Weighted Synergy Graphs for Role Assignment in Ad Hoc Heterogeneous Robot Teams

Somchaya Liemhetcharat and Manuela Veloso
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA
som@ri.cmu.edu and veloso@cs.cmu.edu

***Abstract*— Heterogeneous robot teams are formed to perform complex tasks that are sub-divided into different roles. In ad hoc domains, the capabilities of the robots and how well they perform as a team is initially unknown, and the goal is to find the optimal role assignment policy of the robots that will attain the highest value. In this paper, we formally define the weighted synergy graph for role assignment (WeSGRA), that models the capabilities of robots in different roles as Normal distributions, and uses a weighted graph structure to model how different role assignments affect the overall team value. We contribute a learning algorithm that learns a WeSGRA from training examples of role assignment policies and observed values, and a team formation algorithm that approximates the optimal role assignment policy. We evaluate our model and algorithms in extensive experiments, and show that the learning algorithm learns a WeSGRA model with high log-likelihood that is used to form a near-optimal team. Further, we apply the WeSGRA model to simulated robots in the RoboCup Rescue domain, and to real robots in a foraging task, and show that the role assignment policy found by WeSGRA attains a high value and outperforms other algorithms, thus demonstrating the efficacy of the WeSGRA model.**

I. Introduction

In ad hoc domains, robots from different sources are put together to solve complex tasks that are frequently sub-divided into roles, where each role defines the robot's initial state and its responsibility in the task. For example, in an urban search-and-rescue (USAR) scenario, rescue robots from around the world would arrive at the scene. However, due to space constraints, only a small number of robots can be deployed. Thus, the best combination of robots has to be chosen. Possible roles in USAR include ambulances that directly save civilians, fire engines that put out fires, and police cars that clear debris and direct civilians to safe areas. However, because of the ad hoc nature of the problem, the capabilities of the robots (i.e., how well they perform in the roles) are initially unknown, as well as how well combinations of robots will perform together at the task.

Previous work in role assignment, which we discuss in detail in the related work section, focus on cases where the capabilities of the robots are known. In multi-robot task-allocation, the goal is to maximize the total number of tasks completed, while we are interested in finding the optimal role assignment to complete a single task with highest value. Research in the ad hoc domain has thus far focused on how a learning agent can adapt to unknown teammates; we are interested in how to form a team of robots in an ad hoc problem. We have previously introduced the synergy graph model, that captures the capabilities of agents at a task, and their synergy when working in a team. However, the synergy graph model only selects a team of agents, and does not provide role assignments.

In this paper, we formally introduce the weighted synergy graph for role assignment (WeSGRA) model, where agent types (combinations of robot hardware and software) are vertices in a weighted graph, and the distance between agent types in the graph is related to how well they perform together in a team. Furthermore, each agent type is associated with a list of Normal distributions that describe the capability of the agent type at different roles. We formally define how the synergy of a role assignment policy is computed, and contribute a team formation algorithm that uses a WeSGRA to approximate the optimal role assignment policy.

Because the robot capabilities are initially unknown, the WeSGRA must be learned from data. Thus, we contribute a learning algorithm that learns a WeSGRA using only training examples of role assignments and their values in the task. In extensive experiments, we use a hidden WeSGRA to generate training data, and show that the learned WeSGRA is very similar, and that the role assignment policy found by using the learned WeSGRA is close to optimal.

We demonstrate the efficacy of the WeSGRA model with simulated robots and real robots. Using the RoboCup Rescue simulator, we run combinations of algorithms of RoboCup teams to solve the task, and learn a WeSGRA from the simulated results. Through cross-validation, we show that the WeSGRA model captures the capabilities and interactions of the robots, and we show that the role assignment policy found by using the learned WeSGRA outperforms that found by a market-based technique. In addition, we use a combination of Aldebaran NAO robots and Lego Mindstorms NXT robots in a foraging task, and learn a WeSGRA from the robots' performance. We demonstrate that the role assignment policy found by WeSGRA again outperforms the market-based technique. While we use NAOs and NXTs in our real robot experiments, the WeSGRA model is general, and can be applied to other robot platforms to form effective role assignment policies in ad hoc domains.

II. Related Work

In role assignment, a single complex task is sub-divided into a set of roles, where each role describes the responsibility of the robot. Market-based techniques are commonly used, where robots submit bids on roles and typically the

highest bid is assigned the role [1]. In IQ-ASyMTRe, heterogeneous robots with known capabilities (schemas) are considered, and ranking of feasible teams is based on the sum of schema costs [2], which bears many similarities to market-based approaches; hence, we compare our model and algorithms with a market-based approach. In our work, we are interested in ad hoc domains, where the robot capabilities are initially unknown, and we model the synergistic effects of role assignments.

Multi-robot task allocation (MTRA) has many similarities to role assignment, where the goal is to allocate multiple tasks instead of roles. Many approaches to MRTA compute the value of an allocation based on the tasks completed [3]. Market-based techniques are also commonly used in MRTA, and robot capabilities are modeled as a set of services or resources [4]. Similar to role assignment, most approaches do not model synergistic effects of team members. We are interested in how the performance of a task varies based on the team composition and role assignment, without prior information about robot capabilities.

The ad hoc problem was recently introduced, where an autonomous agent learns to collaborate with previously unknown teammates [5]. Role assignment of an ad hoc agent involves the agent choosing a role based on observations [6]. We are also interested in the ad hoc domain, and we focus on how to form an effective role assignment policy in an ad hoc scenario, and not the behavior of a single agent.

We recently introduced the synergy graph model, where agents are vertices in an unweighted graph, and the distance in the graph is related to how well they perform together in a team [7]. The agent capabilities are represented as Normal distributions, to capture variability and the dynamics of the world. Agent teams were represented as subsets of agents, without role assignments. In this paper, we build upon the synergy graph model and use a weighted graph to improve the expressiveness of the model. Our learning algorithm uses much less data for training, only a single point of data per role assignment, compared to 30 per team in the original synergy graph learning algorithm. Further, we now model how role assignments affect performance, and approximate the optimal role assignment policy.

III. Problem Statement and Approach

In this section, we formally define the problem, and give an overview of our approach. To aid in the explanation of the problem and solution, we will use a consistent motivating scenario from the RoboCup Rescue domain.

A. Motivating Scenario

An earthquake has occurred in a city, and civilians are trapped and require rescue. Fires have started, and fallen rubble has created road blockages. An urban search-and-rescue (USAR) team is deployed in the city, with the goal of saving as many civilians as possible, and also to minimize fire damage to the city. There are three kinds of USAR agents: ambulances that can save civilians, fire engines that put out fires, and police cars that clear rubble from roads.

Many USAR agents from around the world arrive at the disaster scene ready to help, but due to safety concerns, only a small number of USAR agents can be deployed. In particular, there is a fixed set of locations within the city that the USAR agents will be deployed from. Because the USAR agents come from different places, many have not worked together before and it is unknown how well they will be able to coordinate. Thus, the goal is to pick the best ad hoc team comprising USAR agents from different sources.

B. Formal Problem Definition

Let $A = \{a_1, \ldots, a_N\}$ be the set of agent types, where an agent type is a combination of the hardware of a robot and the software (i.e., the algorithms) controlling it. For example, physically-identical robots running different algorithms would be represented as different agent types, while physically-identical robots running the same algorithms would be represented by a single $a \in A$. Hence, N is the number of possible combinations of hardware and software, and not the total number of robots available.

The task is sub-divided in M roles — in the USAR scenario, these roles indicate both the type of robot (ambulance, fire engine, police car) and its initial location in the city. Let $R = \{r_1, \ldots, r_M\}$ be the set of roles, such that each role will be assigned an agent type.

The goal is to find the optimal role assignment policy $\pi^* : R \rightarrow A$, such that $\forall \pi, V(\pi^*) \geq V(\pi)$ for some value function V. In the USAR scenario, the value function V would be a combination of the number of civilians saved and the state of the buildings in the city. For any role assignment policy π, every role must be assigned an agent type, and it is valid for the same agent type $a \in A$ to be assigned multiple roles, i.e., $\exists r_\alpha, r_\beta \in R$ s.t. $\pi(r_\alpha) = \pi(r_\beta)$. Such an assignment means that multiple identical robots (robots with identical hardware and software) will each perform the role.

Since this is an ad hoc scenario, the capabilities of the agent types (how well they perform at different roles) are initially unknown, as well as how well different agent types perform together in a team. Thus, the value function V is unknown. Further, since robots are acting in a dynamic world, the value function V is non-deterministic, i.e., for the same π, multiple samples of $V(\pi)$ will return different numbers. However, examples of $V(\pi)$ for different π are available, and indicates how particular role assignments performed at the task. Using these examples, the optimal role assignment policy π^* is to be determined.

C. Overview of Approach

To solve this general problem of role assignment in an ad hoc scenario, we do the following:

1) We formally define the *Weighted Synergy Graph for Role Assignment (WeSGRA)*, that models the capabilities of agent types at the M roles as Normal distributions, and how well they work together with other agent types in a team (i.e., their *synergy*) with the structure of the WeSGRA graph.

2) We learn the WeSGRA using only the training examples of $V(\pi)$, that maximizes the log-likelihood of the training examples, and effectively models the capabilities and interactions of the agent types.
3) Using the learned WeSGRA, we approximate the optimal role assignment policy to perform the task.

IV. Weighted Synergy Graphs for Role Assignment

In this section, we formally describe how we model the value function V using a Weighted Synergy Graph for Role Assignment (WeSGRA), and how a WeSGRA is used to approximate the optimal role assignment policy.

A. Modeling Capabilities and Synergy

There are N agent types and M roles, and the goal is to find the optimal role assignment policy π^* subject to the value function V. We assume that the performance of a role assignment policy is represented by a Normal distribution, i.e., $V(\pi) \sim \mathcal{N}(\mu_\pi, \sigma_\pi^2)$. As in our previous work in modeling mutual capabilities and synergy [8], [7], we use a Normal distribution because it is unimodal (the single peak corresponds to the mean performance of the team) and the symmetry in deviation reflects that agents are just as likely to perform better as they are worse.

In particular, each agent type $a_i \in A$ is associated with M Normal distributions $\{C_{i,1}, \ldots, C_{i,M}\}$, where each $C_{i,\alpha}$ is the individual capability of a_i at role r_α, i.e., how well the agent type performs at the particular role. In [7], agent capabilities were also represented by a list of distributions, that indicated how well the agent performs at each sub-task. The key difference in this work is that the sub-tasks were assumed to be independent in [7], but in this paper, roles are interdependent and affect the overall task performance.

While the Normal distributions define the individual capabilities of agent types at different roles, it does not model how well agent types perform together. For example, suppose that in the USAR scenario, there are two roles $r_1, r_2 \in R$ involving ambulances deployed next to each other. Further suppose that agent type a_1 has high performance at r_1, r_2, and a_2 has low performance at r_1, r_2. However, if the role assignment policy is $(r_1 \rightarrow a_1, r_2 \rightarrow a_1)$, the overall task performance is low, because the algorithm of a_1 sends both agents to the same civilian. Conversely, if the policy is $(r_1 \rightarrow a_1, r_2 \rightarrow a_2)$, the task performance is higher because different civilians would be saved. Thus, the *synergy* of a team is not modeled by the individual capabilities.

To effectively model *synergy*, i.e., how well a team of agents will perform together, we introduce a compatibility function ϕ, where a high compatibility reflects that agent types have good synergy and work well together. Further, we define that compatibility is transitive — if a_i is highly compatible with a_2, and a_2 is highly compatible with a_3, then a_1 should be compatible with a_3. We concretely define the compatibility function as $\phi : \mathbb{Z}^+ \rightarrow \mathbb{R}^+$, where ϕ is a positive monotonically decreasing function. In this manner, ϕ takes as input a distance and returns compatibility, such that low distances correspond to high compatibility and vice versa. Examples of ϕ are $\phi(d) = \frac{1}{d}$ (a fraction function) and $\phi(d) = \exp(-\frac{d \ln 2}{h})$ (a decay function with half-life h). In addition, the transitive nature of compatibility is modeled by the sum of distances between agent types. We assume that ϕ is domain-specific and known.

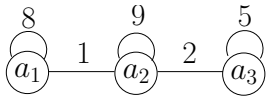

Fig. 1: An example of the distances among three agent types $a_1, a_2, a_3 \in A$, where a low distance between agent types reflects high compatibility and vice versa.

Fig. 1 shows an example of the distances of three agent types a_1, a_2, a_3, where the a_1 and a_2 have high compatibility (distance of 1) with each other, but low compatibility (distances of 8 and 9 respectively) with themselves, and a_3 has high compatibility with a_2 (distance of 2). As such, through transitivity, the compatibility of a_1 and a_3 will also be moderately high (distance of 3).

B. Formally defining the WeSGRA model

In the previous subsection, we defined the individual agent capabilities that represents how well an agent type performs at a role, and the compatibility function ϕ that relates how well pairs of agent types work together in a team. Putting these together, we now formally define the WeSGRA model:

Definition 4.1: A **weighted synergy graph for role assignment** (WeSGRA) model S is a tuple $\{G, C\}$, where:

- $G = (V, E)$ is a connected weighted graph;
- Each $a_i \in A$ is represented by a vertex $v_i \in V$;
- $e = (v_i, v_j, w_{i,j}) \in E$ is an edge between vertices v_i, v_j with weight $w_{i,j} \in \mathcal{Z}^+$;
- $\forall v_i \in V, \exists e = (v_i, v_i, w_{i,i}) \in E$, i.e., every vertex has an edge that loops to itself;
- $\forall a_i \in A, \exists C_i \in C$ s.t. $C_i = \{C_{i,1}, \ldots, C_{i,M}\}$ where $C_{i,\alpha} \sim \mathcal{N}(\mu_{i,\alpha}, \sigma_{i,\alpha}^2)$ is the capability of a_i at role r_α.

The WeSGRA model captures both the individual agent capabilities with C, and the compatibility ϕ between agent types through the distance between vertices in the graph. Thus, we modify the pairwise synergy function in [7] for the WeSGRA model:

Definition 4.2: The **pairwise synergy** between two agents a_i, a_j assigned to roles r_α, r_β respectively is:

$$\mathbb{S}_2(a_i, a_j, r_\alpha, r_\beta) = \phi(d(v_i, v_j)) \cdot (C_{i,\alpha} + C_{j,\beta}) \quad (1)$$

where $d(v_i, v_i) = w_{i,i}$, and for $i \neq j$, $d(v_i, v_j)$ is the shortest distance between vertices v_i, v_j in the WeSGRA graph.

Similarly, we define the synergy function that computes the distribution of a team of agents:

Definition 4.3: The **synergy** of the team of agents assigned by the role policy $\pi : R \rightarrow A$ is:

$$\mathbb{S}(\pi) = \frac{1}{\binom{|R|}{2}} \cdot \sum_{r_\alpha, r_\beta \in R} \mathbb{S}_2(\pi(r_\alpha), \pi(r_\beta), r_\alpha, r_\beta) \quad (2)$$

Thus, the synergy function $\mathbb{S}$ returns a Normal distribution that represents the performance of the team in the role assignment policy. The same agent type can have multiple roles in π, e.g., $\pi(r_\alpha) = \pi(r_\beta) = a_i$, which implies that the pairwise synergy function for this pair of roles will use the self-looping edge in the WeSGRA graph to compute the distance (and hence the compatibility).

C. Approximating the Optimal Role Assignment

Given a WeSGRA S, we want to approximate the optimal role assignment π^* using S. However, the synergy function $\mathbb{S}$ returns a Normal distribution, and we need to be able to rank such distributions. Hence, we use the evaluation function defined by us in [8], that converts a Normal distribution into a single number using a risk factor $\rho \in (0, 1)$:

$$\texttt{Evaluate}(N_\pi, \rho) = \mu_\pi + \sigma_\pi \cdot \Phi^{-1}(\rho) \tag{3}$$

where Φ^{-1} is the inverse of the standard Normal cumulative distribution function. As such, when $\rho = \frac{1}{2}$, the value returned equals the mean of the distribution, and the variance increases (decreases) the value when $\rho > \frac{1}{2}$ ($\rho < \frac{1}{2}$).

With the evaluation function that ranks distributions, our team formation algorithm approximates the optimal team by using simulated annealing to explore the space of role assignment policies. The algorithm starts with a random role assignment policy π, and generates neighbor policies by changing one of the agent types assigned to a role in π. New policies are accepted based on the score computed by `Evaluate` and the temperature schedule. We use simulated annealing as it is infeasible to compute the optimal by brute force (there are M^N possible policies).

V. WeSGRA Learning Algorithm

In the previous section, we contributed the WeSGRA model and our team formation algorithm that approximates the optimal role assignment policy. However, in order to use the WeSGRA model, we must first be able to learn a WeSGRA from data. In this section, we contribute our learning algorithm that learns a WeSGRA using only training examples of the performance of role assignment policies.

The value function V is unknown, but we are given training examples $T = \{(\pi_1, V_1), \ldots, (\pi_{|T|}, V_{|T|})\}$. Although each $V(\pi)$ is a distribution, we are only given a single sample per policy in T, i.e., $\forall (\pi_i, V_i), (\pi_j, V_j) \in T, \pi_i \neq \pi_j$.

Algo. 1 shows the learning algorithm, which proceeds in two steps. First, an initial WeSGRA graph structure in randomly generated, and the agent capabilities are estimated using the training examples T. Next, simulated annealing is run, where the WeSGRA graph structure is varied so as to explore the space of possible weighted graphs. With each new graph structure, the agent capabilities are re-estimated. The new WeSGRA is accepted based on the computed log-likelihood of the training examples.

A. Learning the WeSGRA graph structure

In Algo. 1, an initial WeSGRA graph structure is first generated by `RandomWeSGRAStructure`, that creates $|A|$ vertices, and randomly adds edges of random weights in the graph so as to make the graph connected. Self-looping edges of random weights are then added for every vertex.

Algorithm 1 Learn a WeSGRA from training examples

LearnWeSGRA(A, R, T)

1: $G = \texttt{RandomWeSGRAStructure}(A)$
2: $C \leftarrow \texttt{EstimateCapabilities}(G, R, T)$
3: $S \leftarrow (G, C)$
4: $l \leftarrow \texttt{LogLikelihood}(S, T)$
5: **for** $k = 1$ **to** $k_{\max}$ **do**
6: $\quad G' \leftarrow \texttt{NeighborWeSGRAStructure}(G)$
7: $\quad C' \leftarrow \texttt{EstimateCapabilities}(G', R, T)$
8: $\quad S' \leftarrow (G', C')$
9: $\quad l' \leftarrow \texttt{LogLikelihood}(S', T)$
10: $\quad$ **if** $\texttt{P}(l, l', \texttt{Temp}(k, k_{\max})) > \texttt{random}()$ **then**
11: $\quad\quad S \leftarrow S'$
12: $\quad\quad l \leftarrow l'$
13: **return** S

Next, `NeighborWeSGRAStructure` is used in the simulated annealing loop to explore the space of WeSGRA graph structures. A neighboring graph structure is generated by performing one of the four following actions:

- The weight of a random edge is increased by 1, subject to a maximum weight $w_{\max}$;
- The weight of a random edge is decreased by 1, subject to a minimum weight $w_{\min}$;
- A new edge is added between two vertices;
- An existing edge is removed, as long as it does not disconnect the graph.

Since the weights of edges in a WeSGRA graph structure are positive integers, the actions of increasing and decreasing the edge weights by 1 are capable of potentially exploring all edge weight combinations, and the last two actions (add and remove edges) changes which vertices are directly connected.

B. Estimating Agent Capabilities

Using an existing WeSGRA graph structure, and the training examples T, `EstimateCapabilities` learns the capabilities of the agents. There are N agents and M roles, and as such, there are NM Normal distributions to be estimated. We assume that $|T| > 2NM$ so that the problem is over-constrained as opposed to underconstrained.

Since the WeSGRA graph structure is known, the distance between any two vertices v_i, v_j can be computed. If $v_i \neq v_j$, then $d(v_i, v_j)$ is the shortest distance between them in the graph. If $v_i = v_j$, then $d(v_i, v_j) = w_{i,i}$, i.e., the weight of the self-looping edge of vertex v_i. From the distances, ϕ computes the pairwise compatibility.

`EstimateCapabilites` estimates Normal distributions so as to maximize the log-likelihood of the training examples T. From a single training example $(\pi, V(\pi)) \in T$, an equation is formed that involves the means and variances of the agent capabilities.

Fig. 2 shows an example of a WeSGRA graph structure with 3 agent types and 2 roles. Suppose that the agent

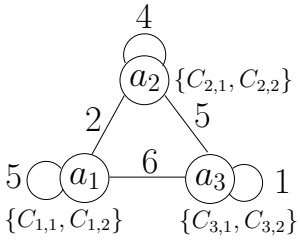

Fig. 2: An example of a WeSGRA with 3 agents types and 2 roles.

capabilities $C_{i,\alpha}$ are unknown, and that $\phi(d) = \frac{1}{d}$. Further suppose that $\pi = (r_1 \rightarrow a_1, r_2 \rightarrow a_2)$. The synergy $\mathbb{S}(\pi) = \phi(d(v_1, v_2))(C_{1,1} + C_{2,2}) = \frac{1}{2}(C_{1,1} + C_{2,2})$. Hence, the log-likelihood of the example $V(\pi)$ is:

$$-\frac{1}{2}\log(2\pi \cdot \frac{1}{4}(\sigma_{1,1}^2 + \sigma_{2,2}^2)) - \frac{(V(\pi) - \frac{1}{2}(\mu_{1,1} + \mu_{2,2}))^2}{2 \cdot \frac{1}{4}(\sigma_{1,1}^2 + \sigma_{2,2}^2)}$$

Thus, each training example in T corresponds to an expression involving the means and variances of the agent capabilities. In order to find the distributions that maximize the log-likelihood of T, the sum of log-likelihoods must be maximized. In particular, the means are first estimated using a least-squares solver, and then the variances are estimated using a non-linear solver given the means.

VI. Experiments and Results

The WeSGRA model captures the individual capabilities of agent types at different roles, and the synergy of the agent types in a team. We have also contributed a learning algorithm to learn a WeSGRA from training examples. In this section, we demonstrate the effectiveness of the WeSGRA model, the learning algorithm, and the role assignment algorithm to approximate the optimal team.

First, we use synthetic data generated from hidden WeSGRA models, and show that the learning algorithm returns a learned WeSGRA that is very similar to the hidden one, that is used to form a near-optimal role assignment policy. Secondly, we use the RoboCup Rescue Agent Simulator and run combinations of existing algorithms written by RoboCup teams. We show that the WeSGRA learned from the simulation data models the interactions between the algorithms well, and forms an effective role assignment policy. Thirdly, we perform experiments on real robots solving a task, using combinations of Aldebaran NAO humanoid robots and Lego NXT robots, and show that the WeSGRA learned from the experimental data forms a team that performs very well at the task. In all our experiments, we assume that $\phi(d) = \frac{1}{d}$ is the domain-specific compatibility function.

A. Learning from Synthetic Data

In our first set of experiments, we ran 100 trials using synthetic data. In each trial, we randomly created a hidden WeSGRA with 5 agent types and 5 roles, that we then used to generate 500 training and 500 test examples of $V(\pi)$ for different role assignment policies π. Since $N = M = 5$, the size of the policy space is $5^5 = 3125$ and thus the training

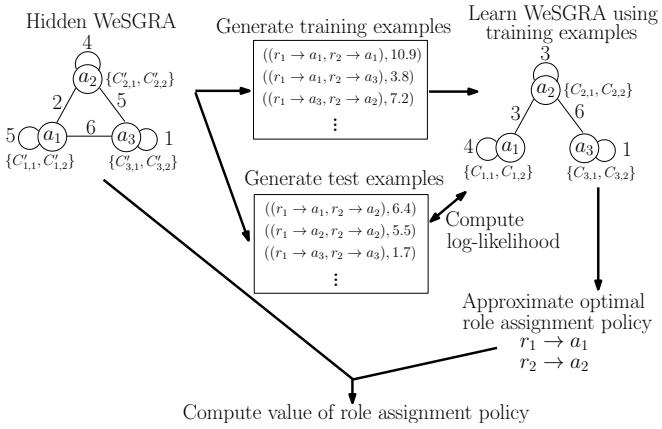

Fig. 3: The experimental process to evaluate the learning and team formation algorithms for WeSGRA.

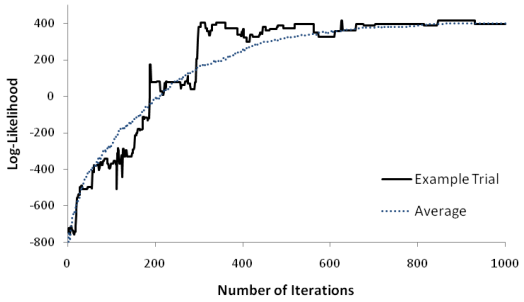

Fig. 4: Learning curve of the learning algorithm using training examples generated by a hidden WeSGRA model.

examples do not cover the space. The training examples were used to learn a WeSGRA, and the accuracy of the learned model was measured using the log-likelihood of the test examples. Next, the learned model was used to approximate the optimal role assignment policy. The value of this policy was obtained by using the hidden WeSGRA, and compared to the optimal (which was found by a brute force search). The hidden WeSGRA was only involved in generating the training and test examples, and evaluating role assignment policies. Fig. 3 shows the steps of the experiments.

Fig. 4 shows the learning curve of our learning algorithm over 1000 iterations of simulated annealing. The dotted line shows the average learning over all 100 trials, while the full line shows one trial. The log-likelihood of the test data becomes higher as the number of iterations increases, showing that our learning algorithm is capable of learning a WeSGRA that closely resembles the hidden one.

To compute the effectiveness of the role assignment policy selected by our team formation algorithm, for each trial, we found (through a brute-force search) the optimal and worst role assignment policies in the hidden WeSGRA, by computing the Normal distribution of every policy and converting it into a number using `Evaluate` with $\rho = \frac{1}{2}$. Next, the value of the role assignment policy found from the learned WeSGRA (v_{learned}) was also computed. Since the value of policies (optimal, minimum, and learned) differed

across trials, we scaled them to be between 0 and 1:

$$\text{Effectiveness} = \frac{v_{\text{learned}} - v_{\text{min}}}{v_{\text{max}} - v_{\text{min}}} \quad (4)$$

where v_{max} and v_{min} are the values of the optimal and worst policies respectively.

Over the 100 trials with synthetic data, the effectiveness of the role assignment policy found from the learned WeSGRA graph was **0.97** $\pm$ **0.08**, which shows that the learned WeSGRA closely matches the hidden one, and also that the team formation algorithm is capable of finding a near-optimal team using the learned graph.

B. Forming an Ad Hoc RoboCup Rescue Team

The previous subsection showed that the learning algorithm effectively learns a WeSGRA when the data does in fact come from a hidden WeSGRA. We now apply the algorithms to data generated from the RoboCup Rescue Agent Simulator, to demonstrate that the WeSGRA model captures the interactions of ad hoc agents well.

1) Defining the Task: The RoboCup Rescue Agent Simulation League releases the simulator, maps, and source code of participating teams annually. We compiled and ran the algorithms of 6 RoboCup teams (with minor modifications); each agent type in the WeSGRA model corresponded to a RoboCup team's algorithm. We used the Istanbul1 scenario from the actual RoboCup 2011 competition, where there are 46 rescue robots to be controlled. Typically, one team's algorithm is used to control and coordinate all 46 robots at once. However, because we are interested in modeling the interactions of multiple algorithms in an ad hoc setting, we did the following: for each of the 46 robots, one of the 6 algorithms was chosen at random to control it. Thus, a role assignment policy in this case would be an assignment of algorithms to each of the 46 rescue robot, and as such there are 6^{46} possible role assignment policies.

2) Performing the Experiments: We randomly generated 600 policies (i.e., policies with different combinations of RoboCup algorithms), and ran an instance of the simulator per policy. At the end of each simulation, the value was retrieved from the simulator, which was a weighted sum of the health of the civilians and rescue agents still alive, and the status of the buildings of the city.

First, we performed 6-fold cross-validation on the 600 examples, where we split the data in 6 sets of 100, and trained on 500 and tested on the remaining 100. However, because of the small number of training examples, we were unable to estimate the variances of the agent type capabilities $C_{i,\alpha}$, and instead set them to be a constant. Fig. 5 shows the cross-validation learning curves, and that the log-likelihood of the test set improves with the number of iterations of simulated annealing, and illustrates that the WeSGRA model is capable of modeling the interactions of the role assignment policies running with the 6 RoboCup algorithms.

Next, we used the learned WeSGRAs from the 6 runs of cross-validation to approximate the optimal role assignment policy. The policies found were then run in the simulator to retrieve its value. We also learned a WeSGRA using all 600 examples, and used the simulator to obtain the value of the role assignment policy found from the learned WeSGRA. We compared the role assignment policies found by our approach with three methods: a random guess, a market-based approach, and picking the policy with the highest value in the training examples. The value of choosing a random policy was computed based on the 600 examples. In the market-based approach, each algorithm a_i formed a different bid $\text{Bid}(a_i, r_\alpha)$ for each of the 46 roles r_α, based on the value of policies in the 600 examples that contained it:

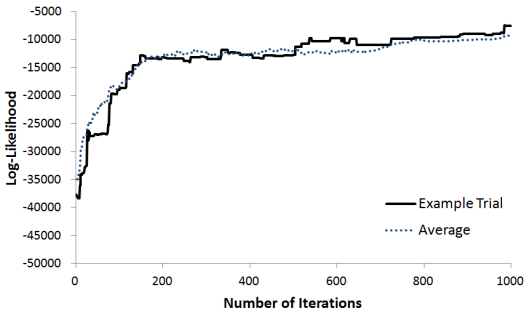

Fig. 5: Learning curve of the learning algorithm using cross-validation of data from the RoboCup Rescue simulator.

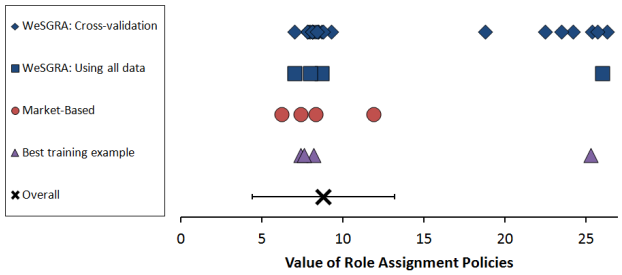

Fig. 6: The distribution of values of role assignment policies. The values in the training examples is shown as a cross (the mean) with horizontal lines showing the standard deviation.

$$\text{Bid}(a_i, r_\alpha) = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} V(\pi) \quad (5)$$

$$\text{where } \Pi = \{\pi \in T : \pi(r_\alpha) = a_i\} \quad (6)$$

For each role, the market-based role assignment algorithm picked the agent type with the highest bid.

Fig. 6 shows the distribution of values of the role assignments found by WeSGRA (from cross-validation, and from using all the data) and methods we used for comparison. The role assignments formed by WeSGRA generally outperforms a random policy, and also the market-based algorithm. The performance of WeSGRA using all the data is similar to picking the best policy in the training data, but we believe this is due to the small size of training data, and WeSGRA's performance will improve with more examples, while choosing the best training policy can lead to over-fitting.

C. Applying WeSGRA on Real Robots

For our third set of experiments, we applied the WeSGRA model to real robots in the foraging domain. We used two hardware platforms — Aldebaran NAO humanoid robots,

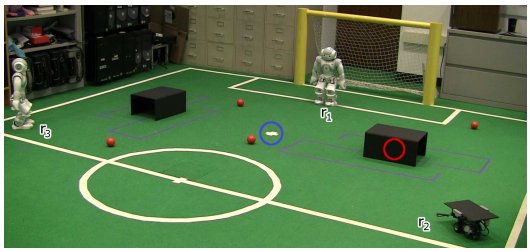

Fig. 7: The experimental setup for the real robot experiments. The red circle indicates a hidden fifth ball, and the blue circle indicates where balls are replaced if they are moved past the side and back lines. Different combinations of robots were placed in the 3 robot roles r_1, r_2, r_3.

and Lego Mindstorms NXT robots. While we used these two robot platforms, our results are general and can be applied to other robot types. To increase the heterogeneity in the robots, we also varied the algorithms the NAOs ran, which we will elaborate in detail below. The foraging domain was chosen as it bears many similarities to the USAR domain, namely searching and "rescuing" in a limited amount of time.

1) Defining the Task: For the foraging task, 3 roles r_1, r_2, r_3 were defined with starting locations in one half of a RoboCup Standard Platform League soccer field, where 5 balls were placed (Fig. 7). 4 of the balls were in open areas and easily seen by the NAOs, while 1 ball was hidden from view under a tunnel, thus requiring an NXT to handle it. The robots were to find and move as many balls as possible to the other half of the field, in as little time as possible. If a ball was moved outside the half of the field (i.e., the side or back lines), the ball was replaced in the middle of the half (denoted by the blue circle in Fig. 7). The value of the team was based on the number of balls foraged and the time in which the balls were foraged:

$$\mathrm{V} = v_{\text{ball}} \cdot |B| + \sum_{b \in B} (t_{\text{total}} - t_b) \tag{7}$$

where v_{ball} is the value per ball foraged, B are the balls foraged, t_b is the time elapsed (in seconds) when b was foraged, and t_{total} is the total time of the trial in seconds.

2) Types of Robots: The NAO robots had three different algorithms: Chase-and-Kick (CK), Kick-past-Midline (KM), and Observer (Ob). A NAO running CK would search for a ball, walk to it, and then perform a straight kick in whichever direction the robot was currently facing. In the KM algorithm, the NAO would explicitly localize (based on landmarks on the field such as the yellow and blue goal posts) and attempt to kick the ball past the middle line. Thus, the KM algorithm explicitly kicks the ball towards the target area (the other half of the field), while the CK algorithm kicks in any direction, but over time CK succeeds in foraging since balls kicked past the side and back lines are replaced. Also, in both algorithms, the NAO would perform obstacle avoidance so as not to walk into another NAO, NXT or the tunnels.

The Ob algorithm of the NAO did not actively forage the balls — instead, the NAO would search for balls, and transmit the ball's position through a wireless connection to other NAOs in the team, and as such potentially reducing the amount of time another NAO needed to find a ball.

The NXT robots were programmed to follow lines, such that they moved straight across the green field, turned around when they encountered a white line, and followed a purple line. Thus, the NXT robot would move across the field until it found one of the two sets of purple lines around a tunnel, and then followed it endlessly (it turned around at the end of the purple line). If a ball was present inside the tunnel, the NXT would push it out as part of the line-following behavior. As such, the NXT robots were not capable of foraging the balls directly, but only assisted the overall team goal.

The robots in our experiments ran autonomously, without any central computer or processing. The only information communicated between the robots was the ball position that the Ob algorithm sent to KM, that KM would use to approach the ball. We chose these algorithms so that there was a greater heterogeneity in the robots, and also that some algorithms were explicitly "helper" types (i.e., Ob and NXTs), where they would not attain any value on their own, but can improve the overall value given the right teammates. The video accompanying our paper shows the setup, robot behaviors and role assignment found by our algorithm.

3) Performing the Experiments: There were 3 roles, and 4 possible agent types (CK, KM, Ob, NXT), and thus there were $4^3 = 64$ possible role assignment policies. We performed 61 of these policies and recorded the times in which balls were successfully foraged (3 combinations involving all NAOs could not be run due to hardware problems).

Next, we set $v_{\text{ball}} = 100$, and computed the values of each of the combinations, given different amounts of time per trial. For example, if t_{total} was 120 seconds, then we ignored all balls foraged after 120 seconds. We varied t_{total} from 120 to 600 seconds at 60 second intervals.

For each value of t_{total}, we performed 6-fold cross-validation on the data, comparing the WeSGRA model (using 4 agent types and 3 roles) with the market-based algorithm (Eqn. 5). Due to the small number of training examples, we again fixed the variances in the WeSGRA model to a constant. As t_{total} increases, the values of policies found by both algorithms increase, as there is more time for the robots to forage balls. To compare the performance of the algorithms, we used the effectiveness measure (Eqn. 4) to scale the results from 0 to 1. Table I shows the results of WeSGRA and the market-based algorithm. WeSGRA outperforms the market-based algorithm across all values of t_{total}. Using a single-tailed paired-sample T-test, we found that our results are statistically significant with $p = 0.00003$.

The market-based technique picked the role assignment ($r_1 \rightarrow$ CK, $r_2 \rightarrow$ KM, $r_3 \rightarrow$ KM) regardless of t_{total} and as such the team could only forage the 4 visible balls. The WeSGRA model picked teams involving 1 NXT and 2 NAOs (running CK and KM), and were thus able to forage more balls in general and attain a higher value. Thus, the WeSGRA model successfully modeled the interactions between the helper types of robots. While the Ob algorithm helped to provide ball information to the other NAOs, it was not selected as part of the optimal team, as having more NAOs walking in the field provides a larger benefit.

Algorithm	t_{total}								
	120	180	240	300	360	420	480	540	600
WeSGRA	0.91 ± 0.11	0.92 ± 0.11	0.96 ± 0.04	0.93 ± 0.08	0.92 ± 0.07	0.93 ± 0.06	0.88 ± 0.06	0.88 ± 0.06	0.90 ± 0.08
Market-based	0.80 ± 0	0.80 ± 0	0.88 ± 0	0.93 ± 0	0.90 ± 0	0.89 ± 0	0.87 ± 0	0.86 ± 0	0.86 ± 0

TABLE I: Effectiveness of algorithms in the foraging domain with real robots.

VII. Conclusions

We formally introduced the weighted synergy graph for role assignment (WeSGRA) model, that models the capabilities of robots at different roles and how well they perform in a team. Each agent type (a combination of robot hardware and software) is represented with a vertex in the WeSGRA model, with a list of Normal distributions as associated capabilities for each role. The WeSGRA model uses a weighted graph to represent compatibility among agent types, where a smaller distance in the graph corresponds to higher compatibility (i.e., agent types work better together).

We contributed a team formation algorithm that approximates the optimal role assignment policy given a WeSGRA. However, in order to use a WeSGRA on actual ad hoc domains, it must first be learned from data. Thus, we contributed a learning algorithm that uses training examples of role assignment policies and the values they attained, and iteratively learns a WeSGRA by varying the weighted graph structure and estimating the capabilities of the agent types.

To demonstrate the efficacy of our learning and team formation algorithms, we first performed experiments using data generated from a hidden WeSGRA. We showed that the learned WeSGRA has high log-likelihood compared to test data, and that the role assignment policy found by searching the learned WeSGRA is in fact close to optimal.

Next, we applied the WeSGRA model to the RoboCup Rescue domain, running combinations of existing RoboCup algorithms in the RoboCup simulator. From the data, we performed cross-validation and showed that the learned WeSGRA represents the underlying interactions well. We further compared the role assignment policies found by using the learned WeSGRA against a market-based algorithm, and showed that our policies performed much better.

Real robot experiments were also performed in a foraging task, using Aldebaran NAO robots and Lego Mindstorms NXT robots. We collected data from different possible role assignment policies, and each policy's value was based on the number of items foraged and the time per foraged item. We showed that the WeSGRA learned from the data selects a role assignment policy that outperforms the market-based algorithm, across all values of the maximum time to forage.

While we used two types of robot hardware, the WeSGRA model is directly applicable to other robot platforms. The model effectively captures robot capabilities and their synergy in the team, and uses only training examples with no prior information. Thus, the WeSGRA model and algorithms can be readily applied to many ad hoc robot problems.

Acknowledgments

The authors thank Marcos Maximo for his work with the NXTs, and Junyun Tay for her help with the video. This work was partially supported by the Air Force Research Laboratory under grant no. FA87501020165, by the Office of Naval Research under grant number N00014-09-1-1031, and the Agency for Science, Technology, and Research (A*STAR), Singapore. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

References

[1] E. G. Jones, B. Browning, M. B. Dias, B. Argall, M. Veloso, and A. Stentz, "Dynamically Formed Heterogeneous Robot Teams Performing Tightly-Coordinated Tasks," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2006, pp. 570–575.

[2] Y. Zhang and L. Parker, "Task Allocation with Executable Coalitions in Multirobot Tasks," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2012, pp. 3307–3314.

[3] B. P. Gerkey and M. J. Mataric, "A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems," *Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

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

[5] P. Stone, G. Kaminka, S. Kraus, and J. Rosenschein, "Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination," in *Proceedings of the International Conference on Artificial Intelligence*, 2010.

[6] K. Genter, N. Agmon, and P. Stone, "Role-Based Ad Hoc Teamwork," in *Proceedings of the Plan, Activity, and Intent Recognition Workshop at the Twenty-Fifth Conference on Artificial Intelligence (PAIR-11)*, 2011.

[7] S. Liemhetcharat and M. Veloso, "Modeling and learning synergy for team formation with heterogeneous agents," in *Proc. Int. Conf. Autonomous Agents Multiagent Systems*, 2012.

[8] S. Liemhetcharat and M. Veloso, "Modeling mutual capabilities in heterogeneous teams for role assignment," in *Proc. Int. Conf. Intelligent Robots Systems*, 2011, pp. 3638–3644.