversable driving cost. In comparison, our solution for problem 3 was too simple, we could then find ways to improve our search. Moreover, the clustering for problem 3 was doing more harm than good due to the chaotic nature of the test map.

In the Formation Sweep problem, the results are similar among groups. Our method was able to keep a decent formation without cars colliding with each other. However, the sweeping coverage could be improved by using an asymmetric formation.

We didn't manage to get results in the Shooter Coordination problem in time, which we'll analyse more in section 5.1.

5 Summary and Conclusions

In this project, we solved five problems for multi-agent systems in the UNITY environment. We obtained great results in the first three problems with Spanning Tree Coverage, DFS, BFS, and Minimum Set Cover algorithms, and did decently in Formation Sweep, but didn't manage to finish the Shooter Coordination problem.

For further improvement, we can solve the shortest path cover problem for an acyclic graph in the Vacuum Clear Planning problem to get a more optimal path, possibly by bipartite maximum matching. For UGV Search and Vehicle Routing problems, more tuning of the parameters on more terrains would further improve and generalise the solutions. An asymmetric formation would be better for Formation Sweep.

5.1 Reflection on failure and risk analysis

First of all, we didn't have good enough time management and spent too much time on the other problems. We only had four days for the Shooter Coordination problem and used the first three to structure the algorithm framework and implement the first approach.

Specifically, we were stuck at assigning the weights to each edge when the deadline was past. In our first approach, this step is too computationally heavy because all waypoints on the edges were inspected for exposure to every single enemy. Given more time (or better time management), we could test only a few points on each edge and only calculate the weights for edges close to the current car positions.

References

[1] Martin Saska, Tomas Krajnik, and Libor Pfeucil. Cooperative μuav-ugv autonomous indoor surveillance. In *International Multi-Conference on Systems, Signals & Devices*, pages 1–6. IEEE, 2012.

[2] Barrie M Baker and MA Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800, 2003.

[3] V Jarnık. About a certain minimal problem. *Práce Moravské Prırodovedecké Spolecnosti*, 6:57–63, 1930.

[4] Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

[5] Yoav Gabriely and Elon Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of mathematics and artificial intelligence*, 31(1):77–98, 2001.

[6] Abderrahmane Lakas, Boumediene Belkhouche, Omar Benkraouda, Amin Shuaib, and Hussain Jaffar Alasmawi. A framework for a cooperative uav-ugv system for path discovery and planning. In *2018 International Conference on Innovations in Information Technology (IIT)*, pages 42–46. IEEE, 2018.

[7] Richard I Hartley and Peter Sturm. Triangulation. *Computer vision and image understanding*, 68(2):146–157, 1997.

[8] D Hunter Hale, G Michael Youngblood, and Priyesh N Dixit. Automatically-generated convex region decomposition for real-time spatial agent navigation in virtual worlds. *AIIDE*, 8:173–8, 2008.

[9] Yang Yu, Xin Yao, and Zhi-Hua Zhou. On the approximation ability of evolutionary optimization with application to minimum set cover. *Artificial Intelligence*, 180:20–33, 2012.

[10] Éric Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673, 1993.

[11] Ibrahim Hassan Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of operations research*, 41(4):421–451, 1993.

[12] Peter Ogren and Naomi Ehrich Leonard. Obstacle avoidance in formation. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 2, pages 2492–2497. IEEE, 2003.

[13] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[14] Remco Straatman and Arjen Beij. Killzone's ai: dynamic procedural combat tactics. In *Game Developers Conference*, 2005.

[15] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[16] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[17] George Merrill Chaikin. An algorithm for high-speed curve generation. *Computer Graphics and Image Processing*, 3(4):346–349, 1974.