Assignment 2: Cooperative Multi Agent AI

GROUP A2:01

Taschin Federico Nimara Doumitrou Daniil

June 14 1996 September 8 1996

taschin@kth.se nimara@kth.se

April 15, 2021

Abstract

Intricate tasks set in complex environments often benefit greatly from the cooperation of multiple agents. In such a setting, each agent can act independently from the rest, but chooses to exchange information and team up in order to optimize his performance on the given task. In this report, we will focus on five characteristic problems set in cooperative environments, which span numerous applications. We will examine Short and Long Range Multi Agent Search, the Vehicle Routing Problem, Formation Sweep and Shooter Coordination. The methods described in this paper can be utilized to plan automatic patrols, ensure fast, robust area scanning and coordinate an intricate ordering-delivery system.

Keywords: Multi-Agent Cooperative AI, Genetic Algorithm, Vehicle Routing Problem.

1 Introduction

Introduction describing the problem and importance in more detail

Multi Agent Artificial Intelligence examines tasks set in environments, where multiple agents mutually interact in order to achieve a certain goal, such as reaching a certain location or defeating their opponents. Usually, the performance of an agent on a problem can be quantified via a utility function. For instance, such a function can refer to the time it took to reach a certain location, or how much damage it withstood in a hostile environment before taking down his opponents. If the agents' utility functions conflict with one another, then we explore the field of **Competitive Multi Agent AI** (e.g. zero sum games). Often times, however, the utility functions are compatible and benefit greatly from cooperation. In this report, we will focus on the latter, examining **Cooperative Multi Agent AI**. Since this field encompasses numerous tasks and applications, we will narrow our focus on five essential problems: Short and Long Range Multi Agent Search, the Vehicle Routing Problem, Formation Sweep and Shooter Coordination.

In **Short Range Multi Agent Search**, our agents are tasked with effectively scanning an area with their low proximity sensors. When tackling this problem, one must create a path that takes into account his agent's scanning proximity in order to ensure that the whole environment is effectively traversed and scanned. For efficiency, however, the agents must attempt to coordinate their movement to minimise completion time. Intuitively, this can be achieved by minimising their paths' overlap and spreading the task evenly among them (similar path lengths).

For **Long Range Multi Agent Search**, our agents' scanning range is beyond their field of view, meaning they are able to perceive everything they see. In this scenario, we wish our agents to efficiently patrol an environment in order to ensure that they have seen everything within it. Once more, the difficulty of this task lies in their coordination, where one needs to reduce vision overlap and evenly spread the task among them.

Vehicle Routing Problem (VRP) constitutes a generalisation of the Travelling Salesman Problem (TSP), which is a difficult NP-Complete Problem. In TSP, one is provided with a list of locations (cities) and distances between each aforementioned pair of positions and is tasked with finding the shortest possible cycle which traverses each location. More formally, given K_n (complete graph of size n), one wishes to find the lowest cost Hamilton Cycle of K_n. VRP is the multi-agent variant.

In **Formation Sweeping**, our agents are tasked with traversing an environment in formation. This formation should be dynamic, ideally spreading out in open spaces and sticking near each other in closed spaces-hallways.

Finally, for **Shooter Coordination**, our agents need to mutually communicate, coordinating their movements to minimise exposure and maximise shooting damage. In such a setting, agents select a target and find a route which maximises cover (minimising exposure and damage taken along the way). Their attacking formation is pivotal, as they should aim to minimise the damage taken and attempt to keep as many agents alive for increased firepower. It can thus be seen as an extension of formation sweeping.

1.1 Contribution

In this report, we propose optimization-based techniques to solve the multi-agent coordination tasks described above. We implement a Spanning Tree algorithm for multi agent short range search and a Dominating Set solution to a variant of the Art Gallery Problem for multi agent long range search. Then, we propose a Multi Agent Spanning Tree Genetic Optimization and a more general Multi-Path Genetic Optimization algorithm to improve the solutions. We then propose a simple but effective technique for the Formation Sweeping task based on a Leader Following approach with trails, on top of which we build a solution for the Shooter Coordination problem, based on A* optimization of path cover. Overall, our report constitutes a robust foundational framework of important Cooperative Multi Agent AI tasks.

1.2 Outline

The remainder of this report is structured the following way: In Section 2, we will introduce and familiarise the reader with related work in this field so that he will be able to better conceptualise our approach and reasoning. We will then present the methods that were utilised to obtain our solutions in more detail in Section 3. After obtaining a firm understanding of our approach, in Section 4, we will illustrate our obtained results, analyse them and compare them with those obtained by other groups. Finally, in Section 5, we will conclude this paper by reflecting on our findings and summarising the key takeaways of this report.

2 Related work

A common idea to solve Short Range Search, when a map of the environment is available, is to use one or more Spanning Trees to cover the whole map. In [1] one tree per agent is iteratively expanded away from the other agents, resulting in trees grown in opposite directions (branching out). After **branching out**, **hilling** is performed, a process that reshapes said branches in order to more efficiently cover the desired space (reduces the number of long branches). If no branching out or hilling is possible, the trees expand homogeneously, until all space is covered. Lastly, the separate trees are adequately linked and the agents then traverse *around* the spanning trees.

The long range search problem consists in finding and cleverly traversing the optimal set of spots that allow seeing the whole map. This is a variant of the Art Gallery Problem, that asks to find the minimum number of points that cover the whole map. The problem is known to be NP-hard [5], but good non-optimal solutions can be found in a short amount of time (heuristic based algorithms). After finding the desirable locations, the agents must efficiently explore them, which boils down to a VRP task. Long range search is essentially a patrolling problem, which hosts rich bibliography and numerous different approaches [8]. For instance, one can attempt using Reinforcement Learning (Gray-Box Learner Agent [9]), where agents plan their path by picking from a finite sequence of actions in order to minimise

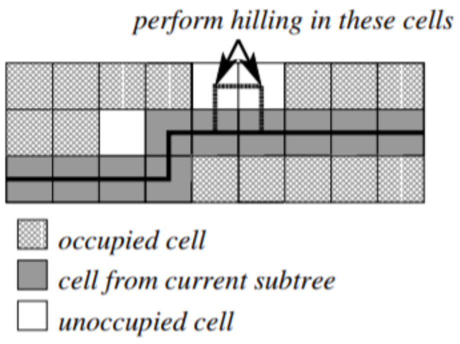


Figure 1: Hilling procedure Illustrated [1]

the nodes' idleness (time between visits). Cooperative Auction Systems propose an interesting alternative, where each agent auctions his arbitrarily given set of vertices in order to exchange-negotiate them for nodes which will result in more efficient patrolling [6] (minimise revisits and idleness). Finally, TSP cyclic approaches base their paths on solving TSP on the patrolling waypoints [2].

The Vehicle Routing Problem is a well known and documented NP-Complete Problem. A common approach is to first solve the single Agent Variant (TSP) heuristically and then split the path (reasonably) into n separate sub-paths (n: number of agents). One then uses this as an initial value for an optimisation algorithm such as Tabu Search [11], Simulated annealing or Genetic Algorithm, such as [3], where genetic operators -such as mutation and crossover- are iteratively applied.

Formation sweep techniques generally fall into three broad categories: **Leader-Follower**, **Virtual Structure** and **Decentralised Local Interaction**. In Leader-Follower, one agent acts as leader, independently driving the formation forward towards a goal, while the other agents try to maintain relative distance and orientation with their leader. For example, [7] adopt such an approach for car-like vehicles, by incorporating the appropriate dynamic motion model and applying the necessary control mechanisms to ensure that all agents respect their formation imposed constraints (relative position and orientation). Virtual Structure is leaderless and agents aim to maintain their position within a virtual structure (e.g. a line or a triangle). Finally, Decentralised Local-Interaction is also leaderless, and agents generate a formation based on local interactions with their neighbours (maintain certain distances). Intuitively, the first approach utilises a virtual structure which is centred around a 'leader' agent, unlike in the second approach, where it is independent of the agents' positions. The final one discards this notion, and hopes to generate a formation based on local agent interaction.

Lastly, Shooter Coordinations aims to navigate and coordinate a group of agents intelligently, in order to maximise firepower and minimise received damage. In [10], agents navigate a complex environment, by discretizing it, scoring each discrete point based on multiple parameters such as distance from target and exposure to enemy fire, and pathfind towards their desired destination via an A* pathfinder which incorporates the aforementioned parameters to ensure fast and safe traversal from point A to point B.

3 Proposed method

3.1 Short Range Multi Agent Search

Our first task is to effectively scan an area utilising our agents' low proximity sensors. Our approach can be split up in two halves: We first generate N spanning trees, basing our approach on [1], and afterwards optimize our spanning trees via an Evolutionary Algorithm.

3.1.1 Spanning Tree Generation

Assuming we have N agents with a scanning radius of r, our Spanning Tree Generation consists of the following steps:

1. Discretize the Environment into $D \times D$ blocks in such a way that the unit square of the grid is located in its entirety within an agent's radius r when the agent is located in its centre.

2. Generate N spanning Trees:

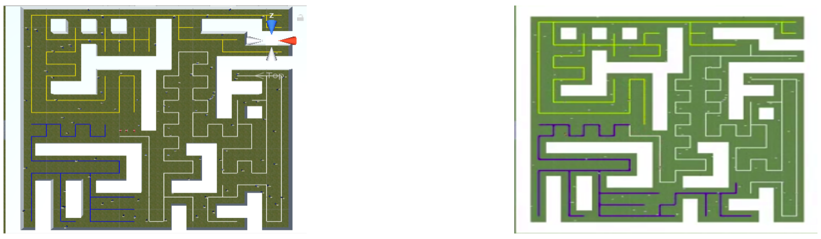
   (a) Branch out in n different directions. In this stage, each agent expands his tree so that he maximises the minimum L_1 distance from the other agents (furthest away from your nearest neighbour).

   (b) If unable to branch out any longer, perform hilling (see Figure 1).

   (c) If unable to perform neither of the above, uniformly expand your branch in all directions.

   (d) Repeat steps (a)-(c) until the whole discretized environment belongs to a tree (fully covered).

After obtaining the spanning trees, our agents perform their scanning by **driving on them** (and not around them as proposed in [1]). After performing these steps, one can expect the following result:



(a) Spanning Trees before Evolutionary Algorithm (b) Spanning Trees after Evolutionary Algorithm

Figure 2: Left (a): Resulting Spanning Trees before any Evolutionary Algorithm Based optimisation. Notice the zig-zags generated by hilling as well as the imbalanced size of our trees. Right (b): Spanning Trees after applying our Evolutionary Algorithm. Notably, the trees are significantly rebalanced, leading to increased performance.

3.1.2 Evolutionary Algorithm

The generated spanning trees are not yet optimal, since their length depends on the shape of the map portion they are pushed into. As we aim to obtain spanning trees of approximately the same size, we resort to a modified version of the genetic algorithm of [3]. The genetic algorithm works on a population of solutions. A solution is a collection $\{P_i\}_{i=1}^{N}$ where N is the number of agents and P_i is the spanning tree assigned to the i-th agent. The algorithm iteratively performs the following steps in order to optimize the population.

1. **Offspring generation**: From the current population, new solutions are generated in two ways:
   (a) **Mutation**: Randomly modify a solution to obtain another valid solution.
   (b) **Crossover**: From two solutions generate a possibly better solution.
2. **Fitness score**: Assign a score to each solution based on "how good" the solution is.
3. **Selection**: Keep only a proportion p of solutions with the highest score, and go back to 1.

This is the general framework of Genetic Optimization. We adapt the Mutation and Crossover steps to the constraints of our particular problem.

Mutation Due to the discretization of the map into a grid, we want subsequent nodes of our trees to lie in adjacent squares, and the union of the trees must cover the whole free space in the grid. Therefore we must ensure that, after performing a mutation, the resulting solution A) still covers the whole grid B) any children node is in the 4-neighborhood of the parent, and C) the resulting tree does not contain cycles. Algorithm 1 shows the Mutate(P_k) operation that preserves these 3 properties.

```
Data: $P_k$, $k$-th spanning tree of a solution S
Result: Mutated spanning tree
$i$ = random node of $P_k$;
$N_i$ = 4-neighborhood of $i$;
$j$ = node in $N_i$ that belongs to $P_k$;
$P(j)$ = parent node of j;
$P(j).children.remove(j)$;
$i.children.add(j)$;
```

Algorithm 1: Mutate(P_k) procedure

Crossover It is common in genetic optimization techniques to perform crossover between two solutions. In this context, however, each solution contains N paths, and a crossover operation would mean ”destroying” the solution's paths in small pieces and recombine them to satisfy the properties above, which becomes hard from both the algoritmic and computational point of view. Instead, we perform crossover between paths inside a single solution. This is much easier because paths inside the same solution already satisfy the properties above, and we only need to assign a subpath starting from a node of path P_k as children of a neighbour node of path P_j. After this operation path P_j effectively inherited a portion of path P_k by maintaining the validity of the solution. The algorithm is therefore the same Algorithm 1 with the only difference that node j comes from another path in the solution instead.

Fitness score We define the *fitness* of a solution as

$$fitness = -\max_{p \in S.paths} score(p) \tag{1}$$

The *score* function is defined as

$$score(path) = length(path) + \alpha n_{leaves} \tag{2}$$

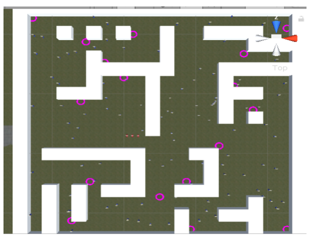
where $length(path)$ is the total distance that the agent would travel by following the path, and n_{leaves} is the number of leaves of the spanning trees in the solution. We penalize the number of leaves since a leaf node requires the agent to make a 180 degrees turn which takes more time.

3.2 Long range multi agent search and Vehicle Routing Problem (VRP)

In Long range multi agent search our agents are tasked with efficiently patrolling an area. Subsequently, this problem can be partitioned into two key segments. First, we ought to pinpoint key locations that our agents need to reach in order to see the whole map. To this end, we will compute and extract the Visibility Graph of our map and heuristically find a Dominating Set. This first section can be viewed as the NP-Complete Art Gallery Problem (see following subsection). After obtaining the aforementioned Dominating Set, we simply need to find and efficient way of visiting its nodes, by solving the corresponding VRP.

3.2.1 Art Gallery Problem

As stated earlier, we wish to find key locations that ensure full visual coverage of our map. To model map vision, we use the visibility graph, which consists of Vertices for each corner in the **Configuration Space** (space showcasing the possible states of the agent, when taking into account its geometric features) and edges between mutually visible vertices (can traverse in a straight line without colliding on any obstacle). Afterwards we heuristically extract its Dominating Set via the greedy algorithm described in [4]. However, simply seeing all of the corners of the map might not suffice as shown in the figure below (see figure 3b). To this end, we augment the visibility graph, by adding extra vertices in the midpoint of each pair v_i, v_j, when their edge (v_i, v_j) is horizontal or vertical. Midpoints of vertical or horizontal (v_i, v_j) represent points along walls. By considering them, we make sure to not only see all the corners, but also potential 'blind' spots that hide behind walls.


(a) Dominating set

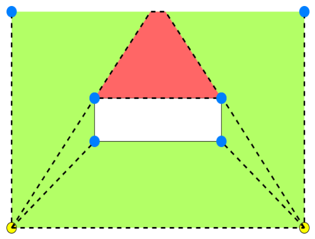

(b) Argument for Augmented Visibility Graph

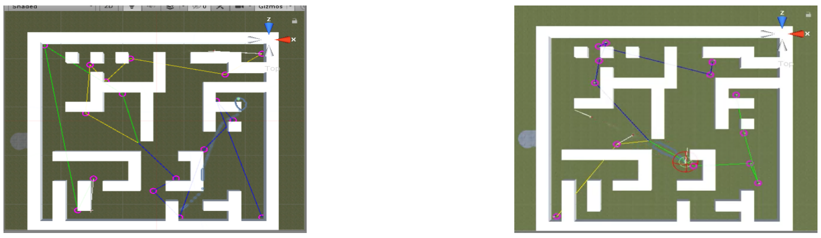
Figure 3: Left (a): Dominating set of our Map. Dominating Set points are illustrated via purple circles. Right (b): Argument for utilising an Augmented Visibility Graph: blue points are visibility corners, yellow points are visibility corners in the dominating set, the red area is the area not visible from the dominating set points.

In the Art Gallery problem, we wish to find a Vertex Cover: A set of vertices (dominating set S) S such that $\forall\ v \in V, v \in S \cup N(S)$, where $N(S) = \cup_{v \in S} N(v)$ (neighbourhood of S). The dominating set was computed greedily:

1. Colour nodes as black (covered and in S), white (uncovered) and gray (not in S but covered by their neighbours). In the beginning $S = \emptyset$.
2. While $\exists$ white nodes: choose $v \in \{x | w(x) = max_{u \in V} w(u)\}$, $w(x)$ is the number of white neighbours of x. Finally, $S = S \cup \{v\}$.

3.2.2 VRP using Genetic Algorithm

We optimize the paths with the genetic algorithm of 3.1.2 without penalization on the leaves and without the requirement of nodes lying on the 4-neighborhood of their parent nodes. Therefore, both in mutation and crossover, a node can be connected to any other node in any order.



(a) Long Range Search solution (b) VRP solution

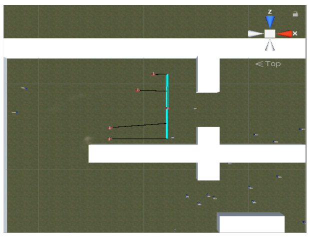
Figure 4: Left (a): Solution for Long Range Multi Agent Search. Lines indicate the paths of each agent (point sequence). Agents go from point(i) to point(i+1), following the underlying visibility graph (Shortest paths are precomputed via Floyd-Warshal algorithm) Right (b): VRP Solution.

For the Long Range Search problem, our agents actively check whether they have seen all of the neighbours $N(v)$ of their next target location v. If they have seen all of its neighbours, then they start moving towards $next(v)$ along the path. The reasoning for this is that they do not benefit from approaching v any longer, since they have already seen all that v sees.

3.3 Formation Sweep

In order to tackle the Formation Sweep problem, we adopted a Leader-Follower approach. The leader moves independently from the rest of the pack, following a predetermined path. The other agents try to follow the leader, while maintaining a linear formation at a safe distance (regulated via a tunable parameter). Each agent's target destination gets frequently updated and stored. Each vehicle tries to reach its most recent visible-reachable (obstacle free) stored destination.

Our car's velocity controls follow a sigmoidal function, accelerating when far away and decelerating when near their desired position. If an agent overshoots (e.g. due to a sharp leader turn) and ends up ahead of the line formation, then he handbrakes and tries to align himself to have the same orientation with the leader. The line formation is dynamical, meaning that it shrinks in tight spaces and expands in open ones.


(a) Formation in open space

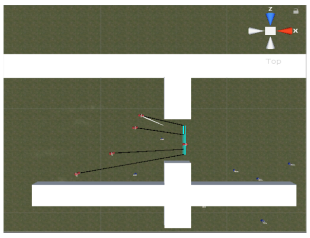

(b) Formation in closed space

Figure 5: Left (a): Linear formation in open space. Black lines point towards their intended positions on the lines. An offset is used to measure a 'safety' distance (in case of abrupt turns). Right (b): Shrinked Linear formation in closed space. Top agent's white ray points towards his most recent obstacle free destination.

3.4 Shooter coordination

In the **Shooter Coordination** task, a team of agents has to traverse the map destroying enemy agents, a common task in many videogames. Both agents and enemies shoot as soon as they are in line of sight. The goal is therefore to plan a maximum cover path that allow the agents to face the minimum number of enemies at a time, and a formation control that makes all agents face the enemy at the same time, therefore maximizing the *dps* (damage per second) inflicted to the enemy.

Formation Agents need to get in line of sight with the target all together at the same time. We therefore chose a tight line formation, with zero offset so that agents are clumped close together in a straight line. One of the agents acts as leader and the others follow their positions with respect to it. If the distance between a car and the leader is higher than the threshold, the leader stops to ensure that the formation will come across an enemy in one piece.

Maximum cover path We discretize the grid and, for each square, compute how many enemies are visible from that point. Every square in the grid has therefore a cost proportional to the number of enemies visible from that point. The problem can be solved with a optimal-path algorithm such as A*. A naive solution would be to find an enemy such that the minimum cost path from the current position to its position is the minimum among all alive enemies. Since we can shoot from distance, however, we do not need to go all the way to the enemy position, but rather to the best position from which the enemy is visible. We therefore modify the well-known A* star algorithm such that every position from which the goal is visible is itself considered the goal.

4 Experimental results

As mentioned above, we wish to solve five core problems present in Cooperative Multi Agent AI: Short and Long Range Multi Agent Search, the Vehicle Routing

Problem, Formation Sweep and Shooter Coordination. All participating groups were tasked with solving the aforementioned tasks using car agents. Each group was able to experiment with its solution on five given mazes (one for each task) and later on five other that constituted the test maps.

4.1 Experimental setup

All simulations were conducted in Unity3D. In all problems, the execution time does not consider any initial overhead due to, for example, path planning, which we report separately.

Genetic Spanning Tree Results We perform $i = 100$ offspring generations and selections with a population size of 50 solutions and a bottleneck selection where a proportion $p = 0.2$ of best solutions is kept. The population is initialized with the result of the spanning tree expansion algorithm. Mutation and crossover are performed at each iteration with a probability $p_{mutation} = 0.8$ and $p_{crossover} = 0.8$ until the necessary number of offsprings is created. The result can be seen in Figure 2b. The computation took 10.2 seconds and the multi-agent execution time was reduced by 22%.

Genetic Travelling Salesman Results We perform $i = 4000$ iterations of the genetic algorithm with a population size $N = 50$. The initialization, unlike the spanning tree version, is completely random. Although the iterations are one order of magnitude higher than those in the spanning tree case, each iteration is much faster due to the relaxed constraints of this problem instance that allow a more efficient representation of the paths as arrays of integers instead of a more complex spanning tree. This results in a computation time that never exceeds 2s.

4.2 Analysis of Outcome

The outcome of each group is summarised in Table 1.

Overall, the top performing groups were Group 1 (ours), Group 2 and Group 12. Interestingly, the approach of each group had its strengths and weaknesses as no single group achieved a relative result of 1.00 in every task. We will briefly discuss some of the more notable results.

Our group was capable of performing relatively well on the first three problems in huge part due to our Evolutionary Algorithm. For reference, in P1, the genetic algorithm cut down the time in **half** (from roughly 500 seconds to 247). Our approach in problem 5 was also relatively robust, likely due to the compact line formation our agents followed, allowing them to fire (appear in the enemies line of sight) roughly simultaneously. Furthermore, our simulations showed that our A* pathfinding was particularly effective in singling out targets, as our agents never fought, when possible, more than 1 target at the time.

Group 2's approach on the first three tasks were similar to ours, as they attempted to improve their initial solution via an optimisation technique. Unlike our

Group	Results				Relative Results				Average
	P1	P2	P3	P5	P1	P2	P3	P5	
Group 1	247.14	112.83	91.08	132	1.00	1.25	1.00	1.16	1.12
Group 2	420	90	113	79	1.70	1.00	1.24	1.94	1.24
Group 3	338.15	119.7	95.35	62	1.37	1.33 l	1.05	2.47	1.31
Group 4	409	121	109	9	1.65	1.34	1.20	5.00	1.61
Group 5	930	119	94	0	3.76	1.32	1.03	5.00	1.71
Group 6	927	130	123	45	3.75	1.44	1.35	3.40	1.75
Group 8	1173.49	285.69	196.97	0	4.75	3.17	2.16	5.00	3.04
Group 9	385.82	138.3	97.45	0	1.56	1.54	1.07	5.00	1.63
Group 10	342	123	104	81	1.38	1.37	1.14	1.89	1.32
Group 11	800	207	195.3	147	3.24	2.30	2.14	1.04	2.21
Group 12	337	130	97	153	1.36	1.44	1.06	1.00	1.24

Table 1: The best results of each group on every task on the **test maps**. Relative results are computed as $\frac{r}{r_{best}}$. Formation Sweep (P4) is not represented in this table, as as all groups successfully completed it (binary metric: done/not done). Test maps are shown as they are a more reliable indicator of the generalisation capabilities of the proposed algorithms.

approach however, they settled for Tabu search. Performance differences are likely attributed to parameter fine tuning as well as agent control, as they seem to exhibit worse relative results on VRP (task 3), which is a simpler task (it is a subproblem) than Long Range Search (task 2). Notice how in general, all other groups perform better (compared to their respective P2) on the simpler P3.

Group 12 utilized simulated annealing to tackle the first three tasks. Interestingly, they completely skipped the initial tree generation for the short range multi agent search and rather immediately started optimising a random selection of trees. Initial values-conditions greatly affect local-optimisation schemes like simulated annealing and this is possibly the reason why our approach outperformed theirs (our tree generation generated a better initial value than randomly picking three trees).

Overall, most teams tackled problems 1-3 in a similar fashion, by first finding some initial 'lacking' solution which they then optimized. Furthermore, in task 1, the majority drove **on the spanning trees**, rather than around (group 13 stands out as an exception). Formation sweep exhibited the most variety, with some teams, like Group 13, opting out of the Leader-Follower Approach and adopting a more dynamical approach, where agents are driven by interior repulsive and attractive forces in order to simulate a formation (Reynolds boids). Shooter coordination was approached, generally, in two different ways. One approach utilized formation movement (e.g. line formation) to move the agents as a unit in order coordinately strike their foes. The other line of thought attempted to move the agents independently (ignoring formation) into **key positions**, before simultaneously lunging at their enemies. It is difficult to critically access the performance on the last task by simply checking the best results, as it exhibited high variance. This variance can be attributed to the fact that keeping multiple agents alive, regardless of their hp, give exponential gain (increased firepower). However, many simulations indicated that agents may break formation, due to control complications, leading to premature agent destruction, heavily hindering performance (adding variance).

5 Conclusion

In this report, we build on the existing literature on path planning, patrolling, and formation keeping, to solve a wide range of multi agent coordination tasks. For short-range multi-agent search, we implemented a greedy spanning tree expansion algorithm and propose a technique to further optimize it with genetic operators. While the expansion algorithm already provides good solutions, the genetic optimization further reduces the total cost in a significant manner. In long-range multi-agent search, we use a greedy approach to find a dominating set on an *augmented* visibility graph, and we perform genetic optimization on the resulting Travelling Salesman problem. These results are computed in a relatively small amount of time and proved to be effective -second and first place in two competition instances. In formation sweeping we implement a leader-following formation where agents try to maintain their position with respect to the leader, and follow leader "traces" when they cannot directly drive towards their expected position. In shooter coordination, we propose a path planning algorithm based on a modified A* on the "danger areas" of the map. Overall, our approaches proved fruitful and can serve as a solid framework for tackling important Cooperative Multi Agent AI tasks.

References

[1] N. Agmon, N. Hazon, and G. A. Kaminka. Constructing spanning trees for efficient multi-robot coverage. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1698–1703, 2006.

[2] Y. Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Proceedings. IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2004. (IAT 2004).*, pages 302–308, 2004.

[3] Giovanni Giardini, Tamás Kalmár-Nagy, and Dane Quinn. Genetic algorithm for combinatorial path planning: The subtour problem. *Mathematical Problems in Engineering*, 09 2011.

[4] Fabian Kuhn. *Network Algorithms.* University of Freiburg, 2012.

[5] D. Lee and A. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986.

[6] Talita Menezes, Patrícia Tedesco, and Geber Ramalho. Negotiator agents for the patrolling task. In Jaime Simão Sichman, Helder Coelho, and Solange Oliveira Rezende, editors, *Advances in Artificial Intelligence -*

IBERAMIA-SBIA 2006, pages 48–57, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[7] P. Ogren and N. E. Leonard. Obstacle avoidance in formation. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 2, pages 2492–2497 vol.2, 2003.

[8] David Portugal and Rui Rocha. A survey on multi-robot patrolling algorithms. In Luis M. Camarinha-Matos, editor, *Technological Innovation for Sustainability*, pages 139–146, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[9] Hugo Santana, Geber Ramalho, Vincent Corruble, and B. Ratitch. Multi-agent patrolling with reinforcement learning. pages 1122– 1129, 02 2004.

[10] Remco Straatman and Arjen Beij. Killzone's ai: dynamic procedural combat tactics.

[11] Jiefeng Xu and James Kelly. A network flow-based tabu search heuristic for the vehicle routing problem. *Transportation Science*, 30, 09 1998.