Multi-Agent Ad Hoc Team Partitioning by Observing and Modeling Single-Agent Performance

Etkin Baris Ozgul*, Somchaya Liemhetcharat†, and Kian Hsiang Low*
*Department of Computer Science, National University of Singapore, Singapore
E-mail: {ebosgul, lowkh}@comp.nus.edu.sg
†Institute for Infocomm Research, A*STAR, Singapore
E-mail: liemhet-s@i2r.a-star.edu.sg

***Abstract*—Multi-agent research has focused on finding the optimal team for a task. Many approaches assume that the performance of the agents are known *a priori*. We are interested in ad hoc teams, where the agents' algorithms and performance are initially unknown. We focus on the task of modeling the performance of single agents through observation in training environments, and using the learned models to partition a new environment for a multi-agent team. The goal is to minimize the number of agents used, while maintaining a performance threshold of the multi-agent team. We contribute a novel model to learn the agent's performance through observations, and a partitioning algorithm that minimizes the team size. We evaluate our algorithms in simulation, and show the efficacy of our learn model and partitioning algorithm.**

I. Introduction

Theoretical work on multi-agent systems has mostly focused on the efficiency of the algorithm. Most research has used the same teams in the same environment to highlight the performance advantage of one algorithm over others. However, when it comes to applying the theoretical work on real world applications, another aspect of the application process comes in the picture. No matter what algorithm and hardware you are using on what application, you will have to assess the situation, environment and other external effects as well as the performance and cost constraints/requirements of your application in order to optimize the team that can yield optimal results. For example, in applications such as urban search and rescue (USAR) of a disaster site or patrolling an area for security, the multi-agent team has to be chosen carefully. Intuitively, using as many robots as possible will give the best performance but it will not be feasible in terms of cost and resources. Further, a team that is bigger than necessary may cause certain drawbacks such as higher communication load or difficulty in coordination.

We are interested in ad hoc teams, where the goal is to form a multi-agent team comprised of agents that have not collaborated before. In this work, agents can refer to humans, software agents, and robots. We believe that ad hoc teams are a realistic approach to solving real-world problems since there are many research institutions that focus on single-agent algorithms. For example, in a USAR scenario, different USAR personnel and robots may arrive at the scene. Since these USAR agents originate from different places, it is difficult to quantify how well they will work as a team, or even how best to allocate the USAR environment among them. We are interested in modeling the performance of single agents (e.g., each USAR personnel/robot) in training environments, in order to partition a new environment using a minimal number of agents. Each agent then acts independently in its allocated area, but the overall team performance has to meet a threshold. Our interest is not particular to USAR – we are interested in effectively partitioning an environment so that an ad hoc multi-agent team is able to perform well.

The first important issue while solving this problem is to model the performance of an agent in a given setting. The model has to be modular enough so that it can be applied on a new environment with ease, without requiring excessive training, so that when we want to solve the problem in a new environment, the solution will not be costly and applicable easily. The other important issue is the partitioning of the environment and allocating them among the agents. The main difficulty here is that a greedy approach might increase efficiency for certain agents but will lead to a lower overall performance than the team can achieve. Therefore, the environment has to be partitioned and allocated to team members according to their capabilities so that each agent yields good results while the overall team performance is maximized.

The related work in the literature, which we detail later, does not provide to solution for both team formation and partitioning the environment together. Since these two tasks closely related to each other, separately solving them will not yield the optimal result. Approaches that focus on fleet optimization and team formation typically rely on either precise information of the agents and environment, which is used to optimize the team for that particular environment and plan the actions of the agents. We are interested in ad hoc teams, where the actions of the agents are not controlled by us; instead, we allocated the environment for the agents to act independently.

In this paper, we contribute an approach of modeling the performance of single-agent algorithms, through observations of its performance in various training environments. We then use the learned models in order to partition a new environment in order to minimize the total number of agents, while ensuring a performance threshold of the multi-agent team.

The structure of the paper is as follows: Section II discusses related work and highlights the differences in our

research. Section III formally defines the problem, and gives an overview of our approach, and Section IV contributes our algorithms for modeling the single-agent algorithms, and partitioning a new environment. Section V describes our experiments in simulation and results, and we conclude in Section VI.

II. Related Work

To the best of our knowledge, the existing work in multi-agent literature does not cover both team optimization and environment partitioning. Existing work either covers only team formation without worrying about the partitioning, such as fleet optimization and team formation, while the others partition the environment among robots but do not cover how to form the multi-agent team and assume the team is fixed. Research in fleet optimization has mostly focused on transportation, e.g., [4]. In [4], the purpose is to find the optimal team formation among a variety of vehicles that would serve the customers without violating their constraints, such as their pick-up time window and travel duration. Similarly, [6] optimizes the fleet size and the contracts for ships. Ships serve the known demands while incorporating various aspects of the shipping industry such as ship maintenance expenses and taxes, which vary for different types and age of ships. Both [4] and [6] consist of optimizing the vehicle fleet and allocating the demands among the vehicles, but they assume all the pick-up time windows are known prior to the optimization, while our approach has no such assumption, only that the probability distribution of the pick-up demands (which we term as "intrusions" later) are known.

Research in team formation focuses on selecting which agents to form a team for a task. A common approach is to pre-define the capabilities of the agents as resources or services [11], and select the agents that cover the requirements of the task. However, such an approach requires knowledge of the agents' capabilities. In contrast, the Synergy Graph model learns the agents' capabilities and their synergy through observations [7], [10]. Thus, the Synergy Graph model is well-suited for ad hoc scenarios where the agent capabilities are unknown. A modified Synergy Graph for Configurable Robots has also been proposed to select relevant modules of different robots of a multi-robot team [9], and to handle failures in modules to achieve robustness [8]. However, the Synergy Graph approach assumes that the environment and task is fixed, and forms the optimal team of agents for the task. We are also interested in the ad hoc scenario, but we consider how to partition a new environment for a multi-agent team, after learning the agents' capabilities.

In addition to team formation and fleet optimization approaches, there exists various work on environment partitioning. They can be characterized in two groups: static and dynamic. [3] and [5] propose static partitioning of a polygon so that the partitioning is done once and never changes. On the other hand, [2] proposes a dynamic partitioning algorithm that allows neighboring agents to negotiate between themselves to exchange allocated locations, which allows them to adapt to changes in locations' probabilities of an intrusion occurring. Although these approaches provide solutions for partitioning problems, they require the number of agents to be known prior the partitioning and they are unable to find an optimal number of agents to complete the task given the constraints. In this paper, we are proposing an approach that can model a performance of an agent in an environment, which is later used to form an optimal team of robots to work in an environment as well as partitioning that environment among team members.

III. Problem Statement and Approach

In this section, we formally define the ad hoc team partitioning problem and give an overview of our approach. We begin with a motivating scenario that we use throughout this paper.

A. Motivating Example

Suppose that there is an environment where a multi-agent team has to patrol. Each location in the environment has an independent probability of intrusions occurring, and the goal of the multi-agent team is to minimize the average detection time of the intrusions. We are interested in an *ad hoc* scenario, where the agents of the multi-agent team have not coordinated before. Further, the agents have pre-defined algorithms, e.g., single-agent patrolling algorithms that are developed by other research groups. These algorithms are not designed for tightly-coupled coordination, and thus the environment has to be partitioned — each agent has an allocated area that is static, and every location in the environment has to be assigned to a single agent, i.e., there are no overlaps in the agents' allocations. Each agent then patrols its allocated area using its own algorithm. Given a desired performance bound of the multi-agent team, e.g., each intrusion is detected within 5 timesteps, the goal is to find the minimum number of agents and the optimum partitioning and allocation of the environment among the agents.

B. Ad Hoc Team Partitioning Problem

Let $L = \{l_{i,j}\}$ be the set of locations in the environment, where $1 \leq i \leq n, 1 \leq j \leq m$. We represent the environment as a $n \times m$ grid with no obstacles in this paper, although our approach is general and applicable to environments with obstacles.

Let $P = \{p_{i,j}\}$ be the set of probabilities, where $1 \leq i \leq n, 1 \leq j \leq m$. Each location $l_{i,j}$ is associated with a probability $p_{i,j}$, where $p_{i,j}$ is the probability that an intrusion will occur at $l_{i,j}$ every timestep.

Let $A = (a_1, \ldots, a_K)$ be the agents in the multi-agent team, and let $\mathcal{A}$ be the set of all types of agents, i.e., $\forall a_\alpha \in A, a_\alpha \in \mathcal{A}$. Each type of agent in $\mathcal{A}$ has a different algorithm and/or hardware thus each agent type performs differently given the same setting. Note that it is possible that $\exists \alpha, \beta$ such that $a_\alpha, a_\beta \in A$ and $a_\alpha = a_\beta$, which implies that a_α and a_β are the same type of agent. In other words, team A can comprise a homogeneous type of agent, or a combination of multiple types of agents.

Let $q_\alpha^{(t)}$ be the location of agent a_α at timestep t.

Let $v_{i,j}^{(t)}$ be the cumulative penalty for every location $l_{i,j}$, where:

$$v_{i,j}^{(0)} = 0$$
$$v_{i,j}^{(t)} = \begin{cases} 0 & \text{if } \exists q_\alpha^{(t)} = l_{i,j} \\ v_{i,j}^{(t-1)} + p_{i,j} & \text{otherwise} \end{cases}$$

Thus, $v_{i,j}^{(t)}$ increases for every timestep that location $l_{i,j}$ is unvisited by an agent.

Let $V(A)$ be the penalty of the multi-agent team, where:

$$V(A) = \frac{1}{T} \sum_{t=1}^{T} \sum_{i,j} v_{i,j}^{(t)}$$

where T is the maximum number of timesteps. Hence, $V(A)$ is the average expected penalty.

The goal of the ad hoc team partitioning problem is to form a multi-agent team A that minimizes K for a given threshold V_{desired} and allocates the environment among the agents optimally:

$$\min K$$
$$\text{subject to } V(A) \leq V_{\text{desired}}$$

C. Our Approach

We briefly describe our approach to solving the ad hoc team partitioning problem below:

Let E be the environment such that $E = \{L, P\}$

- For each agent type $a \in \mathcal{A}$, we model its single-agent penalty with a function $\tilde{V}$, which is an approximation of the unknown function V;
- We model $\tilde{V}(a, E) = w_{1,a}x_1 \dots w_{r,a}x_r$, where $w_{\alpha,a} \in \mathbb{R}$ is a weight, and x_α is a property of an environment E, that we elaborate in the next section;
- We learn the weights w_z from observations of a in a set of environments $E_{training} = \{E^{(1)}, E^{(2)}, ...\}$, where $E \notin E_{training}$;
- We partition the environment E using the learned weights $w_{\alpha,a}$ in order to minimize K while maintaining the threshold V_{desired}, where each partition is allocated to a single agent. The set of partitions $\mathcal{E} = \{E_1, E_2, ..., E_K\}$ obtained from E has to satisfy the following properties/conditions:
  - $\bigcup_{E_\alpha \in \mathcal{E}} E_\alpha = E$
  - $E_\alpha \cap E_\beta = \emptyset$, $\forall E_\alpha, E_\beta \in \mathcal{E}$ where $\alpha \neq \beta$
  - $\sum_{1 \leq \alpha \leq K} \tilde{V}(a_\alpha, E_\alpha) \leq V_{desired}$, where $a_\alpha \in \mathcal{A}$ and $E_\alpha \in \mathcal{E}$

IV. Algorithm

In this section, we will describe the two algorithms we contribute. The first algorithm models the performance of the single-agent algorithms, and learn the weights from observations of the algorithms in training environments. The second algorithm uses the learned models to partition the environment such that the number of partitions is minimized, while attaining the required performance of the multi-agent team.

A. Learning the Weights

The weights represents the effect of each property of the environment E on the function $\tilde{V}$. These weights are obtained through a training process. First, we create a set of sample environments $E_{training} = \{E^{(1}, E^{(2)}, ...\}$, whose dimensions are $n^{(1)} \times m^{(1)}, n^{(2)} \times m^{(2)}, \dots$ respectively. For the training set, the values of $n^{(i)}$ and $m^{i)}$ for each environment sample is randomly chosen from the intervals $[n_{min}, n_{max}]$ and $[m_{min}, m_{max}]$ respectively. For each environment $E \in E_{training}$ and for each type of agent $a \in \mathcal{A}$, $V(a, E)$ and properties of $\{x_1, ..., x_r\}$ are computed. After calculating the $V(a, E)$ values and properties, we store these values in matrix A and vector B, where each row corresponds to an environment sample. Each column in matrix A corresponds to one property in $\tilde{V}$ and B has values of V from the simulations. In order to learn the weights we solve the equation $Aw = B$ using least-squares approach. The weight we learned will be used to estimate the performance of an agent on a new environment.

Although it is possible to choose any other or additional properties, in this work we have used 8 properties of the environment. Using these properties,$\tilde{V}$ is represented as:

$$\tilde{V}(E') = w_1x_1(E') + w_2x_2(E') + w_3x_3(E') + w_4x_4(E') + w_5x_5(E') + w_6x_6(E') + w_7x_7(E') + w_8x_8(E')$$

w_z corresponds to weights and $\tilde{V}_z$ corresponds to the property functions. The properties we used in this paper can be summarized as follows:

- $x_1(E') = \sum_{l_{i,j}, l_{i',j'} \in L} [d(l_{i,j}, l_{i',j'})(p_{i,j} + p_{i',j'})]$,
  where $d(l_{i,j}, l_{i',j'})$ is the manhattan distance between locations $l_{i,j}$ and $l_{i',j'}$;
  Property x_1 captures both the cumulative probabilities in the environment and its size. The value of x_1 is especially important since it can measure when high probability locations are further from each other, which may cause the most frequently visited locations to be far from each other and increase the distance of the most frequently traversed trajectory of the agent.
- $x_2(E') = \sum_{l_{i,j}, l_{i',j'} \in L} [d(l_{i,j}, l_{i',j'})^2(p_{i,j} + p_{i',j'})]$;
  Property x_2 is similar to x_1, where the only difference in x_2 is that the distance is squared so the importance of the distance is higher for this property.
- $x_3(E') = \sum_{p_{i,j} \in P} p_{i,j}$;
  x_3 is the sum of the all locations' probabilities of incrusion occuring. This property gives us the expected number of items that can appear in environment E' in one time step.
- $x_4(E') = \sum_{l_{i,j}, l_{i',j'} \in L} d(l_{i,j}, l_{i',j'})$;

This property is the sum of all the pairwise distances in E'. It captures the time the agent needs to travel the entire environment (without considering the probabilities of each location).

- $x_5(E') = \max_{l_{i,j}, l_{i',j'} \in L} d(l_{i,j}, l_{i',j'})$;
  The maximum distance the agent needs to travel
- $x_6(E') = \min(n, m)$, where n and m are the dimensions of the environment E';
- $x_7(E') = \max(n, m)$, where n and m are the dimensions of the environment E';
  x_6 and x_7 captures the shape of the environment.
- $x_8(E') = nm$, where n and m are the dimensions of the environment E';
  This property represents the area of the environment.

Each property described above captures a different aspect of the environment and they are all required for representing the environment accurately for the given task. For a different task or application, the properties and the function $\tilde{V}$ can be modified. We will validate the importance of these properties later in Experimental Results section.

B. Partitioning the Environment

The goal of the partitioning step is to create contiguous segments of the environment which are assigned to each agent. To do so, the algorithm continuously creates new partitions by dividing current partitions into two until the overall partitioning meets the desired performance requirement. This strategy allows the algorithm to solve the task with the minimum number of robots. Algorithm 1 shows the pseudo code of our partitioning algorithm.

Initially, the algorithm starts with the whole environment E, and partitions it into two rectangle shaped partitions E_1 and E_2, so that the sum $V(a_1, V_1) + V(a_2, V_2)$ is minimized. Next, the partition with highest V value from the existing ones is chosen, and divided into two again as explained above. This process is repeated until the sum of all the V values in the partition list is below $V_{desired}$. This strategy allows us to meet the performance demand with the minimum number of robots.

Algorithm 1: AD_HOC_PARTITIONING L

```
E = {E};
/* Initially E only contains one
   partition which is the environment's
   itself                                  */
while ∑_{1≤α≤K} V(a_α, E_α) ≥ M do
    E' = FIND_HIGHEST_VALUE_SUBSET(E );
    [E_1, E_2] =MIN_PARTITION( E');
    REMOVE(E, E');
    ADD(E, E_1);
    ADD(E, E_2);
end
```

Algorithm 2: FIND_HIGHEST_VALUE_SUBSET

```
Ṽ(E_max) = −∞;
for each E' ∈ E do
    if Ṽ(E_max) < Ṽ(E') then
        E_max ← E';
    end
end
return E_max;
```

Algorithm 3: MIN_PARTITION

```
[nm] = SIZE_OF(E');
min_row_partition_value ← ∞;
min_row_partition_index ← 0;
for i from 1 to n − 1 do
    (E_1, E_2) ← DIVIDE_FROM_ROW(i, E');
    /* Divides E' into two partitions
       from row i                          */
    if Ṽ(E_1) + Ṽ(E_2) < min_row_partition_value
    then
        min_column_partition_value ←
        Ṽ(E_1) + Ṽ(E_2);
        min_column_partition_index ← i;
    end
end
min_column_partition_value ← ∞;
min_column_partition_index ← 0;
for j from 1 to m − 1 do
    (E_1, E_2) ← DIVIDE_FROM_COLUMN(j, E');
    /* Divides E' into two partitions
       from column j                       */
    if Ṽ(E_1) + Ṽ(E_2) < min_column_partition_value
    then
        min_column_partition_value ←
        Ṽ(E_1) + Ṽ(E_2);
        min_column_partition_index ← j;
    end
end
if min_row_partition_value <
min_column_partition_value then
    return DIVIDE_FROM_ROW
    (min_row_partition_index, E');
else
    return DIVIDE_FROM_COLUMN
    (min_column_partition_index, E');
end
```

V. Experiments and Results

A. Experimental Setup

We can think the experimental section as two parts. The first part is learning the weights $W = \{w_1, w_2, ..., w_8\}$. And second part is where we test the performance of our algorithms. Since each agent will patrol segments with different sizes, the environment set for the learning consists of various smaller sized rectangles which are used separately to observe a single agent's performance. Therefore, we will use environments which are $n \times m$ matrices where $2 \leq n, m \leq 10$ and each element is the probability of an intrusion occurring. The probability of an intrusion occurring is selected using Uniform Distribution $U(0, 1)$.

During the testing, we will use a larger environment for our algorithm to partition in the smaller parts. thus, the size of the environment is $n \times m$ where $n = m = 16$. Again differently from the training environment we will have two types of environment here. This will allow us to see the performance of our algorithm in an environment where probability distribution is different than the one we learned the weights. While learning the weights, we used an environment where the probabilities are generated using Uniform Distribution which we will call "Uniform Environment". In addition to the Uniform Environments, now we will also introduce the "Gradient Environment", in which a focal point of gradient is randomly selected and assigned with the maximum value 1, the corners have the minimum value 0 and every other element in the environment matrix has a value that gradually increases as they move from corners to the focal point of the gradient.

Our framework is independent from the patrolling algorithm the agents are using and can be used to partition the environment to any given patrolling algorithm. In our experiments, we will be using a modified version of continuous area sweeping algorithm proposed in [1]. In the algorithm we used in our experiments, we keep record of each grid cell's cumulative expected probability of an intrusion occurring since the last visit of an agent. This cumulative value represents the potential of that cell containing an intrusion. The value representing the potential of an intrusion is set to zero after an agent visits the cell. Before committing an action, agent computes the routes from its current location to all other grid cells. Each route consist of cells whose total potential of an intrusion is maximum en route to the destination with the minimum distance. After computing the routes, each of their potentials are divided with the time that is required to traverse that route and the one with maximum potential of an intrusion is selected as the action. This process is repeated until the end of the simulation. Our algorithm differs from the one in [1] in terms of observing the intrusions. [1] assumes that the agent can observe any location that is in its sight, while in our algorithm assumes the agent can only observe its current location.

B. Learning the Weights of the Single-Agent Algorithm

In order to learn the weights that are required to compute $\tilde{V}$, we generated a random sample set of 1000 uniform $n \times m$ environments, where n and m are chosen randomly within the range $[2, 10]$ for each environment instance.

For each uniform environment instance, we run the agent with the algorithm explained above for $T = 1000$ time steps and calculate the value V for each instance as we have explained in the Section III. Together with the V for each instance, we also collect the properties x_z for the same instance. V values for all instances are stored in vector B and values of properties x_z are stored in matrix A, where $Aw = B$, w being the weights we learn using the least-squares.

To measure the performance of the learned weights we use N-fold cross validation. N-fold cross validation is a technique that is used to measure the performance of a prediction model. In this technique, the data set is divided into N subsets. In each iteration, one subset is chosen for the test while the other $N - 1$ subsets are used for training. This process is repeated N times until each subset is used as a test subset once. In our experiments we choose N to be 10.

$\tilde{V}$, the prediction of V, for the test subset that is computed using the weights we learned differs slightly from the actual V values of those environments. In order to evaluate the performance of the learned weights, we calculate the error of our prediction by comparing it to the actual values of Vs. This comparison shows that we can predict the V with error of **1.406%** with standard deviation of **0.1064%**.

The weights we learn are shown in Table I.

TABLE I: The Weights

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
0.1517	-0.0096	5.7237	0.0013	59.7756	-31.2054	-39.4236	-6.1804

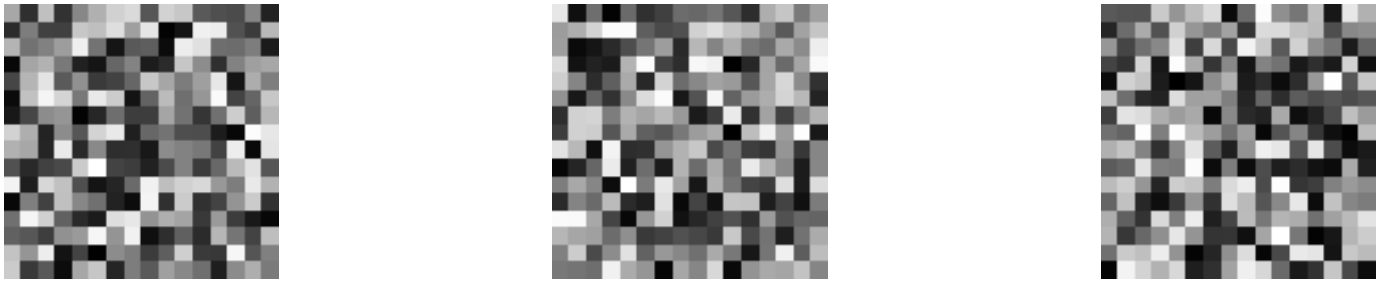
C. Partitioning the Environment

In this section, we will describe Even Partitioning and Random Partitioning as benchmarks and compare the results of our algorithm with these benchmarks.

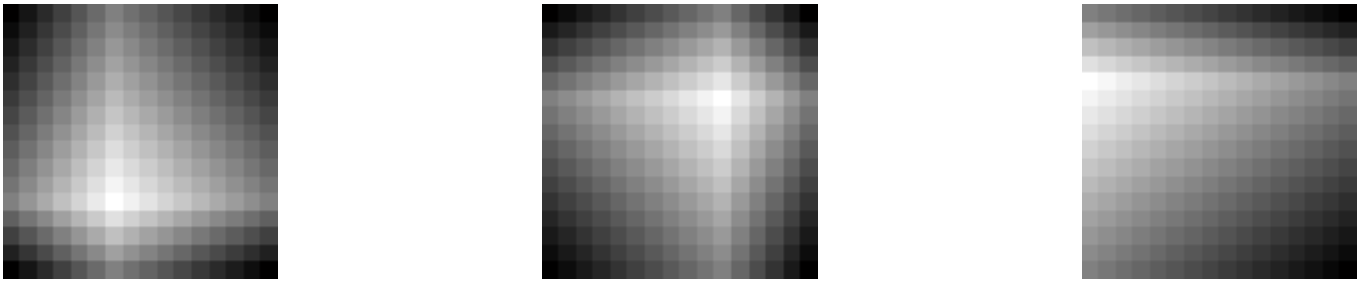
Even Partitioning divides the largest partition into two equal $n \times m$ partitions such that $\mid n - m \mid$ is minimized. This process is repeated until the environment is partitioned into K parts or a certain threshold for performance prediction is reached.

Random Partitioning divides the largest partition into two partitions, where the cut is chosen randomly. This process is repeated until the environment is partitioned into K parts or a certain threshold for performance prediction is reached.

For comparing the result of our algorithm to Even and Random Partitioning algorithms, we will use both the Uniform Environments and the Gradient Environments. For each type of environment, we will first run our algorithm using threshold $V_{desired} = 300$. Our algorithm will find the optimal number of partitions K, that will give us a penalty lower than the threshold $V_{desired}$. After obtaining K value we will run the Even and Random Partitioning algorithms with it. Using the same K value allows us to observe how different partitioning affects the outcome of the system, when they all have same number of agents.



(a) Uniform Environment



(b) Gradient Environment

TABLE II: $\tilde{V}$ values for each partitioning approach

	Ad Hoc Partitioning	Even Partitioning	Random Partitioning
Uniform Environment	267.4 ± 18.9	321.3 ± 21.5	794.6 ± 179.2
Gradient Environment	272.2 ± 17.0	361.2 ± 34.2	813.9 ± 185.7

TABLE III: Results on Uniform Environments

	Ad Hoc	Even	Random
$\tilde{V}$	262.9 ± 19.0	319.3 ± 18.2	917.7 ± 200.1
V	553.1 ± 11.4	598.3 ± 12.1	692.2 ± 23.4
Detection time per intrusion	4.3 ± 0.1	4.7 ± 0.1	5.4 ± 0.1

TABLE IV: Results on Gradient Environments

	Ad Hoc	Even	Random
$\tilde{V}$	270.8 ± 17.8	362.2 ± 29.0	820.1 ± 231.5
V	573.8 ± 11.7	629.9 ± 35.5	704.2 ± 37.9
Detection time per intrusion	4.7 ± 0.1	5.2 ± 0.3	5.8 ± 0.3

D. Simulating the Agents with the Partitions

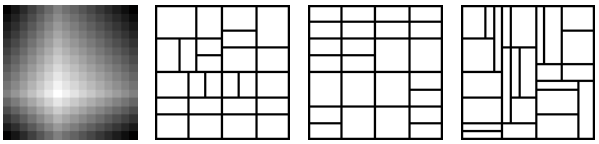
In this section, we will repeat the procedure in the previous part and again create two sets of data, Uniform Test Environments and Gradient Test Environments, and will repeat the procedure as before. We will first partition each environment according to our three partitioning settings Greedy, Even and Random Partitioning and then run a simulation where we will randomly generate intrusions according to the probability of an intrusion occurring for each location. Then we will run the agents in separate settings for the same simulation and measure each partitions' and their agents' performance. For the simulation, we will create 10 new environments for both Uniform Environments and Gradient Environments sets and run each simulation for $T = 100$ steps and calculate: predicted performance from our model $\tilde{V}$, the actual performance criteria V and a reward function that is the average detection time per intrusion. The most important performance measure is the average detection time of the intrusions, since in a patrolling task the objective is to detect events or intrusions as soon as possible. Note that for each environment we compare each setting within the same simulation, meaning the intrusions occur exactly the same time and location for each settings' simulation on the same environment.

The Tables III and IV show that the performance predicted using $\tilde{V}$ function for each setting is proportional to the outcome performance of each setting in the simulation. Although our prediction doesn't give the exact performance result, it does allow us to compare their performances, i.e., the partitioning with lower $\tilde{V}$ will also have lower V in the experiments and will perform better with shorter detection time.

The purpose of our patrolling task was to detect the intrusions as soon as they occur. In other words, we want to minimize the detection time per intrusion. As you can see from both tables above, Ad Hoc Partitioning achieved lower detection time per intrusion by detecting the intrusions in less than 5 time steps on average compared to both Even Partitioning and Random Partitioning which takes them more than 5 time steps to detect an intrusion on average, as we have predicted using our $\tilde{V}$ function.

Figure 1a is the sample environment we used to compare the partitioning approaches. Lighter grid cells correspond higher probabilities of an intrusion while the darker ones have lower probability. Figures 1b, 1c and 1d show examples of the partitions created by Ad Hoc, Even and Random

Partitioning respectively. As you can see our approach, Ad Hoc Partitioning, partitions the lighter areas, which have higher probability of an intrusion, into smaller partitions which allow the robots to detect the intrusions earlier, while Even and Random Partitioning approaches are not able to capture this property of the environment.



(a) Environment (b) Ad Hoc (c) Even (d) Random

Fig. 1: Comparison of three partitioning approaches

VI. Conclusion and Future Work

In this paper we have Ad Hoc Team Partitioning Problem, where we modeled the performance of the agents, optimize a heterogeneous agent team and partition the environment among the agents. Using environmental properties, we successfully manage to model a performance of an agent in a completely new environment, without need of further training of the model. The lack of need for further training allowed us to partition the environment without any computationally costly operation.

Our results show that the partitioning algorithm is not trivial and even in a uniformly distributed environment, a even partitioning cannot perform as well as our proposed solution.

In the future, we would like to extend the modeling to various different algorithms and form the team in a heterogeneous way. In addition, we would like to compare our results with other approaches in the literature. Finally, we plan to extend our experiments to different applications using real world robots.

References

[1] M. Ahmadi and P. Stone. Continuous Area Sweeping: A Task Definition and Initial Approach. In *Proceedings of the International Conference on Advanced Robotics*, pages 316–323, 2005.

[2] M. Ahmadi and P. Stone. A Multi-Robot System for Continuous Area Sweeping Tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1724–1729, 2006.

[3] H. Bast and S. Hert. The Area Partitioning Problem. In *12th Canadian Conference on Computational Geometry*, 1995.

[4] L. Fu and G. Ishkhanov. Fleet Size and Mix Optimization for Paratransit Services. *Transportation Research Record*, 1884:39–46, 2004.

[5] S. Hert and V. Lumelsky. Polygon area decomposition for multiple-robot workspace division. *Special Issue of International Journal of Computational Geometry & Applications on Applied Computational Geometry*, pages 437–466, 1998.

[6] J. Laake and A. Zhang. An Optimization Model for Strategic Fleet Planning in Tramp Shipping. In *Proceedings of the Annual Conference of the Operations Research Society of New Zealand*, 2013.

[7] S. Liemhetcharat and M. Veloso. Modeling and Learning Synergy for Team Formation with Heterogeneous Agents. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 365–375, 2012.

[8] S. Liemhetcharat and M. Veloso. Forming an Effective Multi-Robot Team Robust to Failures. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5240–5245, 2013.

[9] S. Liemhetcharat and M. Veloso. Synergy Graphs for Configuring Robot Team Members. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 111–118, 2013.

[10] S. Liemhetcharat and M. Veloso. Weighted Synergy Graphs for Effective Team Formation with Heterogeneous Ad Hoc Agents. *Journal of Artificial Intelligence*, 208(2014):41–65, 2014.

[11] T. Service and J. Adams. Coalition formation for task allocation: theory and algorithms. *Journal of Autonomous Agents and Multi-Agent Systems*, 22:225–248, 2011.