2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Learning the Synergy of a New Teammate

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*— In many multi-robot problems, the performance of a team of robots is not the sum of their individual capabilities; there is often *synergy* among the robots. We recently introduced the synergy graph model to model such phenomena, where robots are represented by vertices in a graph, their capabilities represented by Normally-distributed variables, and the interactions of robots represented with the structure of the graph. The synergy graph is learned from observations of robot team performances, with the underlying assumption that observations of all the robots are available at once. However, it is common that new information becomes available over time, in particular as new robots enter the domain. In this paper, we contribute a learning algorithm that uses new information to add a new robot into an existing synergy graph, that requires a smaller number of observations and faster computation than relearning the entire synergy graph using the existing learning algorithms. We introduce three heuristics to initialize the learning algorithm, and perform extensive simulations to analyze their characteristics, as well as compare two methods of learning robot capabilities, over a variety of graph structure types. We also compare three approaches to learning synergy graphs, and demonstrate that adding a new teammate into an existing synergy graph introduces higher error than completely relearning the synergy graph. However, it is computationally less expensive to add a new teammate, especially when the number of robots is large.**

I. Introduction

Multi-agent and multi-robot teams have been applied to a variety of domains, such as exploration, resource gathering, and robot soccer. In some domains, the task performance depends on the team composition. Further, the performance of a team may not be the sum of each robot's capabilities: there may be *synergy* among robots in the team. We recently introduced the synergy graph model for such problem domains [1], [2]. Robots are modeled as vertices in a graph, and their capabilities are Normally-distributed variables to model their performance in a dynamic environment. The synergy graph is learned from observations of robots at the task, and the learning algorithms learn both the graph structure and robot capabilities without other information of the task, with the assumption that observations of all the robots are available initially. While the synergy graph model readily models synergies among robots, the existing learning algorithms are unable to incorporate new information without relearning the entire model. In particular, it is likely for new robots to become available over time to perform the task.

In this paper, we contribute a new learning algorithm for the synergy graph model, that uses new information to add a new robot into an existing synergy graph. We assume that a synergy graph has already been learned using the existing algorithms and prior information, and our goal is to use the new information to update the model with another robot. To do so, we build upon the existing algorithms, and similarly use a simulated annealing approach to iteratively improve the learned synergy graph.

Since we use simulated annealing, which is an approximation technique, the initial starting point in the search space may be important. Hence, we introduce three different heuristics to initialize the learning algorithm, and perform extensive experiments with a variety of graph structure types to evaluate these heuristics and analyze their characteristics. Further, the existing synergy graph learning algorithms use two different techniques to learn robot capabilities: a matrix least-squares approach and a non-linear equation solving approach. We also perform experiments in simulation to analyze these two techniques and their effects on our new learning algorithm. Lastly, we compare the performance of our algorithm to completely relearning the synergy graph when new information is available, and show that while higher error is introduced by learning only the newest member, it is computationally much cheaper to do so, especially as the number of existing robots in the graph is large.

II. Related Work

Multi-robot teams are used in task allocation, where the goal is to allocate tasks to robots so as to maximize some utility function [3]. The performance is *task-based*, where the overall performance is the sum of utilities of completed tasks. To model heterogeneous robots, a common technique is to model tasks and robots as lists of resources [4], or as lists of services, where each robot performs one service [5].

We are interested in *team-based* performance, where performance depends on the team composition. Coalition formation is a related field, where each possible coalition (subset of agents) is given a value by a characteristic function. While the general coalition formation problem is intractable [6], heuristics have been used to apply coalition formation to task allocation [7]. Externalities in coalition formation have also been considered, where the value of a coalition depends on the structure of the other coalitions (e.g., [8], [9]).

Recently, we introduced the synergy graph model for team formation, where the performance of a team of robots depends critically on its composition [1]. An extension to the model was also introduced, that applies synergy graphs to the role assignment domain [2]. We will elaborate on the synergy graph model in the next section. Our interest is in using the synergy graph model for team formation, and our goal is to be able to incorporate information about a new teammate to update a learned synergy graph.

978-1-4673-6357-0/13/$31.00 ©2013 IEEE

III. The Synergy Graph Model

We recently introduced the Synergy Graph (SG) model [1], that models the team performance of any team of robots. A team is a subset $R \subseteq \mathcal{R}$, where $\mathcal{R}$ is the set of all robots. The Weighted Synergy Graph for Role Assignment model (WeSGRA) extends the Synergy Graph, and models the performance of role assignments [2]. In this work, we use the Weighted Synergy Graph by combining features of both models [1], [2]:

Definition 3.1: A **Weighted Synergy Graph** (WeSG) is a tuple $\{G, C\}$, where:

- $G = (V, E)$ is a connected weighted graph;
- Each $r_i \in \mathcal{R}$ is represented as a vertex $v_i \in V$;
- $e = (v_i, v_j, w_{i,j}) \in E$ is an undirected edge between vertices v_i, v_j with weight $w_{i,j} \in \mathcal{Z}^+$;
- $C = (C_1, \ldots, C_N)$ is a list of robot capabilities, where $C_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ is the capability of robot $r_i \in \mathcal{R}$.

Using the WeSG, the performance of a team of robots is computed with the synergy function [1]:

Definition 3.2: The **synergy** $\mathbb{S}(R)$ of a team of robots $R \subseteq \mathcal{R}$ is:

$$\mathbb{S}(R) = \frac{1}{\binom{|R|}{2}} \cdot \sum_{\{r_i, r_j\} \in R} \mathbb{S}_2(r_i, r_j) \tag{1}$$

$$\mathbb{S}_2(r_i, r_j) = \phi(d(v_i, v_j)) \cdot (C_i + C_j) \tag{2}$$

where $\phi : \mathcal{Z}^+ \to \mathcal{R}^+$ is a monotonically decreasing compatibility function, and $d : V \times V \to \mathcal{Z}^+$ is the shortest distance between two vertices in the Weighted Synergy Graph.

Fig. 1 shows an example of a Weighted Synergy Graph representing 6 robots $r_1, \ldots, r_6$. While the robots r_2, r_3, and r_6 individually have lower mean capabilities than r_1, r_4, and r_5, the synergy of the first subset is $\mathbb{S}(\{r_2, r_3, r_6\}) \sim \mathcal{N}(9.1, 2.4)$ while the synergy of the second subset is $\mathbb{S}(\{r_1, r_4, r_5\}) \sim \mathcal{N}(4.9, 0.4)$. The first team $\{r_2, r_3, r_6\}$ has a higher synergy than the second because their pairwise distances are smaller, which represents that they work better together. Thus, the WeSG model captures interactions of robots beyond the sum of their capabilities.

A. Learning the Synergy Graph

Synergy graphs model the team performance of robots, and it is learned from observations of robots' performance. In this subsection, we briefly summarize the learning algorithms of the SG and WeSGRA models [1], [2].

Let O be the set of observations used for learning the synergy graph, i.e., the training data. Each $o \in O$ is an tuple (t, p), where $t \in T$ is a team and $p \in \mathbb{R}$ is the observed performance of the team. The SG learning algorithm assumes that all teams of size 2 and 3 are observed with multiple observations each, and the WeSGRA learning algorithm assumes that some subset of T is observed, with only one observation per observed team.

The goal of the learning algorithms is to learn a synergy graph that models the performance of all teams, including those not in O. To do so, both learning algorithms use simulated annealing to approximate the best-fitting synergy graph. Fig. 2 shows the learning process. A random graph structure is initially created (unweighted for SG and weighted for WeSGRA). The graph structure and observations O is used to compute the robot capabilities C — the structure and C then define an initial guess of the model. Within the simulated annealing loop, the graph structure is modified and robot capabilities recomputed to generate a neighbor guess. The neighbor is accepted based on the log-likelihood of training data (compared with the current best estimate) and the temperature schedule. In this way, the space of graph structures is explored and the closest synergy graph is learned at the end of the algorithms. Further details of these learning algorithms can be found in the original papers [1], [2].

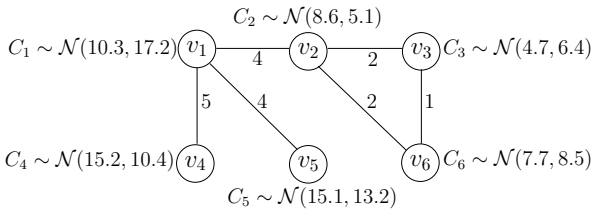

Fig. 1: An example of a Weighted Synergy Graph representing 6 robots.

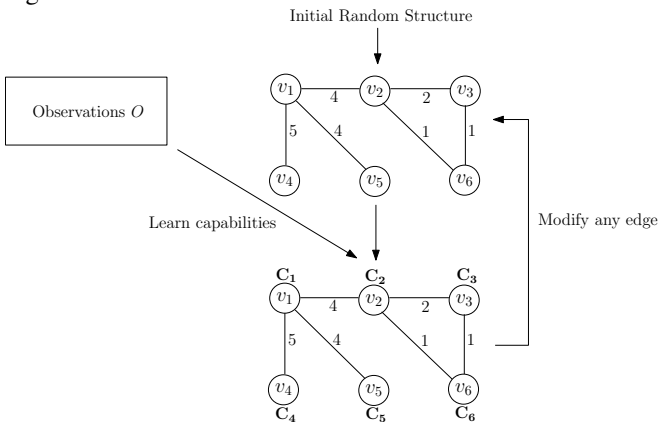

Fig. 2: The process of the Synergy Graph and WeSGRA learning algorithms used to learn a synergy graph from observations. Simulated annealing is performed where the graph structure is modified and robot capabilities are learned.

IV. Adding a New Teammate

The SG and WesGRA models are learned from data, with the underlying assumption that data about all the robots are available initially. This paper focuses on how to incorporate new information into the model; we are interested in the case of adding a new teammate into a synergy graph.

Let $\mathcal{R} = \{r_1, \ldots, r_N\}$ be the set of robots initially known. The prior learning algorithms assume that observations O about teams in $\mathcal{R}$ are available, and the algorithms learn a synergy graph model S from O.

Now suppose there is a new robot r_{N+1}, and let the new set of robots be $\mathcal{R}^+ = \mathcal{R} \cup \{r_{N+1}\}$. Also suppose that observations about the new robot are available, and let O_{N+1} be the set of new observations. We will elaborate on the details of O_{N+1} later. The goal is to learn an updated synergy graph S^+ that models the robots $\mathcal{R}^+$.

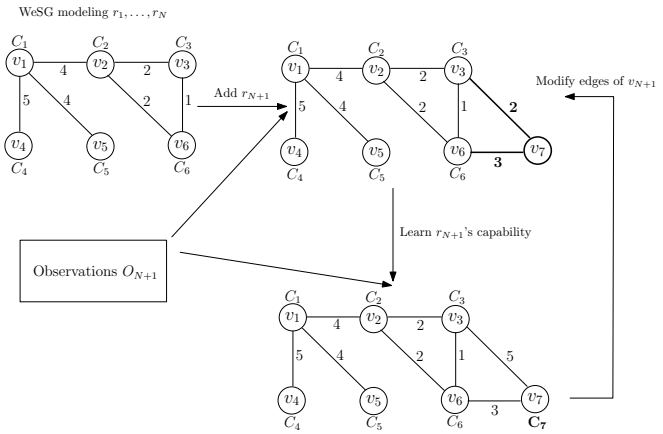

Fig. 3: Our learning algorithm that adds a new teammate into a WeSG. Simulated annealing is performed, where edges of v_{N+1} are modified, and the capability C_{N+1} is learned.

The existing learning algorithms are capable of learning a synergy graph to model $\mathcal{R}^+$ — namely, by using the union of all the observations $O^+ = O \cup O_{N+1}$. However, using O^+ to learn a new model has two main drawbacks. Firstly, the previously learned synergy graph S is discarded. Secondly, the runtime required to learn the new S^+ is high: the learning algorithm of SG has a runtime of $\mathcal{O}(N^3)$, and the learning algorithm of WeSGRA has a runtime of $\mathcal{O}(|O^+|)$.

Algorithm 1 Add a new robot to a WeSG

AddRobotToWeSG(S, r_{N+1}, O_{N+1})

1: // Add a new vertex to represent r_{N+1}
2: $V^+ \leftarrow V \cup v_{N+1}$
3: // Generate initial edges to connect r_{N+1} to $\mathcal{R}$
4: $E_{\text{initial}} \leftarrow$ GenerateEdges(S, r_{N+1}, O_{N+1})
5: $E^+ \leftarrow E \cup E_{\text{initial}}$
6: // Form the new WeSG graph structure
7: $G^+ \leftarrow (V^+, E^+)$
8: // Learn r_{N+1}'s capability
9: $C_{N+1} \leftarrow$ LearnCapability(G^+, O_{N+1})
10: // Form the initial WeSG
11: $S^+ \leftarrow (G^+, C \cup C_{N+1})$
12: $score \leftarrow$ LogLikelihood(S^+, O_{N+1})
13: // Simulated annealing loop
14: **for** $k = 1$ **to** $k_{\max}$ **do**
15: $\quad G' \leftarrow$ NeighborEdges(G^+, r_{N+1})
16: $\quad C'_{N+1} \leftarrow$ LearnCapability(G', O_{N+1})
17: $\quad S' \leftarrow (G', C \cup C'_{N+1})$
18: $\quad score' \leftarrow$ LogLikelihood(S', O_{N+1})
19: $\quad$ **if** $\mathbb{P}(score, score', \text{temp}(k, k_{\max})) > \text{random}()$ **then**
20: $\quad\quad S^+ \leftarrow S'$
21: $\quad\quad score \leftarrow score'$
22: **return** S^+

A. Using Prior Information

Our approach is to use the previously learned synergy graph S to jump-start the learning. We assume that S models the interactions and capabilities of the robots in $\mathcal{R}$, and only seek to learn the robot r_{N+1}'s capability, and the edges that connect r_{N+1} to the other robots.

Fig. 3 shows our learning process, and Algorithm 1 shows the pseudocode. While we use the Weighted Synergy Graph model in Algorithm 1, it is applicable with minor changes for the original Synergy Graph model as well as the WeSGRA model. The key difference in our learning algorithm is that we only consider edges that directly connect r_{N+1} to other robots in $\mathcal{R}$, and only learn r_{N+1}'s capability.

There are three main components to our learning algorithm: generating the initial edges (GenerateEdges), generating neighbor edges in the simulated annealing loop (NeighborEdges), and learning r_{N+1}'s capability (LearnCapability) given the structure of the graph, We will explain each of these three functions in detail below.

B. Generating Initial Edges

The goal of the GenerateEdges function (Line 4 of Algorithm 1) is to generate edges that connect the new vertex v_{N+1} (that represents the new robot r_{N+1}) to the other vertices $(v_1, \ldots, v_N)$. Recall that a WeSG is a connected weighted graph, and so v_{N+1} must have at least one edge that directly connects it to another vertex; the existing graph G is assumed to be connected since it is a valid WeSG.

One method is to randomly generate these edges. While this random method is naive, it provides a baseline for other heuristics introduced below, and does not require any information about the domain or existing synergy graph (only a probability p is required). A probability $0 < p \leq 1$ is defined, and for every possible edge connecting v_{N+1} with another vertex, a dice is thrown and the edge is created with probability p. The weight of a created edge is also randomly chosen to be an integer between $w_{\min}$ and $w_{\max}$. The while-loop ensures that at least one edge is created so v_{N+1} is connected to some other vertex.

While generating random edges suffices to create an initial guess, it requires a defined probability p, which may be domain-specific and difficult to ascertain. An improved method, GenerateEdgesWithDensity, first estimates p by examining the existing WeSG S and determining the density of edges (i.e., number of edges divided by the number of possible edges) in that graph. The underlying assumption is the number of edges connecting the new vertex v_{N+1} with other vertices is similar to the edge density of S. As such, GenerateEdgesWithDensity maintains the domain-independence of GenerateRandomEdges, while not requiring any probability p to be set, and instead learns it from the existing synergy graph.

The third method to generate the initial edges also uses the existing edges of the synergy graph S. However, instead of computing the density of edges, GenerateSimilarEdges finds the robot $r_i \in \mathcal{R}$ most similar to r_{N+1}, and duplicates all its edges. The similarity between robots is computed using the observations in O_{N+1}, which contain observations of teams containing r_{N+1}. For example, suppose that one observation $o \in O_{N+1}$ is (R', v), which indicates a team $\{r_{N+1}, r_j\} = R' \subseteq \mathcal{R}$ that had a performance of v. A new synthetic

observation $o' = (\{r_i, r_j\}, v)$ is created where r_{N+1} is replaced with r_i. All such synthetic observations form a new set O', and the log-likelihood of O' given S is computed. The most similar robot to r_{N+1} is then the one with the highest log-likelihood. GenerateSimilarEdges assumes that the new robot is similar to another robot already present in the synergy graph S, and hence uses its edges as a starting point for the learning algorithm. Such an assumption is domain-dependent, and hence GenerateSimilarEdges may outperform the above two heuristics in certain domains (when the new robot resembles an existing one) but may perform more poorly when a completely new robot is introduced.

C. Generating Neighbor Edges

The three functions described above create an initial guess of the edges connected v_{N+1} to the other vertices in the WeSG. During the simulated annealing, new candidate WeSG structures are generated, so as to effectively explore the space of all possible edges. We use the same four actions of neighbor generation as the WeSGRA learning algorithm [2], except that we only consider edges involving v_{N+1}, i.e., an edge $e = (v_{N+1}, v_i, w)$ (the WeSGRA learning algorithm modifies any edge in the graph):

- Remove an existing edge if it does not disconnect v_{N+1}
- Add a new edge with a randomly-generated weight w
- Increase the weight of an edge by 1, subject to $w_{\max}$
- Decrease the weight of an edge by 1, subject to $w_{\min}$

There are $(w_{\max} - w_{\min} + 1)^N$ possible edges that connect v_{N+1} to the other vertices, and so it is infeasible to consider all combinations of edges. Through these four actions, we can explore the space of such edges iteratively. Further, since only edges involving v_{N+1} are considered, compared to all possible edges in the SG and WeSGRA learning algorithms, a much smaller and restricted space of edges are considered.

D. Learning r_{N+1}'s Capability

The learning algorithms of SG and WeSGRA have capability learning functions that learn the capabilities of all the robots in the synergy graph. The SG and WeSGRA algorithms differ in the techniques used to learn the robots' capabilities given the observations O and a synergy graph structure. In SG, O (the set of observations involving $r_1, \ldots, r_N$) contain all teams of size 2 and 3 are observed, with multiple observations per team. A Normal distribution per observed team is estimated using the data, and a matrix least-squares operation is used to solve for the means and variances of the robot capabilities [1]. In WesGRA, O contains samples of role assignments, and each observation in O forms a log-likelihood expression involving the means and variances of the robots in the team. A non-linear solver then solves for the robot capabilities using the expressions to maximize the log-likelihood [2].

Our LearnCapability capability function learns robot r_{N+1}'s capability using the graph structure of a WeSG and the observation set O_{N+1}. We assume that the capabilities of $r_1, \ldots, r_N$ are known, so the only unknowns are μ_{N+1} and σ^2_{N+1}, the mean and variance of r_{N+1}'s capability, i.e., $C_{N+1} \sim \mathcal{N}(\mu_{N+1}, \sigma^2_{N+1})$.

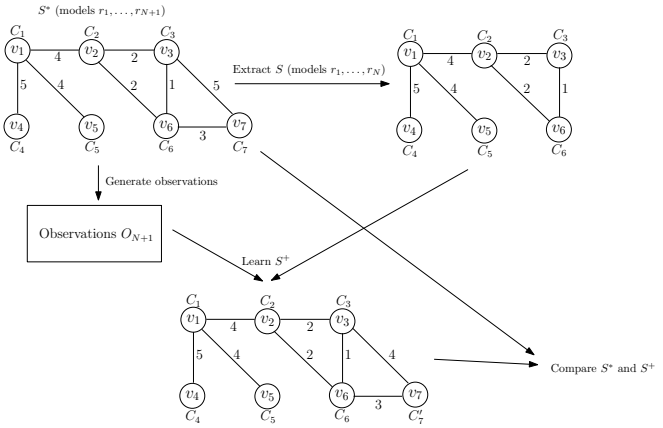

Fig. 4: The experimental process to compare the initial edge generation functions and capability learning methods.

We use two techniques to learn r_{N+1}'s capability, which are modified from the SG and WeSGRA algorithms. If O_{N+1} contains observations of all teams of size 2 and 3 involving r_{N+1}, i.e., $\forall o = (t, v) \in O_{N+1}, r_{N+1} \in t$, we use a matrix least-squares operation to solve for C_{N+1}. Otherwise, we form log-likelihood expressions and use a non-linear solver.

V. Analyzing the Learning Algorithm

In this section, we compare the three edge generation functions and two capability learning functions to analyze their characteristics and performance.

A. Experimental Setup

Fig. 4 shows the process of our experiments. We first randomly generate a WeSG S^* with $N+1$ robots, and label the vertices v_1 to v_{N+1} such that $\forall i$ s.t. $1 \leq i \leq N+1$, the subgraph with vertices $v_1, \ldots, v_i$ remains connected. In particular, the subgraph containing $v_1, \ldots, v_N$ forms the pre-existing WeSG to our learning algorithm (S in the input to Algorithm 1). Fig. 5 shows the four different graph structures of WeSGs created: a chain, loop, star, and a random structure (where edges are added probabilistically). For each graph structure type, we generate 50 WeSG models, and hence 200 WeSG models S^* are generated in total.

From the WeSG S^*, we generate the observation set O_{N+1}. For the experiments using the matrix least-squares robot capability learner, we use all pairs and triples of robot teams that contain r_{N+1}, i.e., $\forall o = (t, v) \in O_{N+1}$, $t = (r_{N+1}, r_i)$ or $t = (r_{N+1}, r_i, r_j)$. The value v is sampled from the synergy of t (Definition 3.2). For each team, 30 samples are generated so 30 different observations are present per team in O_{N+1} (there are $\mathcal{O}(N^2)$ teams). For the experiments using the non-linear solver, we generate 25 samples of team performances, i.e., $|O_{N+1}| = 25$. The matrix least-squares learner requires $\mathcal{O}(N^2)$ observations (every robot team of size 2 and 3 that contains r_{N+1}), while the non-linear solver runs with only 25 observations.

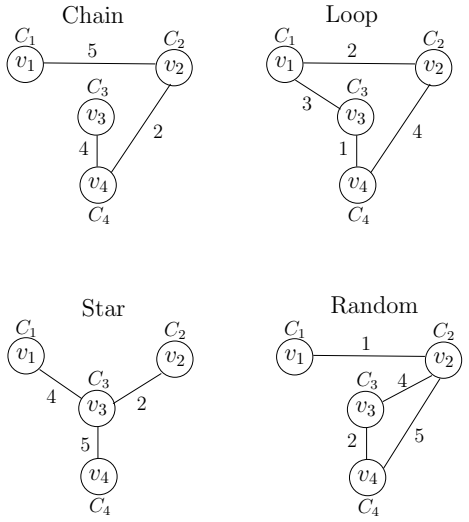

Fig. 5: Examples of the four WeSG structure types generated: chain, loop, star, and random.

Algorithm 1 then adds robot r_{N+1} into the WeSG to form S^+. To compare S^* and S^+, we use the KL-divergence of the synergy of possible teams. We compute the synergy (a Normally-distributed variable) of all teams containing r_{N+1}, i.e., $R \subset \mathcal{R}^+$ s.t. $|R| \geq 2$ and $r_{N+1} \in R$ and calculate the KL-divergence of the synergy using the learned S^+ from the actual S^*. The difference between S^* and S^+ is defined as the median KL-divergence of the synergies:

$$D(S^*, S^+) = \text{median}_R(D_{KL}(\mathbb{S}^*(R) \parallel \mathbb{S}^+(R))) \quad (3)$$

where $R \subset \mathcal{R}^+$ s.t. $|R| \geq 2$ and $r_{N+1} \in R$, and $\mathbb{S}^*$ and $\mathbb{S}^+$ are the synergy functions using S^* and S^+ respectively.

The learning algorithm performs simulated annealing for a fixed number of iterations, which may have an impact on the learned WeSG. Hence, we ran the algorithms for both 50 and 100 iterations of simulated annealing, to determine its effects. Also, there is a random element to the learning algorithm, and so we repeated the experiment 10 times per hidden S^*, for a total of 2000 trials (4 graph types, 50 WeSGs per type, 10 trials per WeSG) for each setup of the learning algorithm (capability learner and initial edge generation).

B. Comparison Results

Table I shows the results of these experiments. We set $N = 14$, so r_{N+1} is the 15^{th} robot in the WeSG. Across the capability learning functions, the average difference between the hidden WeSG and the learned WeSG generally improves as the number of iterations of simulated annealing goes from 50 to 100. However, when the difference is already small (e.g., with the Star structure type), more iterations do not improve the algorithm's performance. The least-squares capability solver learns WeSGs that have a smaller difference from the hidden WeSG across the four structure types, due to having much more data to learn from.

The three heuristics used for the initial edge generation have varying performance. The *most similar* heuristic performs the best, learning the closest WeSG across the capability learners and number of iterations. The *random* and *density* heuristic perform similarly, since the underlying algorithm is similar except for a different density p used.

WeSG Learner		WeSG Structure Type			
Capability Learner	Initial Edges	Chain	Loop	Star	Random
Least-squares (50 iterations)	Random	16.0	47.8	3.8	7.9
	Density	14.9	33.8	4.6	7.2
	Most Similar	9.4	21.8	2.2	7.5
Least-squares (100 iterations)	Random	14.4	32.4	3.3	5.8
	Density	14.8	31.6	2.9	8.7
	Most Similar	9.3	18.6	1.7	6.8
Non-linear (50 iterations)	Random	75.3	200.0	12.7	16.6
	Density	77.3	162.0	14.2	28.2
	Most Similar	40.9	97.1	14.4	17.7
Non-linear (100 iterations)	Random	91.2	74.0	13.9	24.2
	Density	69.9	135.2	15.8	15.5
	Most Similar	45.9	72.7	14.8	17.6

TABLE I: Average difference $D(S^*, S^+)$ between the hidden WeSG and learned WeSG given different hidden structure types and robot learning algorithms.

Between the four WeSG structure types, star structures were the easiest to learn, followed by random, while chain and loop have similar performance. The random structure type performs well due to the random search in the simulated annealing iterations, as seen from the improvement in performance between 50 and 100 iterations. The star structure has the best performance probably because the pairwise distances between robots do not change much with the addition/removal of new edges. In contrast, the chain and loop structures are more difficult to learn, since an extra edge can easily *disrupt* the structure and change the shortest distance between existing robots.

VI. Comparing Different Learning Approaches

In the previous section, we analyzed the different heuristics and capability learning functions of our learning algorithm. In this section, we compare the performance of our learning algorithm against the baseline of the SG and WeSGRA learning algorithms [1], [2].

The SG and WeSGRA learning algorithms assume that the observation set of all the robots $r_1, \ldots, r_{N+1}$ are available initially ($O^+ = O \cup O_{N+1}$), while our learning algorithm only requires the observations of r_{N+1} interacting with the other robots (O_{N+1}), but assumes the existence of a WeSG modeling $r_1, \ldots, r_N$ and adds r_{N+1} into the WeSG.

To compare these learning algorithms, we did the following: we first generate a hidden WeSG S^* with $N+1$ robots, and label the vertices v_1 to v_{N+1} such that $\forall i$ s.t. $1 \leq i \leq N+1$, the subgraph with vertices $v_1, \ldots, v_i$ remains connected, similar to the previous section. The observation sets O and O_{N+1} are then generated using S^*. We used 3 learning approaches to learn the WeSG. First, in *Completely Relearn*, the SG and WeSGRA learning algorithms are run with the complete observation set O^+. In *Learn N then Add Teammate*, we use the SG and WeSGRA learning algorithm to learn a WeSG of n robots (using the observation set O, then our new iterative learning algorithm to learn the $n+1$ robot's capability. In the third approach, *Completely Iterative*,

WeSG Learner	Number of Robots					
	5	6	7	8	9	10
Completely Relearn	1.7 ± 5.1	4.4 ± 9.3	7.8 ± 12.8	13.0 ± 15.3	15.1 ± 16.3	17.9 ± 18.6
Learn N then Add Teammate	19.2 ± 52.9	18.5 ± 28.7	22.3 ± 28.1	24.0 ± 25.0	33.0 ± 63.5	29.4 ± 28.1
Completely Iterative	28.2 ± 47.9	33.3 ± 37.5	33.4 ± 31.2	43.0 ± 36.2	48.8 ± 38.9	52.3 ± 44.3

TABLE II: Average difference between the hidden WeSG and learned WeSG using different learning methods.

we assume that the WeSG modeling r_1 and r_2 is given (the subgraph of S^* containing two vertices), and iteratively add r_3, r_4, and so on until r_{N+1}.

The learned WeSG models from the different learning approaches are compared to the hidden one S^* using the distance function in the previous section (Equation 3). We varied the number of robots from 5 to 10, and Table. II shows the results. *Completely Relearn* performs the best, as expected, with a low difference of 1.7 with 5 agents to 17.9 with 10 agents. The difference increases with the number of robots as the learning problem becomes more difficult. In comparison, *Learn N then Add Teammate* has a higher difference. However, the rate of increase in error is lower than completely relearning, which suggests that when n is large, *Learn N then Add Teammate* will perform comparably to *Completely Relearn*. Also, the runtime cost of *Completely Relearn* is much higher than that of *Learn N then Add Teammate*. The last approach, *Completely Iterative*, has much higher error than the other two approaches, which is due to the fact that errors accumulate as more robots are learned iteratively. Hence, it would be recommended to use *Completely Relearn* at certain intervals, so as to reset the accumulated errors, albeit at high runtime cost, and use iterative learning in small steps.

VII. Conclusion

We contributed a learning algorithm that uses new information relating to a new robot, and add a new vertex and corresponding edges to an existing synergy graph. Our algorithm creates an initial guess of edges, and iteratively improves the structure and learns the robot's capability through simulated annealing. We introduced three heuristics for generating the initial edges, and use two capability learning functions — a matrix least-squares method and a non-linear solver.

We compared the effectiveness of the three edge generation functions, and showed that the *most similar* heuristic, that replicates the edges of an existing robot, performs well across a variety of synergy graph structure types. We also compared the two capability learning functions, and showed that while the matrix least-squares solver performs better than the non-linear solver, it requires a much larger number of observations than the non-linear solver. We also compared three approaches to learning the robot capabilities, completely relearning, iteratively learning only the last robot, and iteratively learning all the robots. We showed that the error accumulates in iterative learning if multiple agents are learned, so completely relearning the robot capabilities and synergy at certain intervals is recommended to reset the error.

Incorporating new information is important in updating a learned model, and our learning algorithm allows synergy graphs to be applicable to a larger variety of domains. We performed our experiments in simulation, and our algorithm does not require any domain-dependent information. Hence, it can be run on data from real robots without any modifications. While the performance of the edge generation functions is domain-dependent, our simulations suggest that the *most similar* heuristic will perform well overall.

Acknowledgments

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] 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*, 2012, pp. 365–374.

[2] S. Liemhetcharat and M. Veloso, "Weighted Synergy Graphs for Role Assignment in Ad Hoc Heterogeneous Robot Teams," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5247–5254.

[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] J. Chen and D. Sun, "Resource Constrained Multirobot Task Allocation Based on Leader-Follower Coalition Methodology," *Journal of Robotics Research*, vol. 30, no. 12, pp. 1423–1434, 2011.

[5] 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.

[6] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohme, "Coalition Structure Generation with Worst Case Guarantees," *Journal of Artificial Intelligence*, vol. 111, pp. 209–238, 1999.

[7] L. Vig and J. Adams, "Market-based Multi-Robot Coalition Formation," in *Proceedings of the International Symposium on Distributed Autonomous Robotics Systems*, 2006, pp. 227–236.

[8] T. Michalak, D. Marciniak, M. Szamotulski, T. Rahwan, M. Wooldridge, P. McBurney, and N. Jennings, "A Logic-Based Representation for Coalitional Games with Externalities," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2010, pp. 125–132.

[9] B. Banerjee and L. Kraemer, "Coalition Structure Generation in Multi-Agent Systems with Mixed Externalities," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2010, pp. 175–182.