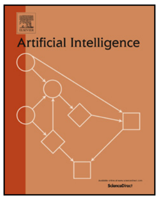
Artificial Intelligence 208 (2014) 41–65

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint



Weighted synergy graphs for effective team formation with heterogeneous ad hoc agents

Somchaya Liemhetcharat[*], Manuela Veloso

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

A R T I C L E   I N F O

Article history:
Received 12 September 2012
Received in revised form 12 December 2013
Accepted 17 December 2013
Available online 19 December 2013

Keywords:
Multi-agent
Multi-robot
Heterogeneous
Capability
Synergy
Ad hoc
Team formation

A B S T R A C T

Previous approaches to select agents to form a team rely on single-agent capabilities, and team performance is treated as a sum of such known capabilities. Motivated by complex team formation situations, we address the problem where both single-agent capabilities may not be known upfront, e.g., as in ad hoc teams, and where team performance goes beyond single-agent capabilities and depends on the specific *synergy* among agents. We formally introduce a novel weighted synergy graph model to capture new interactions among agents. Agents are represented as vertices in the graph, and their capabilities are represented as Normally-distributed variables. The edges of the weighted graph represent how well the agents work together, i.e., their synergy in a team. We contribute a learning algorithm that learns the weighted synergy graph using observations of performance of teams of only two and three agents. Further, we contribute two team formation algorithms, one that finds the optimal team in exponential time, and one that approximates the optimal team in polynomial time. We extensively evaluate our learning algorithm, and demonstrate the expressiveness of the weighted synergy graph in a variety of problems. We show our approach in a rich ad hoc team formation problem capturing a rescue domain, namely the RoboCup Rescue domain, where simulated robots rescue civilians and put out fires in a simulated urban disaster. We show that the weighted synergy graph outperforms a competing algorithm, thus illustrating the efficacy of our model and algorithms.

1. Introduction

Heterogeneous agents have varying capabilities that affect their task performance. We research on teams of such heterogeneous agents and how the performance of a team at a task relates to the composition of the team. Team performance has previously been computed as the sum of individual agent capabilities, e.g., the amount of resources an agent possesses [38,6]. In this work, we are interested in a model of team performance that goes beyond the sum of single-agent capabilities. We understand that there is *synergy* among the agents in the team, where team performance at a particular task depends not only on the individual agents' capabilities, but also on the composition of the team itself. Specific agents may have or acquire a high task-based relationship that allows them to perform better as a team than other agents with equivalent individual capabilities but a low task-based relationship. There are many illustrations of such synergy in real human teams, basically for any task. An example is an all-star sports team comprised of top players from around the

[*] Corresponding author. Somchaya Liemhetcharat has graduated from Carnegie Mellon University and is now at the Institute for Infocomm Research (I^2R), Singapore. Tel.: +65 6408 2000, fax: +65 6776 1378.

E-mail addresses: som@ri.cmu.edu (S. Liemhetcharat), veloso@cs.cmu.edu (M. Veloso).

0004-3702/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.artint.2013.12.002

world, hence individual agents with high capabilities, who may have a lower synergy as a team and perform worse than a well-trained team of individuals with lower capabilities but much higher synergy.

To model task-based relationships, we introduce a connected weighted graph structure, where the vertices represent the agents, and the edges represent the task-based relationships. In such graphs, we define the level of synergy of a set of agents, as a function of the shortest path between agents. We further devise a non-binary metric of team performance based on a Gaussian model of the individual agent capabilities. Such probabilistic variables allow us to capture the inherent variability in team performance in a dynamic world. We show that our formulation of team performance captures many interesting characteristics, such as the effects of including new agents into the team.

Most existing team formation approaches assume that the agent capabilities are known *a priori* (e.g., [48]). We are motivated by research in ad hoc agents, that learn to collaborate with previously unknown teammates [40]. An ad hoc team is one where the agents in the team have not collaborated with each other. Assuming an ad hoc team, we address the team synergy learning question as: given a set of agents with unknown capabilities, how do we model and learn the capabilities and synergy of the agents through observations, in order to form an effective team, i.e., a subset of the agents? A solution to this problem will enable ad hoc teams to be applied to a variety of problems in the real world, where effective teams need to be composed from agents who may not have previously worked together.

A motivating scenario is the urban search-and-rescue (USAR) domain. Many USAR robots have been developed by different research groups, with a variety of hardware capabilities. When a disaster occurs, researchers from around the world arrive with their USAR robots. Due to safety and space constraints, only a subset of these robots may be able to be deployed to the site. Since many of these researchers have not collaborated in the past, selecting an effective team is an ad hoc problem, where the agent capabilities and synergy are initially unknown. Some of these robots may have been designed to work well with other robots developed by the same group, and in some cases, robots from different sources may have synergy in a team, e.g., a robot that clears rubble quickly so that another robot can search. Thus, it is necessary to model and learn the synergy of these robots and select the best team of robots to be deployed.

We contribute a learning algorithm that uses only observations of the performance of teams of two and three agents, in order to learn the agent capabilities and weighted graph structure of the weighted synergy graph. The learning algorithm iterates through weighted graph structures, and computes the agent capabilities using the observations. We also contribute two team formation algorithms that uses the learned weighted synergy graph to find an effective team that solves the task. Our approach does not make many assumptions about the agents, only that observations of their performance is available, and as such our approach is applicable to many multi-agent domains.

We perform extensive experiments to demonstrate that our learning algorithm effectively learns the structure of representative graph types and agent capabilities. We compare the weighted synergy graph to the unweighted synergy graph that we previously introduced [26], and demonstrate that the weighted synergy graph is more expressive and hence applicable to more domains. We apply the weighted synergy graph model to the RoboCup Rescue domain (that simulates rescue robots in a USAR scenario), and show that the learned weighted synergy graph is used to form a near-optimal team, and outperforms IQ-ASyMTRe [48], a competing algorithm.

In summary, the contributions of this work are:

1. A novel model of multi-agent team performance, the weighted synergy graph model, where agents are vertices in a connected weighted graph, edges represent how well agents work together, and agent capabilities are Normally-distributed variables;
2. The definition of the synergy of a multi-robot team as a function of the weighted synergy graph model;
3. A team formation algorithm that forms the optimal team in exponential time;
4. A team formation algorithm that approximates the optimal team in polynomial time;
5. A learning algorithm that learns a weighted synergy graph using only observations of agent teams comprising two and three agents;
6. Extensive experiments that evaluate our model and algorithms using synthetic data;
7. Application of the weighted synergy graph model to the RoboCup Rescue domain.

The article is organized as follows: Section 2 discusses related research in multi-robot task allocation, coalition formation, team formation, and ad hoc teams, and how it compares to our work. In Section 3, we formally define the weighted synergy graph model and our team formation algorithms. Section 4 contributes the synergy graph learning algorithm, while Section 5 presents extensive learning experiments. Section 6 compares the expressiveness of the weighted and unweighted synergy graph models. Section 7 details the experiments in the RoboCup Rescue domain, and Section 8 draws conclusions.

2. Related work

This section presents a review of related work, discussing the relevant domains of task allocation, coalition formation, ad hoc coordination and team formation.

2.1. Multi-robot task allocation

Multi-robot task allocation (MRTA) is focused on the problem of allocating a set of tasks to a group of robots so as to maximize a utility function. The MRTA problem is categorized along three axes: single-task robots (ST) versus multi-task robots (MT), single-robot tasks (SR) versus multi-robot tasks (MR), and instantaneous assignment (IA) versus time-extended assignment (TA) [17].

Multi-robot teams have been used for many tasks, such as in the RoboCup Rescue Simulation League [20]. A city is simulated to have had a disaster, necessitating rescue robots to help to mitigate the crisis. Civilians are located around the city that have to be rescued and/or directed to safe zones, and fires that break out in the city have to be put out. The scenario is a MRTA problem (specifically ST-MR-TA), and the goal is to assign the rescue agents to different tasks in the city effectively. The tasks appear over time, and require varying durations to be completed. The rescue robots are allocated to tasks, and re-assigned new tasks as the old tasks are completed. There are a variety of methods to assign tasks to robots, such as using a biologically-inspired approach [12], using coalition formation with spatial and temporal constraints [35], modeling the problem as a generalized allocation problem with task interdependencies [13], and developing predictive models [21]. We use the RoboCup Rescue simulator and existing algorithms designed for it as a test-bed for our weighted synergy graph model. We view the RoboCup Rescue problem as a team formation problem, where existing MRTA algorithms are selected before the simulation starts (each MRTA algorithm controls a subset of the rescue robots during the simulation), and the goal is to form the optimal combination of MRTA algorithms to achieve maximum utility.

Representing heterogeneous capabilities

The capabilities of heterogeneous robots have been represented in different ways. One technique defines utility as the difference between the quality of the task execution and the expected resource cost [17]. The utility of a robot performing a task is quantified, and is general and does not impose any restrictions on the utility function. However, this generality is a double-edged sword and does not impose any structure on the tasks that would allow them to be grouped.

Another technique is the resource model, where a set of resources is first defined, e.g., robot arm, camera, sonar [6]. Then, each robot is defined as a vector that contains the number of each resource a robot has, e.g., 2 arms, 1 camera and 0 sonar. In this model, resources are either completely interchangeable (i.e., a robot arm on one robot is equivalent to a robot arm on another) [38,44,39], or with a level of quality [6]. In these resource models, tasks are defined as a list of required resources, e.g., a task requires 2 robot arms, 3 cameras, and 1 sonar, and in the latter model, a minimum quality of each resource is also defined.

The service model of capabilities [38,46] is similar to the resource model, with the following distinctions: a robot is either able or unable to performance a service, i.e., there is no quantity or quality of a single service a robot possesses. Tasks are defined as a list of the number of each type of service required (similar to the resource model), but in the task allocation process, each robot can only perform a single service, instead of providing all the services it is capable of. The authors state that while resources can be exchanged in multi-agent systems, resources on multi-robot systems are usually physical and cannot be exchanged. Having a minimum number of resources does not necessarily imply task completion because other constraints such as proximity have to be represented and met. The resource model can still be applied with the addition of constraints, but the service model abstracts and represents these constraints succinctly.

Robot capabilities are also described by schemas, i.e., inputs and outputs of information types, and robots are defined as sets of schemas: the sensors of the robots provide outputs without any inputs, and other schemas are perceptual (e.g., vision, localization), motor (e.g., controlling actuators), and communication [33,43]. The task is defined as a set of desired outputs, and a team of robots is capable of completing the task if a joint plan exists that produces the outputs by chaining the schemas in the robots.

We have previously introduced the concept of mutual state capabilities, where a robot's capability in the task depends on its teammate and their joint state [25]. Such a model of capabilities captures the notion of synergy among robots, which we further elaborate in this work.

Evaluating task allocations

There are two main methods to quantify the performance of an allocation of tasks to robots. The first method, task-based performance, defines a utility gained by completing each task. The second method, team-based performance, defines the quality of performance of each robot performing the task allocated to it.

Task-based performance is used in market-based techniques [8,10] and other approaches [7]. Each task is associated with a reward, which is computed based on domain-specific factors, such as the amount of unexplored space [49] or number of items picked up [9]. In market-based techniques, the cost of performing the task is used to form the bid, and an auctioneer assigns tasks based on robots' bids [43]. The final performance of the task allocation is then computed based on the profit, i.e., the difference between the utility gained from the tasks and the costs incurred [9]. In some approaches, only the sum of utilities gained is used to calculate performance, and the costs are only used for the allocation process [45,43], and in others the goal is to complete all the subtasks while minimizing the total cost [31]. The benefit of task-based performance is that the utility gained from completing a task is independent of how the task is completed — after the allocation, each task is either completed or not completed, and there is no measure of how well a particular task was done, other than the costs incurred by the robots assigned to it.

In team-based performance, the quality of a completed task varies depending on the robots allocated to it, e.g., R_1 may complete task T_1 with a lower quality than if R_2 completed T_1. The performance of the task allocation is then the difference between the quality of completed tasks and the costs incurred by the robots. With this formulation, the ST-SR-IA problem can be posed as an optimal assignment problem and solved in polynomial time, or with a market-based approach [17]. When each task requires more than one robot to complete, it becomes a ST-MR-IA problem, and has many similarities with the coalition formation problem, which we describe next. Team-based performance measures how well a task was completed, so the composition of a team has a larger impact beyond the costs.

2.2. Coalition formation

Coalition formation involves the partitioning of a set of agents A into disjoint subsets so as to maximize the overall utility. In characteristic function games, the value of each possible subset is given by a characteristic function, and the goal is to find the optimal coalition structure to as to maximize the sum of the characteristic function of each coalition. The number of coalition structures is $O(|A|^{|A|})$ and $\omega(|A|^{|A|/2})$, so enumerating possible coalition structures to find the optimal is intractable [37]. Coalition formation is applicable to MRTA [38] in domains such as deciding flight targets for UAVs [16].

In characteristic function games, the value of a coalition depends only on the agents within it. There has been recent work in coalition formation with externalities, where a coalition's value depends on the structure of other coalitions. A logic-based representation is used for coalition formation with externalities, that is fully expressive and at least as concise as regular coalition formation representations [32]. Positive and negative externalities are considered separately, where PF^{+}_{sub} means weakly sub-additive, so merging two coalitions decreases their joint value (or keeps it constant) while increasing the values of other coalitions in the structure (or keeps them constant), and PF^{-}_{sup} means weakly super-additive, where merging a coalition increases its value and decreases the values of other coalitions (or keeps values constant) [34].

Mixed externalities are also considered, where both positive and negative effects can occur [2]. Agents are defined with a set of types, and agents of some types as competitors, and the value of a coalition structure improves if they are in singleton coalitions. Conversely, agents of the remaining types are collaborators, and the value improves if all of them are in a single large coalition. Using this formulation, the authors use a branch and bound algorithm to find the best coalition structure with guaranteed worst-case bounds.

Externalities in coalitions is an interesting area because it considers how the values of coalitions are computed. In regular coalition formation, the characteristic function defines the values of coalitions, and the function is assumed to be general and unknown, so little work has been done to analyze possible structures in the characteristic function.

2.3. Ad hoc teams

The ad hoc problem was recently introduced, and the goal is "to create an autonomous agent that is able to efficiently and robustly collaborate with previously unknown teammates on tasks to which they are all individually capable of contributing as team members" [40]. An example of an ad hoc problem is with two robots — one that is pre-programmed to follow a certain policy, and another that has to adapt its behavior so that the pair is jointly optimal without explicit communication. The pre-programmed robot and the ad hoc robot can be viewed as the teacher and learner in a multi-armed bandit problem [42,3]. Similarly, a two-player game-theoretic setting is used to study how an ad hoc agent can vary its actions so as to maximize the payoff with a best-response teammate that has varying amounts of memory of previous interactions [41]. The work has been extended to situations where a single ad hoc agent leads multiple teammates in selecting the optimal joint action [1].

Role assignment in an ad hoc team is considered, where an ad hoc agent has to select a role, such that it maximizes the team's overall utility based on its observations of its teammates [15]. An ad hoc agent in the pursuit domain has to vary its behavior to better suit the team's objective of capturing the target, by modeling its teammates and choosing a best response [4]. In the case where the system state and joint action is fully observable, but the model of teammates is unknown, biased adaptive play can be used by an ad hoc agent to optimize the joint action of the team [47].

Locker-room agreements, i.e., policies agreed upon by the team prior to evaluation, can be used to coordinate robots without additional communication. Robots can estimate the state of a teammate in order to decide which robot should approach the ball in the robot soccer domain, and while the robots agree on the policy $> 90\%$ of the time, communication is necessary for situations where errors in state estimation may cause policy fluctuations in the team [18].

We are interested in the ad hoc team formation problem, where the agents in the team have not collaborated with each other. The capabilities of the agents and how well they coordinate at the task are initially unknown, and the goal is to form an effective ad hoc team by selecting relevant agents to form the team.

2.4. Team formation

Team formation is focused on the problem of selecting the best subset of agents that can complete a task. In the ASyMTRe model, each robot is defined with schemas and the team is selected by composing feasible teams through planning, and then selecting the optimal team using a heuristic [33]. The algorithm has been extended to form teams that complete multiple tasks, using both team formation and a market-based task allocation algorithm [43]. IQ-ASyMTRe is a recent extension to ASyMTRe that handles information quality [48], and we compare our proposed model to IQ-ASyMTRe.

Graphs can be used to represent relationships among agents, where a subgraph of connected agents are selected to complete a task [14]. Similarly, by using social networks of agents, where agents have different skills, and edge weights represent communication costs, the optimal team to complete the task has to cover all the required skills [22,23], or trade off between skills and connectivity [11]. The edges in a social network graph can also be used as constraints, where an agent is assigned a task, and must find teammates that are directly connected to it [7], or form a connected sub-network [5].

We recently introduced the synergy graph model, where agents are vertices in an unweighted connected graph, and agent capabilities are modeled as Normally-distributed variables [26]. In this work, we formally define the weighted synergy graph model, where weighted edges are used in the graph. We show that our model is more expressive while having the same runtime cost, and thus it can be applied to more domains. In addition, we perform extensive experiments to demonstrate the efficacy of our learning algorithm, and also apply the weighted synergy graph model to the RoboCup Rescue domain.

2.5. How our work fits

Our goal is to form effective ad hoc teams, through observations of the agents' performance in the task and modeling the synergistic effects among agents in the team. We build upon the related research, and focus on the following:

- First, we model team-based performance based on the agent's individual capabilities, and the synergistic effects among members of the team, and not a sum of single-agent capabilities. Compared to the MRTA problem, we are interested in forming a single team for a task, and not the allocation of multiple agents to multiple tasks.
- Second, coalition formation focuses on how to partition a set of agents to maximize a value function, while we are interested in modeling the value function, based on observations of the agents at the task.
- Third, research in the ad hoc domain has so far focused on how a single ad hoc agent can adapt to its teammates in order to improve task performance. We are interested in forming an effective ad hoc multi-agent team.

3. Team formation at a task

In this section, we formally define the team formation problem and our weighted synergy graph model that models the synergistic effects of agents working together in a team, and contribute our team formation algorithms that use the weighted synergy graph model to form an effective team for the task.

3.1. Formally defining the team formation problem

We begin with the definition of the set of agents and the definition of a team:

Definition 3.1. The **set of agents** is $\mathcal{A} = \{a_1, \ldots, a_N\}$, where each $a_n \in \mathcal{A}$ is an agent.

Definition 3.2. A **team** is any subset $A \subseteq \mathcal{A}$.

There is a task to be performed that can be accomplished with *any* number of agents with varying performance, and hence any subset of $\mathcal{A}$ is a valid team. The performance of a team is the utility attained by that team when performing the task, and is domain-dependent. In a dynamic world, the performance of teams of agents is non-deterministic, so multiple observations of the same team at the task may result in different values:

Definition 3.3. The **performance** of a team $A \subseteq \mathcal{A}$ is $\mathcal{P}_A$ and is non-deterministic.

Definition 3.4. An **observation** o_A is a real value corresponding to an observed utility attained by the team $A \subseteq \mathcal{A}$, i.e., o_A is a sample of $\mathcal{P}_A$.

For example, suppose that a disaster has occurred in an urban area (such as a city), and that multiple urban search-and-rescue (USAR) personnel have arrived on the scene to offer their aid. $\mathcal{A}$ is the set of all USAR personnel, and the task is saving lives and minimizing damage to the city. Suppose $A_0 \subseteq \mathcal{A}$ is a USAR team that performs the task. The performance of the team is measured and forms the observation $o_{A_0} = 3.4$. However, due to the dynamic nature of the USAR task (for example, wind causing fires to spread), the observed performance of A_0 may be different if the task was repeated, i.e., o_{A_0} would be a different number each time A_0 performed the task.

Since the performance of a team is non-deterministic, we define the δ-optimal team:

Definition 3.5. The **δ-optimal team** is the team $A^*_\delta \subseteq \mathcal{A}$ such that there exists some utility u where A^*_δ obtains a utility of at least u with probability δ, and the probability of any other team A doing so is at most δ:

$$\mathbb{P}(\mathcal{P}_{A^*_\delta} \geqslant u) = \delta \quad \text{and} \quad \mathbb{P}(\mathcal{P}_A \geqslant u) \leqslant \delta \quad \forall A \subseteq \mathcal{A}$$

The goal is to find the δ-optimal team of agents $A^*_\delta \subseteq \mathcal{A}$, and we assume that δ is given as part of the domain information. The δ-optimality measure was designed in order to rank non-deterministic performance; when performance is deterministic (or only the mean is considered), comparing the performance of teams is done with the $\geqslant$ operator. In δ-optimality, δ determines whether a risky team or risk-averse team is preferred. For example, when $\delta = \frac{1}{2}$, only the mean performance is considered, and the δ-optimal team is equivalent to the optimal team with non-deterministic performance. When $\delta < \frac{1}{2}$, a high-risk, high-reward team is preferred, i.e., one that has a low probability of attaining a high performance. Conversely, when $\delta > \frac{1}{2}$, a low-risk, low-reward team is preferred, i.e., one that has a high probability of attaining a low performance.

3.2. Modeling task-based relationships

In order to find the δ-optimal team A^*_δ, we want to create a model of how well agents work together at the task. In the social networks domain, social graphs are used for team formation, where an edge between a pair of agents indicates that the agents have a social relationship, and the weight of the edge indicates the communication cost between them [22,11]. We are interested in forming teams that perform well in a task, and hence we model the task-based relationships among the agents as a task-based graph, where agents are vertices in the graph and edges represent the task-based relationships.

One possible approach to the task-based graph is to use a fully-connected graph, where the weight of an edge indicates the **cost** of 2 agents working together. Fig. 1a shows an example of a fully-connected task-based graph with 4 agents. The agent a_1 works better with a_2 than a_3, as indicated by the lower edge weight of 1 between a_1 and a_2, compared with an edge weight of 4 between a_1 and a_3.

A fully-connected task-based graph has a major drawback — the task-based relationships between pairs of agents are completely independent, since every pair of agents is connected by an edge whose weight can be arbitrary. Using the notion that edge weights represent the cost of agents working together, we introduce the concept of transitivity in task-based relationships. For example, if agent a_1 works very well with a_2, and a_2 works very well with a_3, then a_1 will work well with a_3. The transitivity occurs because for a_1 to work very well with a_2, there should be some underlying coordination strategy, and similarly between a_2 and a_3. Assuming that the agents use the same algorithms regardless of partners (i.e., they do not switch strategies), then a_1 and a_3 will be able to work well together since there is some overlap in their coordination strategies with a_2, albeit with higher cost. We assume that agents are always able to coordinate (e.g., all agents use the same communication protocol), or performance is based on their joint actions (similar to game theory), so any pair of agents has some task-based relationship.

To capture this notion of task-based transitivity, we use a connected graph where the shortest distance between agents indicates the cost of them working together. Fig. 1b shows a connected task-based graph, by modifying the graph in Fig. 1a such that edges $\{a_1, a_3\}$, $\{a_1, a_4\}$, and $\{a_2, a_4\}$ have been removed. However, the shortest distance between agents are equal in both Fig. 1a and Fig. 1b, and thus both graphs express the same task-based relationships.

While the shortest distance between agents in the task-based graph represents the cost of agents working together, we want to explicitly model the task-based relationship. Thus, we introduce a compatibility function $\phi : \mathbb{R}^+ \to \mathbb{R}^+$, where $\phi(d(a_i, a_j))$ returns the task-based compatibility between agents a_i and a_j, and $d(a_i, a_j)$ is the shortest distance between them in the task-based graph. ϕ is a monotonically-decreasing function, so larger distances correspond to lower compatibility. Like the value function V, ϕ is domain-specific, and two intuitive examples of ϕ are:

$$\phi_{\text{fraction}}(d) = \frac{1}{d}$$
$$\phi_{\text{decay}}(d) = \exp\left(-\frac{d \ln 2}{h}\right)$$

where ϕ_{fraction} is a fraction function, and ϕ_{decay} is an exponential decay function with half-life h.

In [26], we assumed that the edges in the task-based graph were unweighted (having a weight of 1), and that the compatibility function would be adjusted to capture the task-based relationship among agents. However, there are many

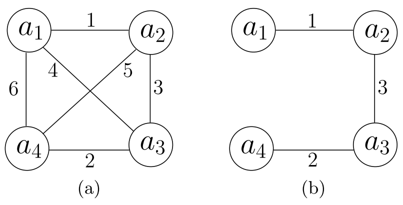


Fig. 1. a) A fully-connected task-based graph with 4 agents, where the edge weights represent the cost of agents working together to perform the task. b) A connected task-based graph with the same 4 agents, where some edges have been removed while still preserving the pairwise distances between agents.

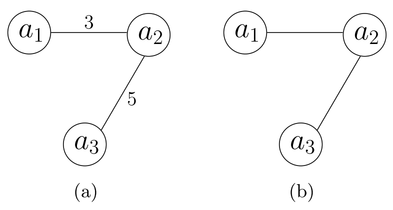


Fig. 2. a) A weighted task-based graph with 3 agents. b) An unweighted task-based graph with the 3 agents (all edges have a weight of 1). If the compatibility function in (a) is $\phi(d) = \frac{1}{d}$, the task-based relationships of the agents cannot be represented with an unweighted graph and an adjusted compatibility function.

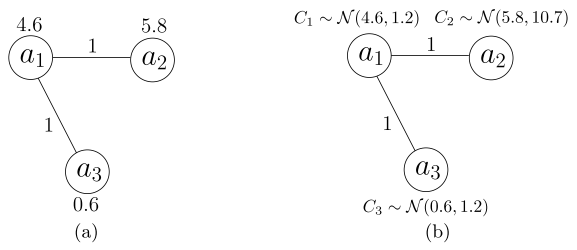

4.6
5.8
a_1
1
a_2
1
a_3
0.6
(a)
$C_1 \sim \mathcal{N}(4.6, 1.2)$
$C_2 \sim \mathcal{N}(5.8, 10.7)$
a_1
1
a_2
1
a_3
$C_3 \sim \mathcal{N}(0.6, 1.2)$
(b)

Fig. 3. Weighted task-based graphs with 3 agents, where the agents' heterogeneous capabilities are attached to each vertex and are represented as a) values; b) Normally-distributed variables.

combinations of weighted edges such that an adjusted compatibility function cannot capture the same relationship with unweighted edges.

Fig. 2a shows one such example of a task-based graph that cannot be represented with unweighted edges. Suppose $\phi(d) = \frac{1}{d}$, the compatibility of $\{a_1, a_2\}$, $\{a_1, a_3\}$, and $\{a_2, a_3\}$ are then $\frac{1}{3}$, $\frac{1}{8}$ and $\frac{1}{5}$ respectively. Suppose we use an unweighted task-based graph and find an adjusted compatibility function ϕ', such that $\phi'(d'(a_i, a_j)) = \phi(d(a_i, a_j))$, where d' and d are the shortest distance functions in the unweighted and weighted task-based graphs respectively. In an unweighted task-based graph, the longest possible distance between any pair agents is $|\mathcal{A}| - 1$, if all the agents formed a chain. Thus, with 3 agents, the greatest distance is 2. Since the compatibility function is monotonically decreasing, $\phi'(2) = \frac{1}{8}$, which is the lowest compatibility among a_1, a_2, and a_3, and implies that $d'(a_1, a_3) = 2$. However, this in turn implies that $d'(a_1, a_2) = d'(a_2, a_3) = 1$ (Fig. 2b). As such, no compatibility function ϕ' can be defined, since $\phi'(1)$ has to be equal to $\frac{1}{3}$ and $\frac{1}{5}$ which is impossible.

Thus, in this work, we use weighted edges in the task-based graph to represent how well agents perform as a team.

3.3. *Representing agent capabilities*

The task-based graph and compatibility function described above model how well agents work together at a task. However, heterogeneous agents have different capabilities that affect their performance at a task. The performance of a team of agents then depends on the capabilities of the agents and their task-based relationship.

One method to represent heterogeneous agent capabilities is to assign a value μ_i for each agent a_i, where μ_i corresponds to the agent a_i's mean capability at the task. In this work, we view capability as a measure of an agent's contribution to the team performance at a task, and not a binary (capable/incapable). As such, the mean capability refers to the average performance the agent contributes to the task.

Fig. 3a shows an example of 3 agents a_1, a_2, a_3, where a_1 works equally well with agents a_2 and a_3, i.e., $d(a_1, a_2) = d(a_1, d_3)$. Even though a_1 works equally well with them, a_2 is has a higher mean capability than a_3. As such, the mean performance of team $\{a_1, a_2\}$ is greater than $\{a_1, a_3\}$.

While values model the agents' mean capabilities at the task, it does not capture variability. Since the agents act in a dynamic, non-deterministic world, their performance varies over multiple instances. Thus, instead of a single values, we use a Normally-distributed variable to represent an agent's capability, i.e., each agent a_i is associated with a variable C_i, which is the agent a_i's non-deterministic capability at the task. We use a Normal distribution because it is unimodal, corresponding to the agent's performance with a peak value, and variability as its deviation. Also, Normal distributions are widely used for their mathematical properties, which we exploit later.

By using a Normal variable instead of a single value, we can now model how consistent an agent is at the task. Fig. 3b shows a modification of Fig. 3a, where a_2 has a higher variance for its performance compared to a_3. As such, depending on δ, the team $\{a_1, a_3\}$ may outperform $\{a_1, a_2\}$.

3.4. *Defining the weighted synergy graph*

We have detailed how task-based relationships are represented with the compatibility function ϕ, which uses the distance between agent vertices in a graph, and how agent capabilities are represented as Normally-distributed variables. In this section, we formally define the weighted synergy graph and how it is used to compute the performance of a team of agents at a task.

Definition 3.6. The **weighted synergy graph** is a tuple (G, C), where:

- $G = (V, E)$ is a connected weighted graph,
- $V = \mathcal{A}$, i.e., the set of vertices corresponds to the set of agents,
- $e_{i,j} = (a_i, a_j, w_{i,j}) \in E$ is an edge between agents a_i, a_j with weight $w_{i,j} \in \mathbb{R}^+$,
- $C = \{C_1, \ldots, C_N\}$, where $C_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ is agent a_i's capability at the task.

Weighted synergy graphs are connected, so at least one path exists between any pair of vertices. The distance $d(a_i, a_j)$ between any two agents a_i, a_j is defined to be the shortest distance between them in the graph.

Using the weighted synergy graph, we quantify the performance of a pair of agents:

Definition 3.7. The **pairwise synergy** $\mathbb{S}_2(a_i, a_j)$ between two agents a_i, a_j in a weighted synergy graph is $C_{i,j} = \phi(d(a_i, a_j)) \cdot (C_i + C_j)$, where $d(a_i, a_j)$ is the shortest distance between a_i and a_j in the weighted synergy graph, and $\phi : \mathbb{R}^+ \to \mathbb{R}^+$ is the compatibility function.

We assume that C_i and C_j are independent for all i, j, and so the summation $C_i + C_j$ in the pairwise synergy function can be performed easily. This assumption is reasonable as the effect of agents working in a team is captured by the compatibility function and their distance in the weighted synergy graph, so the variables representing their individual capabilities (C_i, C_j) are independent.

The pairwise synergy function between any two agents always exists, since we assume that the weighted graph is connected (there is always a shortest distance in the graph between any pair of agents), and that the task can be accomplished with any number of agents (Section 3.1).

We use the graph structure to model the synergy among agents, specifically using the edges (that connect two agents) to compute the shortest distance between pairs of agents, and hence the pairwise synergy is the building block for the synergy of a team of agents. Using the pairwise synergy function, we now define the performance of a team of agents:

Definition 3.8. The **synergy** $\mathbb{S}(A)$ of a set of agents $A \subseteq \mathcal{A}$ in a weighted synergy graph is the average of the pairwise synergy of its components, i.e., $\frac{1}{\binom{|A|}{2}} \cdot \sum_{\{a_i, a_j\} \in A} \mathbb{S}_2(a_i, a_j)$.

Using the definitions above, the synergy of a team A is a Normally-distributed random variable $C_A \sim \mathcal{N}(\mu_A, \sigma_A^2)$. In particular:

$$\mu_A = \frac{1}{\binom{|A|}{2}} \sum_{a_i, a_j \in A} \phi\big(d(a_i, a_j)\big) \cdot (\mu_i + \mu_j) \tag{3.1}$$

$$\sigma_A^2 = \frac{1}{\binom{|A|}{2}^2} \sum_{a_i, a_j \in A} \phi\big(d(a_i, a_j)\big)^2 \cdot \big(\sigma_i^2 + \sigma_j^2\big) \tag{3.2}$$

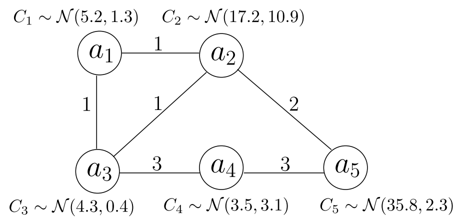
The synergy of a team $A \subseteq \mathcal{A}$ depends critically on the capabilities of the agents in A, and the pairwise distances between them. We illustrate how synergy changes based on the team composition below.

3.4.1. *A weighted synergy graph example*

While the synergy function $\mathbb{S}$ is linear and takes the average of pairwise synergy, many interesting agent relationships are modeled. Fig. 4 shows an example of a weighted synergy graph with five agents in a rescue task.

a_1, a_2 and a_3 are personnel in the ambulance, namely the agent that carries the stretcher, the agent that performs CPR, and the agent that drives the ambulance respectively. Since these three agents have trained as a team, their task-based relationship is very high, and so the distance between them in the weighted synergy graph is very low (a distance of 1 between any pair of the three). As heterogeneous agents, their individual capabilities are different. For example, the capability of a_2, the CPR agent, has a high mean (that reflects the high payoff CPR provides to the task) and correspondingly high variance (that reflects that CPR does not always succeed). As each agent is individually capable of saving lives (i.e., the driver agent and stretcher agent can also perform first aid), the task can be completed with *any* subset of agents.

a_4 is an intern at the hospital, and a_5 is a surgeon there. The surgeon's capability is both better and more consistent than the CPR agent's: $C_5 \sim \mathcal{N}(35.7, 3.3)$ versus $C_2 \sim \mathcal{N}(20.3, 10.9)$, reflecting the surgeon's skills. The surgeon has a high



$C_1 \sim \mathcal{N}(5.2, 1.3)$
$C_2 \sim \mathcal{N}(17.2, 10.9)$
a_1
a_2
a_3
a_4
a_5
1
1
1
2
3
3
$C_3 \sim \mathcal{N}(4.3, 0.4)$
$C_4 \sim \mathcal{N}(3.5, 3.1)$
$C_5 \sim \mathcal{N}(35.8, 2.3)$

Fig. 4. An example of a weighted synergy graph modeling capabilities and the task-based relationships of a group of agents in a rescue task.

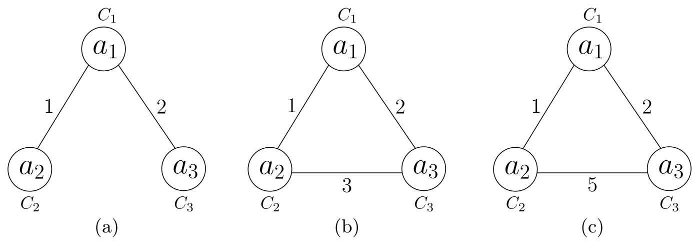


Fig. 5. Three equivalent weighted synergy graphs, i.e., the shortest distance between pairs of agents is equivalent in the three graphs.

task-based relationship with the CPR agent (reflected by a distance of 2 in the graph) since both agents frequently work together, but is not as high as within the personnel in the ambulance (i.e., a_1, a_2 and a_3) that have distances of 1 with one another. The intern, a_4, has the lowest mean capability and second-highest variance, reflecting its poor skill at the task. Further, the intern has only worked with the surgeon (on operations in the hospital) and the ambulance driver (to coordinate the arrival of patients). As such, the intern has edges with weights of 3 with both the surgeon a_5 and the driver a_3. Since the intern has no direct edges with the other agents (e.g., the CPR agent a_2), the task-based relationship between them will be transitive. For example, the task-based relationship of a_2 and a_4 uses a distance of 4 (1 from the edge $e_{\{a_2,a_3\}}$ and 3 from the edge $e_{\{a_3,a_4\}}$). The transitivity reflects that the intern works less well with the CPR agent than the driver agent, by the distance of edge $e_{\{a_2,a_3\}}$.

Using Definitions 3.7 and 3.8, and assuming that the compatibility function $\phi(d) = \frac{1}{d}$, we can compute the synergy of the agents. The team $\{a_1, a_2, a_3\}$ (the agents in the ambulance) has a synergy of $C_{\{a_1,a_2,a_3\}} \sim \mathcal{N}(17.8, 2.8)$. Since the surgeon a_5 is more capable than a_2 (the CPR agent), one might consider replacing a_2 with a_5. However, the synergy would be $C_{\{a_1,a_3,a_5\}} \sim \mathcal{N}(12.1, 0.3)$, which has a lower mean than the team $\{a_1, a_2, a_3\}$. The lower mean is due to the task-based relationships among the agents — the other agents in the ambulance (the stretcher agent and the driver agent) work well and perform better with the CPR agent than with the surgeon.

When these four agents are in a team, their synergy is $C_{\{a_1,a_2,a_3,a_5\}} \sim \mathcal{N}(17.8, 0.8)$, which shows that the addition of the surgeon agent (instead of replacing the CPR agent) potentially benefits the team by lowering the variance of their performance. However, adding agents does not always improve the team performance — the team consisting of all the agents has a synergy $C_{\{a_1,a_2,a_3,a_4,a_5\}} \sim \mathcal{N}(13.0, 0.3)$, since the intern agent has a low capability and a poor task-based relationship with the other agents, as reflected by its high edge weights to other agents.

Thus, the weighted synergy graph captures interesting and complex relationships among agents, where the composition of the team makes a significant difference in the overall task performance.

3.4.2. Equivalence in weighted synergy graphs

Fig. 5 shows three examples of weighted synergy graphs with 3 agents, such that $d(a_1, a_2) = 1$, $d(a_1, a_3) = 2$, and $d(a_2, a_3) = 3$. In Fig. 5a, only 2 edges are present while 3 edges are present in Figs. 5b and 5c. In particular, the edge $e_{2,3}$ of Fig. 5c is not used, since a shorter path exists between a_2 and a_3 using edges $e_{1,2}$ and $e_{1,3}$. Thus, in general, more than one graph structure can be used to define the distance relationship among agents.

Definition 3.9. Weighted synergy graphs $S = (G, C)$ and $S' = (G', C')$ are **equivalent** if:

- $V = V'$,
- $C = C'$,
- $d(a_i, a_j)$ in graph $G = d(a_i, a_j)$ in graph G' $\forall a_i, a_j \in \mathcal{A}$.

Since the shortest distance between all pairs of agents are identical, their compatibility (as computed by ϕ) is identical, and hence the synergy of a team of agents are equal given equivalent weighted synergy graphs. In this work, we do not distinguish between equivalent weighted synergy graphs.

3.5. Assumptions of the weighted synergy graph model

The weighted synergy graph model defined above captures the task-based relationships among agents and is used to compute the synergy of agent teams using the distances in the weighted graph and the agent capabilities. We now list the assumptions of the model, and a short discussion of the rationales behind the assumptions and how the assumptions can be relaxed:

1. Team performance is Normally-distributed;
2. The synergy of a team is a function of the pairwise synergies;
3. Task-based relationships are transitive and modeled via shortest distance in the graph.

Regarding Assumption 1, Section 3.3 explains the rationale of using Normal distributions to model team performance. If the actual performance was from some other distribution, the weighted synergy graph model can be modified to use that distribution or any arbitrary distribution, although the computation of pairwise synergy $\mathbb{S}_2$ and synergy $\mathbb{S}$ could be more complex (by adding arbitrary distributions together).

Assumption 2 comes about from the graph-based nature of the weighted synergy graph model. Since edges involve two agents, we derive the synergy function $\mathbb{S}$ from the pairwise synergy $\mathbb{S}_2$. Further, the task can be accomplished by any number of agents, so pairwise synergy between any two agents always exists. If the assumption is relaxed, the synergy graph can be extended to use hyperedges (edges between more than two agents), and $\mathbb{S}$ can be updated to reflect the change.

The weighted synergy graph model assumes transitivity in the task-based relationship (Assumption 3). Our work focuses on tasks where this assumption holds — we believe that this is the case for many tasks involving software agents (for example in the RoboCup Rescue domain, which we elaborate later). In cases where the assumption does not hold, the model can be updated to use other measures other than the shortest distance between agents, and is the focus of our future work.

3.6. Solving the team formation problem

We defer how a weighted synergy graph is learned from observations of performance to the next section. In this section, we explain how to use a weighted synergy graph to form the δ-optimal team for the task, i.e., the team A^*_δ such that $\mathbb{P}(\mathcal{P}_{A^*_\delta} \geqslant u) = \delta$ and $\mathbb{P}(\mathcal{P}_A \geqslant u) \leqslant \delta \ \forall A \subseteq \mathcal{A}$.

Using the weighted synergy graph model and the synergy equations, we can compute the synergy of a team of agents $A \subseteq \mathcal{A}$. However, the synergy computed is a Normally-distributed variable, and we need to rank such variables to choose one possible team over another.

To do so, we use an evaluation function (the V function introduced by us in [24] that we rename to `Evaluate` here) that converts a Normally-distributed variable into a real number using a risk factor $\rho \in (0, 1)$:

$$\texttt{Evaluate}(X, \rho) = \mu_X + \sigma_X \cdot \Phi^{-1}(\rho) \tag{3.3}$$

where $X \sim \mathcal{N}(\mu_X, \sigma_X^2)$ and Φ^{-1} is the inverse of the cumulative distribution function of the standard Normal distribution.

In particular, for any Normally-distributed variable X, $\texttt{Evaluate}(X, 1-\delta)$ returns a value v_X such that $P(X \geqslant v_X) = \delta$.

Theorem 3.10. *Let $A, A' \subseteq \mathcal{A}$, and $C_A = \mathbb{S}(A)$, $C_{A'} = \mathbb{S}(A')$. Let $\texttt{Evaluate}(C_A, 1-\delta) = v_A$ and $\texttt{Evaluate}(C_{A'}, 1-\delta) = v_{A'}$. If $v_A \geqslant v_{A'}$, then $P(C_{A'} \geqslant v_A) \leqslant \delta$.*

Proof. $P(C_{A'} \geqslant v'_A) = \delta$ and $v_A \geqslant v'_A \Rightarrow P(C_{A'} \geqslant v_A) \leqslant P(C_{A'} \geqslant v_{A'}) \Rightarrow P(C_{A'} \geqslant v_A) \leqslant \delta$. □

Corollary 3.11. $A^*_\delta = \operatorname{argmax}_{A \subseteq \mathcal{A}} \texttt{Evaluate}(\mathbb{S}(A), 1-\delta)$.

Proof. Let $A_\delta = \operatorname{argmax}_{A \subseteq \mathcal{A}} \texttt{Evaluate}(\mathbb{S}(A), 1-\delta)$. Let $\texttt{Evaluate}(\mathbb{S}(A_\delta), 1-\delta) = v$. $\forall A \subseteq \mathcal{A}$, $P(\mathbb{S}(A) \geqslant v) \leqslant \delta$ (from Theorem 3.10) $\therefore A^*_\delta = A_\delta$. □

`Evaluate` returns a real number that we use to rank possible teams and find the δ-optimal team A^*_δ, since the team that returns the highest value from `Evaluate` corresponds to the δ-optimal team. We now contribute two team formation algorithms: `Find`δ`OptimalTeam`, a branch-and-bound algorithm that finds the δ-optimal team in exponential time, and `Approx`δ`OptimalTeam`, that approximates that δ-optimal team in polynomial time. Both algorithms assume that $n = |A^*_\delta|$ is known and given as a parameter to the algorithm. If n is unknown, then the algorithms are run iteratively for $n = 1, \ldots, N$ and the best-performing team is selected.

3.6.1. Finding the δ-optimal team

Our first team formation algorithm, `Find`δ`OptimalTeam`, uses branch-and-bound to find the δ-optimal team. Algorithm 1 shows the pseudo-code of the algorithm. The inputs to the algorithm are n (the size of the desired team),

Algorithm 1: Find the δ-optimal team of size n

```
Find$\delta$OptimalTeam($n, \delta, S, A, A_{\text{best}}, v_{\text{best}}$)
1: if $|A| = n$ then
2:     $v_A \leftarrow$ Evaluate($\mathbb{S}(A), 1-\delta$)
3:     if $v_A \geqslant v_{\max}$ then
4:         return ($A, v_A$)
5:     else
6:         return ($A_{\text{best}}, v_{\text{best}}$)
7:     end if
8: end if
9: $n \leftarrow \max_{a_i \in A}(i)$
10: for $i = n+1$ to $N$ do
11:     $A_{\text{next}} \leftarrow A \cup \{a_i\}$
12:     $(\text{next}_{\min}, \text{next}_{\max}) \leftarrow$ CalculateBounds($n, \delta, S, A_{\text{next}}$)
13:     if $\text{next}_{\max} > v_{\text{best}}$ then
14:         $v_{\text{best}} \leftarrow \max(v_{\text{best}}, \text{next}_{\min})$
15:         $(A_{\text{best}}, v_{\text{best}}) \leftarrow$ Find$\delta$OptimalTeam($n, \delta, S, A_{\text{next}}, A_{\text{best}}, v_{\text{best}}$)
16:     end if
17: end for
18: return ($A_{\text{best}}, v_{\text{best}}$)
```

δ (for δ-optimality), S (the weighted synergy graph), A (the team being considered), A_{best} (the best team found so far), and v_{best} (the value of the best team found so far). $|A| \leqslant n$ during the execution of the algorithm, and contains the fixed members of the team, e.g., if $n = 5$ and $A = \{a_1, a_2\}$, then FindδOptimalTeam returns the optimal team of 5 agents given that $\{a_1, a_2\}$ are in the team. The initial call to FindδOptimalTeam sets $A = A_{\text{best}} = \emptyset$ and $v_{\text{best}} = -\infty$.

Lines 1–8 describe the base case when the team is fully-formed — the value of the team is computed with Evaluate, and the team is returned if its value is greater than the current best team. Otherwise, branching and bounding is performed. The branching occurs as agents are added to A and the algorithm is recursively called (lines 11 and 15). The bounds of the team are computed with CalculateBounds (described below), and the team is pruned if the maximum of the bound ($\text{next}_{\max}$) is lower than the current best value v_{best} (line 13).

To compute the bounds of a team given a synergy graph, CalculateBounds uses the following heuristic. The minimum and maximum pairwise distance between vertices in the weighted synergy graph (excluding pairs of the selected team A_{next}) are computed, as well as the maximum and minimum agent capabilities (excluding the agents in A_{next}). The maximum bound is computed by using the synergy function $\mathbb{S}$ and assuming that all distances with undetermined agents are the minimum distance, and agent capabilities are maximum. Similarly, the minimum bound is computed using the maximum distances and minimum agent capabilities.

Theorem 3.12. *Finding the δ-optimal team is NP-hard.*

Proof. We reduce the max-clique problem, which is NP-complete [19], to the weighted synergy graph team formation problem.

Suppose that $G = (V, E)$ is an unweighted graph, and the goal is to determine if a clique of at least size n exists, i.e., a subgraph of n vertices that is fully connected.

We define a connected weighted graph $G' = (V, E')$ where $\forall e = (v, v') \in E, \exists e' = (v, v', 1) \in E'$. Further, we add edges of weight 2 such that the graph G' is connected. The number and identity of these edges are unimportant.

Using the graph G', we create a weighted synergy graph $S = (G', C)$ such that all $C_i \sim \mathcal{N}(1, 1) \in C$.

We define the compatibility function as: $\phi(d) = \begin{cases} 1 & \text{if } d \leqslant 1 \\ 0 & \text{otherwise} \end{cases}$, and define $\delta = \frac{1}{2}$, i.e., only the means of the capabilities matter.

$$\begin{aligned}
\mathbb{S}(A) &= \frac{1}{\binom{|A|}{2}} \sum_{a, a' \in A} \mathbb{S}_2(a, a') \quad \text{(from Eq. (3.8))} \\
&= \frac{1}{\binom{|A|}{2}} \sum_{a, a' \in A} \phi\big(d(a, a')\big)(C_a + C_{a'}) \quad \text{(from Eq. (3.7))} \\
&= \frac{1}{\binom{|A|}{2}} \sum_{a, a' \in A} 2\phi\big(d(a, a')\big) \quad \left(\text{since } \delta = \frac{1}{2} \text{ and } C_a = C_{a'} \sim \mathcal{N}(1, 1)\right) \\
&= \frac{1}{\binom{|A|}{2}} \sum_{a, a' \in A \text{ s.t. } d(a, a') = 1} 2 \quad \big(\text{since } \phi(d) = 1 \text{ if } d \leqslant 1 \text{ and } 0 \text{ otherwise}\big) \\
&= \frac{1}{\binom{|A|}{2}} 2\big|\big\{e' = (a, a', 1) \in E' \text{ s.t. } a, a' \in A\big\}\big|
\end{aligned}$$

Algorithm 2: Approximate the δ-optimal team of size n

```
Approx$\delta$OptimalTeam($n$, $\delta$, $S$)
 1: $A_{\text{best}} \leftarrow$ RandomTeam($S, n$)
 2: $v_{\text{best}} \leftarrow$ Evaluate($\mathbb{S}(A_{\text{best}}), 1-\delta$)
 3: repeat
 4:   $A_{\text{neighbor}} \leftarrow$ NeighborTeam($A_{\text{best}}$)
 5:   $v_{\text{neighbor}} \leftarrow$ Evaluate($\mathbb{S}(A_{\text{neighbor}}), 1-\delta$)
 6:   if accept($v_{\text{best}}, v_{\text{neighbor}}$) then
 7:     $A_{\text{best}} \leftarrow A_{\text{neighbor}}$
 8:     $v_{\text{best}} \leftarrow v_{\text{neighbor}}$
 9:   end if
10: until done()
11: return $A_{\text{best}}$
```

Hence, a clique in G of size n corresponds to a having a synergy of $\frac{1}{\binom{n}{2}} 2\binom{n}{2} = 2$, since a clique in G corresponds to a clique in G' that is fully connected with edges of weight 1.

Further, a clique has maximum synergy (compared to other subsets of size n) since it has the maximum number of edges with weight 1.

Thus, determining if the δ-optimal team of size n has synergy 2 is at least as hard as determining if a clique of n exists, and so finding the δ-optimal team is NP-hard. □

Finding the δ-optimal team is NP-hard, and branch-and-bound can fully explore the space in the worst case. As such, the runtime of FindδOptimalTeam is $O(N^n)$. If n is unknown, then the algorithm is run for increasing n for a total runtime of $O(N^N)$.

3.6.2. Approximating the δ-optimal team

Finding the δ-optimal team takes exponential time in the worst case, and in many situations, a near-optimal team is sufficient to solve the problem. Algorithm 2 shows the pseudo-code for ApproxδOptimalTeam, that approximates the δ-optimal team of size n, given δ and a weighted synergy graph S.

Algorithm 2 first begins by generating a random team of size n, which is performed by randomly selecting n agents from $\mathcal{A}$. The value of the team is then computed with the Evaluate function. The random team and its value thus forms the initial guess of the algorithm.

Next, the algorithm begins its approximation loop. Lines 3–10 of Algorithm 2 show a general approximation algorithm (with functions accept and done) to illustrate that our algorithm is compatible with most approximation algorithms. In this work, we use simulated annealing but other approximation algorithms (such as hill-climbing) are suitable as well. In simulated annealing, the done function would check if the desired number of iterations has been run, and the accept function would accept a neighbor based on the temperature schedule (computed from the current iteration number) and the difference in Evaluate scores of the current best guess and its neighbor.

NeighborTeam(A_{best}) randomly swaps one selected agent $a \in A_{\text{best}}$ with an unselected one $a' \in \mathcal{A} \setminus A_{\text{best}}$. In this way, neighbor teams are generated from the current best estimate A_{best} so as to effectively explore the space of possible teams of size n. The value of the neighbor team v_{neighbor} is computed with Evaluate, and the team is accepted or rejected based on the criteria of the approximation algorithm (e.g., simulated annealing uses a temperature schedule).

Thus, Algorithm 2 finds an approximation to the δ-optimal team given its size n. The algorithm runs in $O(n^2)$ (the synergy function $\mathbb{S}$ takes $O(n^2)$ and simulated annealing runs a constant number of iterations) if n is known. Otherwise, the algorithm is run iteratively for increasing n and has total runtime of $O(N^3)$. In comparison, a brute-force algorithm would take $O(\binom{N}{n})$ if n is known, and $O(2^N)$ otherwise.

3.6.3. Comparing the team formation algorithms

To evaluate both team formation algorithms, and compare their performance (amount of the search space explored, and value of the formed team), we generated random weighted synergy graphs. We varied the number of agents in the graph from 10 to 15, and randomly created 1000 connected weighted graph structures where each edge weight varied from 1 to 5. For each weighted graph structure generated, the agent capabilities $C_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ were also randomly generated, such that $\mu_i \in (50, 150)$ and $\sigma_i^2 \in (0, 100^2)$. The size of the desired team was set to $\lfloor \frac{N}{2} \rfloor$, so that the search space is as large as possible.

FindδOptimalTeam always finds the optimal team, so we were interested in evaluating how many times CalculateBounds and Evaluate were called. For ApproxδOptimalTeam, we ran simulated annealing for 1000 iterations (so 1000 calls to Evaluate were made), and we were interested in evaluating the quality of the team formed.

Table 1 shows the results of our comparisons. The effectiveness of the formed team is expressed as a value in [0, 1], where 0 means the worst possible team (with the minimum value), and 1 means the optimal team (with the maximum value):

Table 1
The number of evaluations done by FindδOptimalTeam and ApproxδOptimalTeam to compute and approximate the δ-optimal team respectively, and the quality of the team found (where 0 means the worst team and 1 is the optimal team).

# agents	FindδOptimalTeam		ApproxδOptimalTeam	
	# evaluations	Team effectiveness	# evaluations	Team effectiveness
10	340 ± 69	1	1000	0.996 ± 0.021
11	545 ± 125	1	1000	0.992 ± 0.034
12	1216 ± 254	1	1000	0.997 ± 0.020
13	1993 ± 449	1	1000	0.996 ± 0.021
14	4439 ± 932	1	1000	0.998 ± 0.010
15	7307 ± 1694	1	1000	0.998 ± 0.013

$$\text{Effectiveness}(A) = \frac{\texttt{Evaluate}(\mathbb{S}(A), 1-\delta) - \texttt{Evaluate}(\mathbb{S}(A_\delta^{\min}), 1-\delta)}{\texttt{Evaluate}(\mathbb{S}(A_\delta^*), 1-\delta) - \texttt{Evaluate}(\mathbb{S}(A_\delta^{\min}), 1-\delta)} \tag{3.4}$$

FindδOptimalTeam finds the optimal team but evaluates a large number of agent teams — the number of calls to CalculateBounds and Evaluate are greater than the size of the search space. In comparison, running ApproxδOptimalTeam for a fixed number of iterations (1000) finds the δ-optimal team or a team very close to optimal most of the time. When there are 12 or more agents, ApproxδOptimalTeam performs competitively with FindδOptimalTeam while evaluating a smaller amount of teams (only 1000). We believe the high performance of FindδOptimalTeam is because the neighbor generation allows for good exploration of the space — a team with a high score will remain with a high score when a single member is swapped. Further, the agent capabilities were uniformly sampled within a range and distances were randomly generated; ApproxδOptimalTeam may not perform as well if there are outliers in the team (e.g., an agent with extremely high capabilities and low pairwise distances). Comparatively, in FindδOptimalTeam, the bounds of performance are computed when some agents in the team is fixed; the bounds are large when few agents are fixed and only become narrow as most agents are fixed. As such, pruning can only occur towards the bottom of the branch-and-bound search tree and so a large search space is required. Thus, we use ApproxδOptimalTeam for the later parts of this work.

4. Learning the weighted synergy graph

We have formally defined the weighted synergy graph and our team formation algorithm to approximate the δ-optimal team. However, the team formation algorithm assumes the existence of a weighted synergy graph. In this section, we contribute our learning algorithm that learns a weighted synergy graph using only observations of the performance of teams of agents in $\mathcal{A}$.

Let $\mathcal{A}_2 \subset 2^{\mathcal{A}}$ such that $\mathcal{A}_2 = \bigcup_{A \in \mathcal{A} \text{ s.t. } |A|=2} \{A\}$. Similarly, let $\mathcal{A}_3 \subset 2^{\mathcal{A}}$ such that $\mathcal{A}_3 = \bigcup_{A \in \mathcal{A} \text{ s.t. } |A|=3} \{A\}$. Hence, $\mathcal{A}_2$ and $\mathcal{A}_3$ are the sets of all pairs and triples of agents respectively. Our learning algorithm uses the observations of the agents in $\mathcal{A}_{2,3} = \mathcal{A}_2 \cup \mathcal{A}_3$. Specifically, let O be the set of observations, where $\forall A \in \mathcal{A}_{2,3}$, $\exists o_{A,1}, \ldots, o_{A,M}$ such that each $o_{A,m}$ is an observation of the performance of the team A at the task. Since any subset of agents will attain a performance value at the task, and the synergy function $\mathbb{S}$ is computed from the pairwise synergy function $\mathbb{S}_2$, the observation set O is sufficient for learning. In particular, $\mathcal{A}_2$ provides information about the shortest distance between pairs of agents and the agents' capabilities using the pairwise synergy function $\mathbb{S}_2$ (Definition 3.7). However, there are multiple solutions for any pairwise synergy (increasing capabilities versus decreasing distances), and $\mathcal{A}_3$ provides information about the overall structure of the graph using the synergy function $\mathbb{S}$ (Definition 3.8), and provides additional constraints to the learning problem.

Algorithm 3 shows the pseudo-code of our learning algorithm. The algorithm first generates a random weighted graph structure with $N = |\mathcal{A}|$ vertices, and the function LearnCapabilities estimates the capabilities of the agents using the weighted graph structure and observation set O. We elaborate on LearnCapabilities later in this section. The weighted graph structure and estimated capabilities then form an initial guess S of the weighted synergy graph, and the log-likelihood of the observations in O given S is computed with LogLikelihood.

4.1. Learning the weighted synergy graph structure

The learning algorithm iteratively improves the learned weighted synergy graph: using the current estimate of the weighted synergy graph, the function NeighborStructure generates a neighbor weighted graph structure by using the current graph structure and performing one of four possible actions:

1. Increase the weight of a random edge by 1
2. Decrease the weight of a random edge by 1
3. Add an edge of random weight between two vertices
4. Remove a random edge that does not disconnect the graph

Algorithm 3: Learn a weighted synergy graph

```
LearnWeightedSynergyGraph($O$)
1:  $G = (V, E) \leftarrow$ RandomStructure($\mathcal{A}$)
2:  $C \leftarrow$ LearnCapabilities($G, O$)
3:  $S \leftarrow (G, C)$
4:  score $\leftarrow$ LogLikelihood($S, O$)
5:  repeat
6:      $G' = (V, E') \leftarrow$ NeighborStructure($G$)
7:      $C' \leftarrow$ LearnCapabilities($G', O$)
8:      $S' \leftarrow (G', C')$
9:      score$' \leftarrow$ LogLikelihood($S', O$)
10:     if accept(score, score$'$) then
11:         $S \leftarrow S'$
12:         score $\leftarrow$ score$'$
13:     end if
14: until done()
15: $S \leftarrow$ PruneEdges($S$)
16: return $S$
```

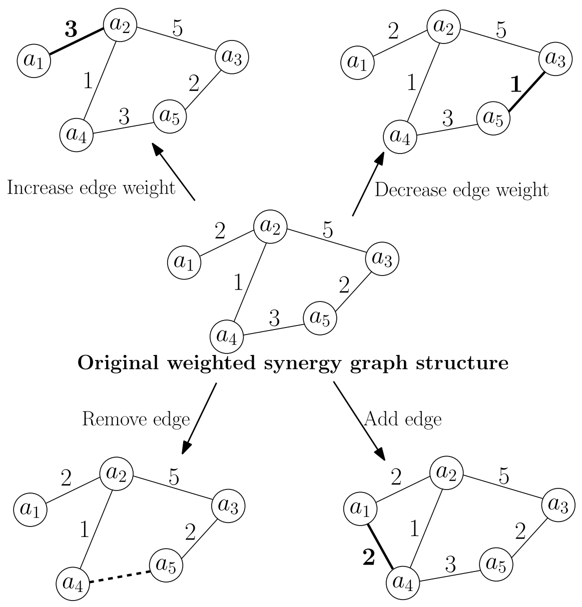


Fig. 6. The four possible actions used to generate neighbor weighted graph structures for the learning algorithm.

Fig. 6 illustrates these four actions. While the edge weights in the weighted synergy graph definition are real numbers, `NeighborStructure` only generates structures with integer weights. We use integer weights in the learning algorithm, as they provide a close approximation while still allowing discrete steps in neighbor generation. Higher accuracy can be achieved by increasing/decreasing the edge weights with smaller step size values. However, decreasing the step size increases the space of possible weighted graphs, so it is a trade-off that has to be managed.

4.2. Learning the agent capabilities

`LearnCapabilities` uses the generated weighted synergy graph structure and the observation set O to learn the capabilities of the agents. Algorithm 4 shows how the capabilities are learned.

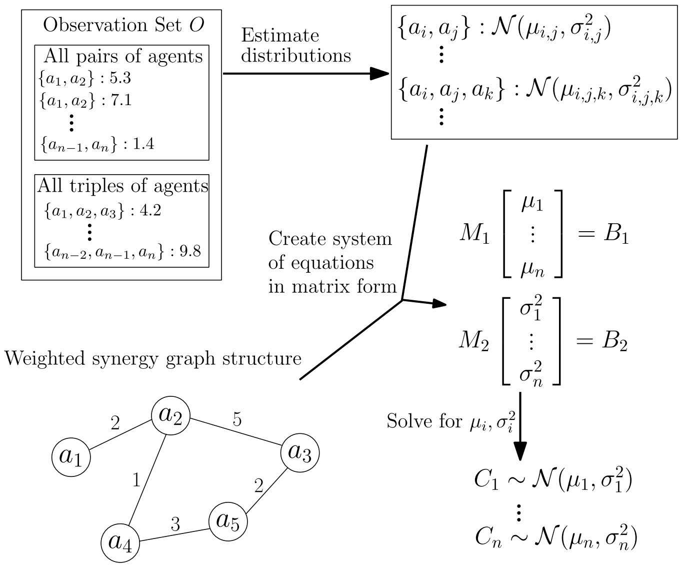
First, distributions are estimated from the observation set O. For every team $A \in \mathcal{A}_{2,3}$, there are m observations $o_{A,1}, \ldots, o_{A,m}$ of the team's performance. An unbiased estimator then estimates the distribution $\mathcal{N}(\mu, \sigma^2)$ that represents A's performance and forms D where:

$$\forall A \in \mathcal{A}_{2,3}, \quad \exists \big(A, D_A \sim \mathcal{N}\big(\mu_A, \sigma_A^2\big)\big) \in D \tag{4.1}$$

Next, the matrices M_1, M_2, B_1, B_2 are initialized as zeros. The matrices are used to form the matrix equations $M_1 X_1 = B_1$ and $M_2 X_2 = B_2$, where $X_1 = [\mu_1, \ldots, \mu_N]^T$ and $X_2 = [\sigma_1^2, \ldots, \sigma_N^2]^T$ are the means and variances of the agent capabilities respectively.

Algorithm 4: Learn capabilities from weighted synergy graph structure

```
LearnCapabilities($G$, $O$)
1: $D \leftarrow$ EstimateDistributions($O$)
2: $M_1 \leftarrow [0]_{|D|\times N}$
3: $M_2 \leftarrow [0]_{|D|\times N}$
4: $B_1 \leftarrow [0]_{|D|\times 1}$
5: $B_2 \leftarrow [0]_{|D|\times 1}$
6: $\alpha \leftarrow 0$
7: for all $(A_2 = \{a_i, a_j\}, \mathcal{N}(\mu_{A_2}, \sigma^2_{A_2})) \in D$ do
8:     $\alpha \leftarrow \alpha + 1$
9:     $d_{i,j} \leftarrow$ distance($a_i, a_j, G$)
10:    $M_1(\alpha, i) \leftarrow \phi(d_{i,j})$
11:    $M_1(\alpha, j) \leftarrow \phi(d_{i,j})$
12:    $M_2(\alpha, i) \leftarrow \phi(d_{i,j})^2$
13:    $M_2(\alpha, j) \leftarrow \phi(d_{i,j})^2$
14:    $B_1(\alpha) \leftarrow \mu_{A_2}$
15:    $B_2(\alpha) \leftarrow \sigma^2_{A_2}$
16: end for
17: for all $(A_3 = \{a_i, a_j, a_k\}, \mathcal{N}(\mu_{A_3}, \sigma^2_{A_3})) \in D$ do
18:    $\alpha \leftarrow \alpha + 1$
19:    $d_{i,j} \leftarrow$ distance($a_i, a_j, G$)
20:    $d_{i,k} \leftarrow$ distance($a_i, a_k, G$)
21:    $d_{j,k} \leftarrow$ distance($a_j, a_k, G$)
22:    $M_1(\alpha, i) \leftarrow \frac{1}{3}(\phi(d_{i,j}) + \phi(d_{i,k}))$
23:    $M_1(\alpha, j) \leftarrow \frac{1}{3}(\phi(d_{i,j}) + \phi(d_{j,k}))$
24:    $M_1(\alpha, k) \leftarrow \frac{1}{3}(\phi(d_{i,k}) + \phi(d_{j,k}))$
25:    $M_2(\alpha, i) \leftarrow \frac{1}{9}(\phi(d_{i,j})^2 + \phi(d_{i,k})^2)$
26:    $M_2(\alpha, j) \leftarrow \frac{1}{9}(\phi(d_{i,j})^2 + \phi(d_{j,k})^2)$
27:    $M_2(\alpha, k) \leftarrow \frac{1}{9}(\phi(d_{i,k})^2 + \phi(d_{j,k})^2)$
28:    $B_1(\alpha) \leftarrow \mu_{A_3}$
29:    $B_2(\alpha) \leftarrow \sigma^2_{A_3}$
30: end for
31: $(\mu_1, \ldots, \mu_N) \leftarrow$ LeastSquares($M_1, B_1$)
32: $(\sigma^2_1, \ldots, \sigma^2_N) \leftarrow$ LeastSquares($M_2, B_2$)
33: $C \leftarrow (C_1 \sim \mathcal{N}(\mu_1, \sigma^2_1), \ldots, C_N \sim \mathcal{N}(\mu_N, \sigma^2_N))$
34: return $C$
```



Observation Set O
All pairs of agents
$\{a_1, a_2\}$: 5.3
$\{a_1, a_2\}$: 7.1
$\{a_{n-1}, a_n\}$: 1.4
All triples of agents
$\{a_1, a_2, a_3\}$: 4.2
$\{a_{n-2}, a_{n-1}, a_n\}$: 9.8
Estimate distributions
$\{a_i, a_j\} : \mathcal{N}(\mu_{i,j}, \sigma^2_{i,j})$
$\{a_i, a_j, a_k\} : \mathcal{N}(\mu_{i,j,k}, \sigma^2_{i,j,k})$
Create system of equations in matrix form
$M_1 [\mu_1 \ldots \mu_n] = B_1$
$M_2 [\sigma^2_1 \ldots \sigma^2_n] = B_2$
Weighted synergy graph structure
Solve for μ_i, σ^2_i
$C_1 \sim \mathcal{N}(\mu_1, \sigma^2_1)$
$C_n \sim \mathcal{N}(\mu_n, \sigma^2_n)$
a_1
a_2
a_3
a_4
a_5
2
5
1
2
3

Fig. 7. The capabilities of agents are learned from the observation set and a weighted synergy graph structure.

Lines 7–30 of Algorithm 4 show how the matrices are filled in, where the counter α keeps track of the row number of the matrices. The distance between agents are computed using the weighted graph structure G, and used by the compatibility function ϕ. Lines 10–13 and 22–27 use the synergy equations in Definitions 3.7 and 3.8 to compute the contribution of each agent.

For example, Fig. 7 shows an example weighted synergy graph structure, and the process of Algorithm 4. We will use the example weighted synergy graph and the team $\{a_1, a_2\}$ to explain how the matrices are set. Suppose the compatibility function is $\phi(d) = \frac{1}{d}$. Using Definition 3.7, the team $\{a_1, a_2\}$ forms the equation $\frac{1}{2}(C_1 + C_2)$. Furthermore, the observations involving $\{a_1, a_2\}$ in the observation set are used to estimate the distribution $\mathcal{N}(\mu_{\{a_1,a_2\}}, \sigma^2_{\{a_1,a_2\}})$, and two equations are generated:

$$\frac{1}{2}(\mu_1 + \mu_2) = \mu_{\{a_1,a_2\}}$$
$$\frac{1}{2^2}\left(\sigma_1^2 + \sigma_2^2\right) = \sigma^2_{\{a_1,a_2\}}$$

These two equations are separately formed and evaluated because the distributions $C_i \in C$ are independent. Since the distance between a_1 and a_2 is 2, $d_{i,j} = 2$ in Algorithm 4 (Line 9), and $\phi(d_{i,j}) = \frac{1}{2}$, the values of columns 1 and 2 of M_1 are set to $\frac{1}{2}$ (Lines 10–11). Similarly, Lines 12–13 sets columns 1 and 2 of M_2 to be $\phi(d_{i,j})^2 = \frac{1}{2^2}$ using the second equation. The values of the corresponding rows of B_1 and B_2 are set to be $\mu_{\{a_1,a_2\}}$ and $\sigma^2_{\{a_1,a_2\}}$ respectively (Lines 14–15). A similar process occurs for observations involving three agents using the synergy function $\mathbb{S}$.

Once M_1, M_2, B_1, and B_2 have been filled in by iterating through all teams with two and three agents and filling in the matrices, a least-squares solver is used to solve for the means and variances of the agent capabilities. These then form the agent capabilities $C = \{C_1, \ldots, C_N\}$ that are returned by the function.

Algorithm 4 exploits the fact that the agent capabilities are Normally-distributed and mutually independent. If the capabilities were from a different distribution, then a different technique, such as solving log-likelihood expressions, may be required.

4.3. Calculating the log-likelihood and accepting neighbor candidates

The weighted synergy graph structure and learned capabilities are combined to form a weighted synergy graph. The function `LogLikelihood` computes the sum of log-likelihood of every observation $o \in O$ given a synergy graph S. Every observation is $o = (A, p)$ where $A \subseteq \mathcal{A}$ is an agent team, and $p \in \mathbb{R}$ is the observed performance of the team. Using the synergy graph S and the synergy function $\mathbb{S}$, the distribution of A's performance ($\mathcal{N}(\mu_A, \sigma_A^2)$) is computed using Definition 3.8. The log-likelihood of o is defined as the log-likelihood of p in this distribution, i.e., $\log(\frac{1}{\sigma_A\sqrt{2\pi}} \exp(\frac{(p-\mu_A)^2}{2\sigma_A^2}))$. The sum of log-likelihood of the observations computed by `LogLikelihood` gives a measure of how closely a synergy graph matches the observations, where a higher sum log-likelihood indicates a better match.

The score of the current best weighted synergy graph is compared to the score of the neighbor weighted synergy graph, and accepted by on the approximation algorithm used. For example, when simulated annealing is used, the difference in the scores is compared to the temperature schedule.

Hence, our learning algorithm iteratively improves the weighted synergy graph structure and learns the agent capabilities that best match the observations given.

5. Evaluating the learning algorithm

In this section, we describe the extensive experiments that show the efficacy of our learning algorithm. We first generate weighted synergy graphs that are hidden from the learning algorithm. The observation set O used for training, i.e., the performance of teams of two and three agents, is extracted from the hidden model, and then used to learn a new weighted synergy graph, which we compare against the hidden one. For each team of two or three agents, 30 observations are generated, following the distribution of their performance in the hidden model.

In order to quantitatively measure how well the learning algorithm performs, we used the log-likelihood of data given the weighted synergy graph. During training, the log-likelihood of the training examples is used to determine whether or not to accept a neighbor weighted synergy graph. In addition, a set of test data is generated, that consists of the performance of teams of four or more agents. Thus, the test data contains information that is never seen or used by the learning algorithm. We measured the log-likelihood of the test data given the learned weighted synergy graph in each iteration of simulated annealing. However, because each trial uses a randomly generated hidden graph, log-likelihood values vary from trial to trial. Thus, we scaled the log-likelihood numbers to be between 0 and 1, where 0 is the log-likelihood of the initial guess, and 1 means that the log-likelihood of the learned weighted synergy graph matches that of the hidden one used to generate the data.

5.1. Learning representative graph structures

In our first set of experiments, the hidden weighted synergy graphs were of three representative structure types — chain, loop, and star. The learning algorithm does not know the structure type of the hidden weighted synergy graph, and instead starts with a random graph structure. Fig. 8 shows the hidden, initial and final weighted synergy graphs for each of the

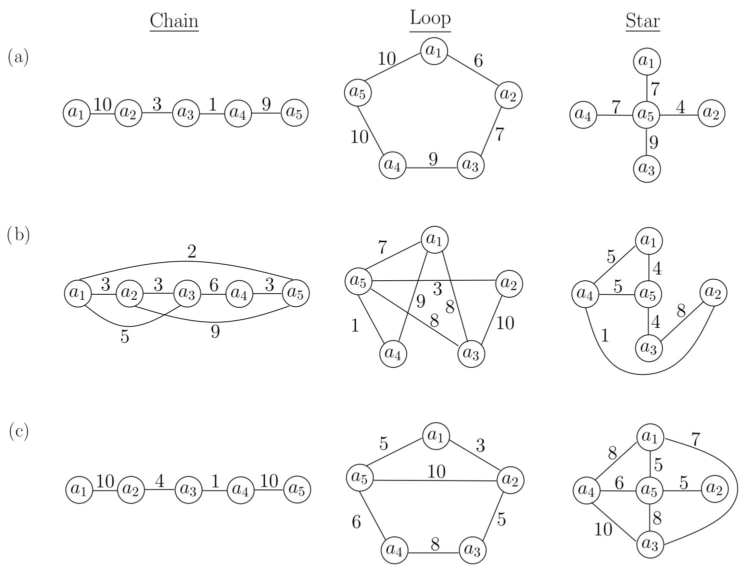


Fig. 8. Results of the learning algorithm using representative graph structure types. The agent capabilities are not shown, and the vertices are laid out for visual purposes. a) Examples of weighted synergy graphs with 5 agents generated to form representative structure types. b) The initial randomly-generated weighted synergy graph of the learning algorithm. c) Learned weighted synergy graphs corresponding to the weighted synergy graphs in (a), after 1000 iterations of simulated annealing.

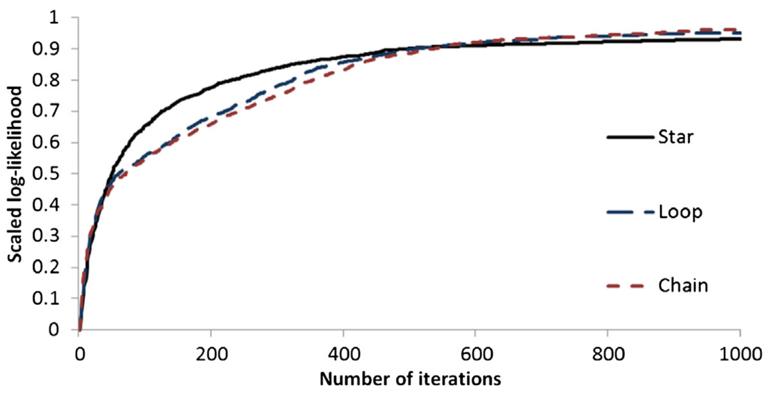


Fig. 9. Performance of the learning algorithm with different weighted synergy graph structure types, averaged over 100 trials.

representative structure types. When the hidden graphs are generated, the edge weights are random integers in the range [1, 10]. The limits were chosen to provide reasonable bounds for the learning algorithm's exploration of structures, while allowing for a significant difference in compatibility among agents.

It is very interesting that the structure learned closely matches the hidden graph, even though the learning algorithm has no prior regarding such structures. The algorithm randomly generates a graph by creating edges between all possible pairs of vertices with 50% probability. Fig. 9 shows the learning curves of simulated annealing for 1000 iterations with 10 agents and averaged over 100 trials per structure type. While the scaled log-likelihood of *star* rises very quickly compared to *loop* and *chain*, the latter two outperform *star* after 600 iterations. The final score of *star*, *loop*, and *chain* are 0.93, 0.95 and 0.96 respectively, which shows that the learned weighted synergy graphs closely matches the hidden ones. The experiments were run in Java using MATLAB as the least-squares solver, on an Intel Quad-Core 2.4 GHz machine. On average, the learning algorithm took 20.6 ± 0.3 seconds to perform 1000 iterations of simulated annealing to learn the weighted synergy graph with 10 agents, and was consistent across the structure types.

5.2. Learning random weighted synergy graphs

In the second set of experiments, we randomly generated weighted synergy graphs, to further explore the space of possible graph structures. We varied the number of agents from 10 to 15, and varied the agent capabilities C with a scale-factor γ. Each agent capability $C_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ was randomly generated such that $\mu_i \in (\frac{\gamma}{2}, \frac{3\gamma}{2})$ and $\sigma_i^2 \in (0, \gamma)$. Thus,

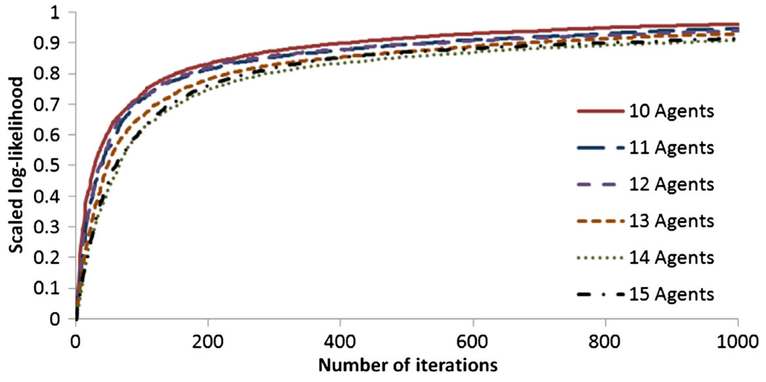


Fig. 10. Performance of the learning algorithm on random graph structures with varying number of agents, averaged over 100 trials.

Table 2
Scaled log-likelihood of the learned weighted synergy graphs, varying the number of agents, and ρ, the scale-factor of agent capabilities.

# agents	Scale-factor of agent capabilities (γ)				
	1.0	2.5	5.0	7.5	10.0
10	0.96	0.95	0.97	0.97	0.96
11	0.96	0.95	0.95	0.95	0.95
12	0.93	0.93	0.94	0.93	0.94
13	0.93	0.93	0.93	0.92	0.93
14	0.90	0.93	0.93	0.92	0.91
15	0.89	0.90	0.91	0.91	0.91

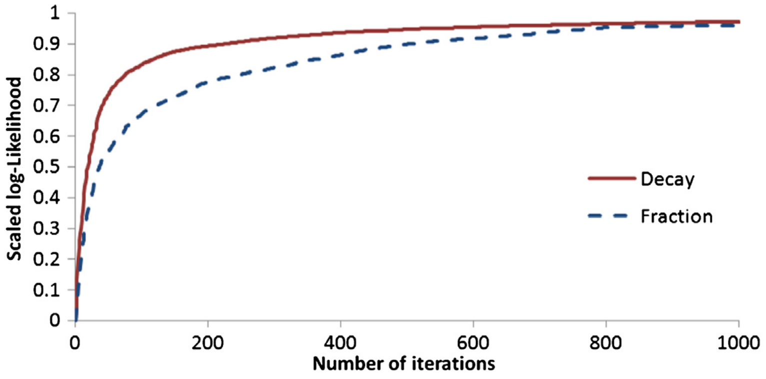


Fig. 11. Learning curves of two compatibility functions — ϕ_{decay} and ϕ_{fraction}, averaged over 100 trials.

a higher value of γ creates a larger range of possible agent capabilities, and we wanted to investigate if it would have any effect on the learning algorithm.

Fig. 10 shows the performance of the learning algorithm with a varying number of agents, and $\gamma = 10$, averaged over 100 trials per number of agents. The shape of the learning curve is consistent, and the performance of the learning algorithm decreases slightly as the number of agents increases. Table 2 shows the final score of the learned weighted synergy graph as the number of agents and scale-factor γ varies. In all cases, the score is 0.89 or higher, which indicates that the learned weighted synergy graph closely matches the hidden one. The scale-factor γ has little to no effect of the final learning outcome, while an increase in the number of agents decreases the score slightly.

In the experiments above, we used the compatibility function $\phi_{\text{fraction}}(d) = \frac{1}{d}$. Other compatibility functions are possible, such as $\phi_{\text{decay}}(d) = \exp(-\frac{d \ln 2}{3})$, which is an exponential decay function. Fig. 11 shows the learning curves of the learning algorithm with the two compatibility functions. Although ϕ_{fraction} increases more slowly than ϕ_{decay}, the final score of the algorithm is similar after all 1000 iterations, which indicates that while the compatibility function has an effect on the learning rate, the final outcome is similar.

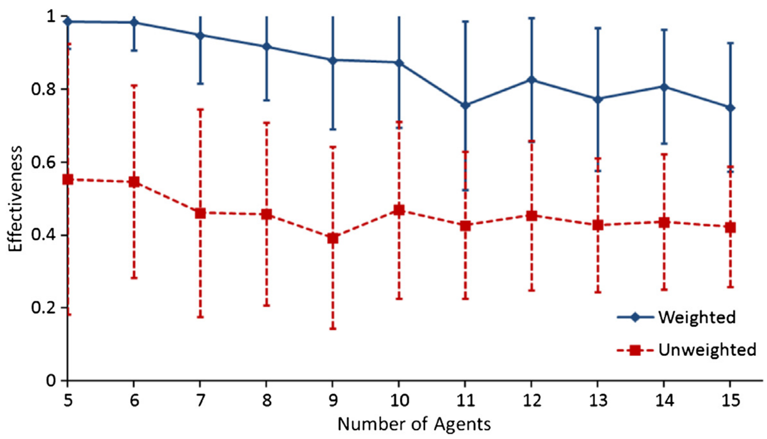


Fig. 12. Effectiveness of teams found in the learned weighted synergy graph, and learned unweighted synergy graphs, using simulated annealing with 1000 iterations. The compatibility function was $\phi_{\text{fraction}}(d) = \frac{1}{d}$.

6. Evaluating the weighted synergy graph model

In the previous section, we evaluated our learning algorithm and showed that it learns representative weighted graph structures and attains high log-likelihood when learning from randomly generated weighted synergy graphs. In this section, we evaluate the expressiveness of the weighted synergy graph model compared to the unweighted synergy graph.

6.1. Experimental setup

In these experiments, we assume that the agent capabilities C are known, and the goal is to learn the structure of the synergy graph. Algorithm 3 learns the structure of the weighted synergy graph, except that `LearnCapabilities` is not called since the agent capabilities are known. To learn the structure of the unweighted synergy graph, the learning algorithm in [26] is used, which is similar to Algorithm 3, except that the neighbor graph function only adds and removes random edges (since all edges are unweighted).

The process of this set of experiments is similar to the previous section, but we only learn the structure of the weighted and unweighted synergy graphs, and not the agent capabilities (since they are known). We varied the number of agents N from 5 to 15. In each trial, a random weighted synergy graph was created where the agent capabilities were generated with $\gamma = 10$. The observation set comprising the performance of teams with 2 and 3 agents are extracted from the hidden synergy graph. In addition, the performance of teams with 4 and more agents are extracted from the hidden synergy graph and form the test set. From the same observation set, a weighted synergy graph and an unweighted synergy graph are learned. We calculate the log-likelihood of the learned weighted synergy graph and unweighted synergy graph using the test data. In addition, with each of the learned synergy graphs, we approximate the δ-optimal team of size $n = \max(4, \lfloor \frac{N}{2} \rfloor)$. We chose such a value of n so as to maximize the number of possible teams (i.e., $\binom{N}{\lfloor \frac{N}{2} \rfloor} \geqslant \binom{N}{i}$ $\forall i \in [1, N]$), while having a minimum size of 4 since the performance of teams with sizes 2 and 3 are used in learning the synergy graphs. From the teams found, we computed their effectiveness:

$$\text{Effectiveness}(A) = \frac{\texttt{Evaluate}(\mathbb{S}(A), 1-\delta) - \texttt{Evaluate}(\mathbb{S}(A_\delta^{\min}), 1-\delta)}{\texttt{Evaluate}(\mathbb{S}(A_\delta^*), 1-\delta) - \texttt{Evaluate}(\mathbb{S}(A_\delta^{\min}), 1-\delta)}$$

where $A_\delta^* = \operatorname{argmax}_A \texttt{Evaluate}(\mathbb{S}(A), 1-\delta)$ is the δ-optimal team, and $A_\delta^{\min}$ is the worst-performing team given δ, i.e., $A_\delta^{\min} = \operatorname{argmin}_A \texttt{Evaluate}(\mathbb{S}(A), 1-\delta)$.

Thus, the effectiveness of a team is a value from 0 to 1, where 1 is the δ-optimal team, and 0 is the worst-possible team. We use the effectiveness to measure the performance of the learned synergy graph, because the goal of the synergy graph is to form an effective team, and the performance is scaled since the actual performance of teams varies from trial to trial due to the randomized generation of the hidden synergy graph.

6.2. Comparison results

In our first set of experiments, we used the fraction compatibility function, i.e., $\phi_{\text{fraction}}(d) = \frac{1}{d}$, and set $\delta = 0.5$. For each value of N, we performed 100 trials, where a different hidden synergy graph was generated in each trial. The weighted and unweighted synergy graphs were then learned using 1000 iterations of simulated annealing for each trial using the observation set extracted from the hidden model. Fig. 12 shows the effectiveness of the teams found by the weighted and unweighted synergy graphs. As the number of agents N increases, the effectiveness of the teams found by both synergy

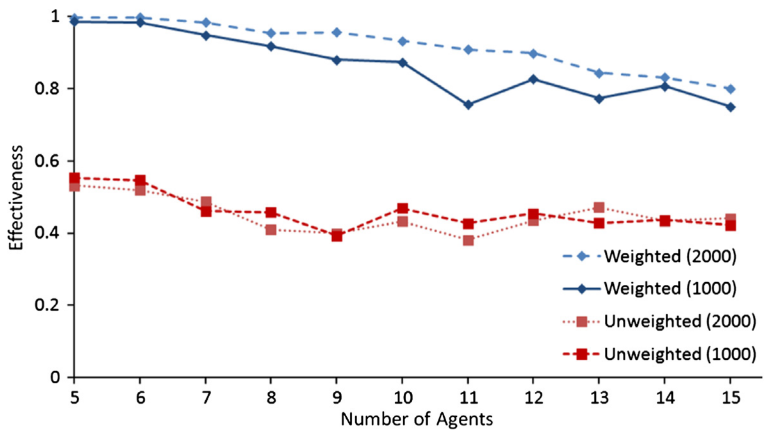


Fig. 13. Average effectiveness of teams found in the learned weighted and unweighted synergy graphs, using 1000 and 2000 iterations of simulated annealing to learn the synergy graph structure, using $\phi_{\text{fraction}}(d) = \frac{1}{d}$.

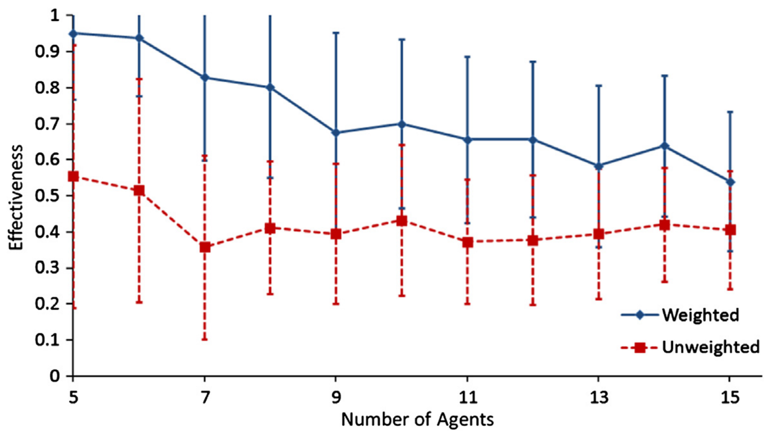


Fig. 14. Effectiveness of teams found in the learned weighted and unweighted synergy graphs with $\phi_{\text{decay}}(d) = \exp(-\frac{d \ln 2}{2})$, and 1000 iterations of simulated annealing.

graph types decrease, reflecting the increase in difficulty in learning the graph and finding the δ-optimal team. However, across all values of N, the team found by the learned weighted synergy graph outperforms the team found by the learned unweighted synergy graph. We performed a paired Student's T-test (one-tailed) on the 1100 total trials (11 values of N with 100 trials per N), and the results were statistically significant to a value of $p = 9.8 \times 10^{-258}$.

We repeated the experiments with $\phi_{\text{fraction}}(d) = \frac{1}{d}$, but increased the number of iterations of simulated annealing to 2000, to investigate if a higher number of iterations would improve the overall performance of the learned synergy graphs. Fig. 13 shows the average effectiveness of the teams found by the weighted and unweighted synergy graphs with both 1000 and 2000 iterations of simulated annealing. The learned weighted synergy graph performs better with 2000 iterations compared to 1000 iterations (statistically significant to a value of $p = 8.7 \times 10^{-20}$). However, a greater number of iterations of simulated annealing does not affect the performance of the learned unweighted synergy graph ($p = 0.14$). Thus, a greater number of iterations of simulated annealing allows the weighted synergy graph learning algorithm to converge on a closer match to the hidden synergy graph, while the "best" unweighted synergy graph is already found within 1000 iterations and hence increasing the number of iterations has little effect.

Thirdly, we used the compatibility function $\phi_{\text{decay}}(d) = \exp(-\frac{d \ln 2}{h})$ with the half-life $h = 2$. In this way, the compatibility function ϕ_{decay} decreases at a slower pace than ϕ_{fraction} initially, but has much smaller values once d is large. As before, we varied N and ran 100 trials per value of N. We ran 1000 iterations of simulated annealing for both learning algorithms, and Fig. 14 shows the effectiveness of the teams found by the learned synergy graphs. While the effectiveness of the learned weighted synergy graph decreases more rapidly as N increases compared to ϕ_{fraction}, the learned weighted synergy graph outperforms the learned unweighted synergy graph, with $p = 3.6 \times 10^{-160}$.

Thus, these experiments show that the weighted synergy graph is more expressive than the unweighted synergy graph. The results are statistically significant across a large range of agent sizes, with two compatibility functions, and with different number of iterations of simulated annealing.

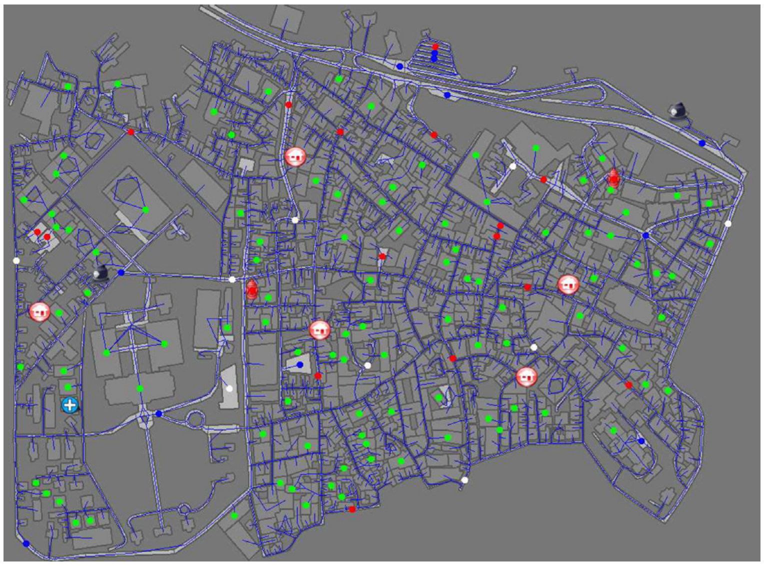


Fig. 15. Screenshot of the RoboCup Rescue simulator showing the initial positions of the simulated robots. Green, red, blue, and white circles are civilians, fire engines, police cars, and ambulances respectively. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

7. Using weighted synergy graphs with the RoboCup Rescue simulator

The previous sections showed the efficacy of our learning algorithm, and the expressiveness of the weighted synergy graph model. In this section, we evaluate using the weighted synergy graph model to capture the team performance of simulated robots in a realistic rescue scenario.

7.1. The RoboCup Rescue simulator

The RoboCup Rescue Simulation League provides an open-source simulator for agent development [20]. The simulator uses a map of a city, such as Berlin, Istanbul, Paris etc. (Fig. 15), and simulates the event that a natural disaster has occurred. Civilians are randomly positioned in the city with varying amounts of health, some of whom may be buried under rubble. Fires break out in random parts of the city, and roads throughout the city are blocked by fallen debris, making them impassable for humans and vehicles.

Three types of rescue robots are deployed: ambulances, fire engines and police cars. The ambulances help to rescue civilians, while fire engines put out fires and police cars clear the road obstructions. The simulator runs for a fixed number of timesteps, and the score is a weighted sum based on the number of civilians and rescue robots alive and the proportion of buildings in the city that are not burnt.

The RoboCup Rescue domain is a multi-robot task allocation problem in the ST-MR-TA category [17]. Different approaches have been used to solve this problem, such as treating it as a generalized allocation problem [13], using a biologically-inspired approach [12], and extending coalition formation to handle spatial and temporal constraints [35].

7.2. Experimental setup

As part of the RoboCup Rescue competition, participants from various universities around the world develop algorithms to control all the rescue robots (i.e., the non-civilians). We use the RoboCup Rescue simulator as an ad hoc multi-robot scenario, where combinations of pre-existing algorithms are used to compose an effective team. The rescue robots have a standardized communication protocol defined in the RoboCup Rescue simulator, which allows different RoboCup participants' algorithms to be run simultaneously, each controlling a subset of the rescue robots. In particular, we are interested in modeling the performance of ad hoc combinations of these algorithms. For example, when running two algorithms simultaneously, each algorithm controls half of the rescue robots. We wanted to compare the effectiveness of the weighted synergy graph model at modeling the interactions and forming an effective team, versus the unweighted synergy graph [26] as well as IQ-ASymTRe [48].

We used the Istanbul1 map from RoboCup 2011, and the source code of 6 RoboCup participants (Poseidon, RoboAKUT, Ri-one, RMAS_ArtSapience, SBCe_Saviour, and SEU_RedSun) [36]. The source code of 8 RoboCup participants were available for download, but only 6 ran out of the box without much modification. In the Istanbul1 map, there are 46 rescue robots to be controlled, and each of the 6 RoboCup algorithms are designed to control all 46 robots to perform the task. We treated the 6 RoboCup algorithms as 6 separate agents, such that any subset of these 6 agents can be used to control the 46 robots.

We distributed the 46 rescue robots among the selected agents randomly, such that each agent controlled an approximately equal number of rescue robots. For example, if two agents (RoboCup algorithms) were picked, then each algorithm would control 23 robots (assigned randomly), and the score of the two agents would be the score returned by the RoboCup Rescue simulator at the end of its simulation, which is based on the number of civilians and rescue robots alive and the health of the buildings in the city. The RoboCup Rescue simulator models randomness and noise in the simulation, and we used the default settings of the simulator in our experiments, with the Istanbul1 map from RoboCup 2011.

We varied the number of selected agents N from 2 to 5. For each value of N, there are $\binom{6}{N}$ combinations of agents, and we ran 30 simulations for each combination. For example, when $N = 3$, there are 20 combinations of agent triples (e.g., Poseidon, RoboAKUT, and Ri-one), and we ran 30 simulations for each triple. Each simulation had a different allocation of rescue robots to agents, and hence each simulation resulted in a different score given by the simulator. An observation consists of the agents (e.g., Poseidon, RoboAKUT, and Ri-one) and a single score they attained (e.g., 14.8), and there are 30 observations per agent team combination.

We used the scores of simulation runs where $N = 2$ and $N = 3$ as the observation set for training, with $1050 = 30(\binom{6}{2} + \binom{6}{3})$ total observations. To evaluate the learned models, the algorithms formed teams of size 4 and 5. Since the score of a team changes depending on the robot allocation, we used the average score attained in the 30 simulations as our measure, hence corresponding to $\delta = 0.5$ in our problem definition. We did not form any team of size 6, because only one such team exists (using all the agents). We did not include data from $N = 1$ for training or testing, since the simulator is deterministic (given a fixed initial random seed) so there would not be variance in the agents' performance over 30 simulations.

7.3. Modeling the agent interactions

For the weighted and unweighted synergy graph models, we used the decay compatibility function, i.e., $\phi_{\text{decay}}(d) = \exp(-\frac{d \ln 2}{h})$, where $h = 2$, and ran 1000 iterations of simulated annealing. We chose the decay compatibility function as the compatibility decreases more gradually than ϕ_{fraction}, and used 1000 iterations of simulated annealing as it had good results in the previous sections. In addition, since the learned synergy graph depends on a random process of changing edges, we performed 10 trials to learn the synergy graph from the RoboCup Rescue data.

IQ-ASyMTRe [48] calculates the expected cost of a coalition A and task t as:

$$\text{cost}(A,t) = \widehat{\text{cost}}(A,t)/F(Q_A, Y_t) \tag{7.1}$$

where $\widehat{\text{cost}}(A,t)$ is the summation of costs of all activated schemas in A, Q_A is the coalition quality of A, and Y_t is the task type of t.

Since we have a single task, and all coalitions (combinations of agents) can complete the task, we set $F(Q_A, Y_t) = 1$ for all A. As such, $\text{cost}(A,t) = \widehat{\text{cost}}(A,t)$. Since the internal schemas of the participants' algorithms are unknown, we treat each agent as a single schema, and estimate its cost as the average of the score of agent combinations involving it:

$$\widehat{\text{cost}}(a) = \frac{\sum_{A \text{ s.t. } a \in A} \text{score}(A)}{|A \text{ s.t. } a \in A|} \tag{7.2}$$

where a is an agent (i.e., one of the 6 algorithms), A is a team of 2 to 5 agents, and $\text{score}(A)$ is the score obtained in the RoboCup Rescue simulator using the agents in A to control the rescue robots.

From the cost of each agent, we then define the cost of a coalition in IQ-ASymTRe as:

$$\widehat{\text{cost}}(A,t) = \sum_{a \in A} \widehat{\text{cost}}(a) \tag{7.3}$$

To form a team using IQ-ASyMTRe, we iterate through all possible combinations of agents given the desired team size, and pick the team with the highest cost. Typically, the team with the lowest cost is picked in IQ-ASyMTRe, but because we used the score as the measure of cost (there is no actual metric for cost of the algorithms), it is desirable to pick the team with the highest score.

7.4. Ad hoc team formation results

Table 3 shows the scores of the teams formed with the weighted synergy graph, unweighted synergy graph, and IQ-ASyMTRe. The score corresponds to the final value returned by the RoboCup simulator at the end of the simulation. The weighted and unweighted synergy graph models perform similarly, showing that while the weighted synergy graph model is more expressive (shown in the previous section), the interactions of the agents in the RoboCup Rescue domain can be modeled with an unweighted graph. We believe that both synergy graph models performed identically due to the small number of agents — if there were a larger number of agents then the interactions would be more complex and the weighted synergy graph would outperform the unweighted one. Both synergy graph models find the optimal 4-agent team, and find the optimal 5-agent team 80% of the time. In comparison, IQ-ASyMTRe finds a good but non-optimal 4-agent team, and finds a 5-agent team that is close to the worst possible combination. The results are statistically significant to a value

Table 3
Scores of combinations of algorithms in the RoboCup Rescue simulator, formed by the weighted synergy graph model and other algorithms.

Algorithm	4 agents	5 agents
Weighted Synergy Graph	14.3 ± 0	12.7 ± 0.7
Unweighted Synergy Graph	14.3 ± 0	12.7 ± 0.7
IQ-ASyMTRe	12.3	8.4
Best Possible Team	14.3	13.0
Worst Possible Team	7.1	8.1

of $p = 5.4 \times 10^{-137}$ for 4 agents, and $p = 3.7 \times 10^{-9}$ for 5 agents (single-tailed paired T-test) between the synergy graph model and IQ-ASyMTRe. The p-values for the weighted and unweighted models versus IQ-ASMTRe are identical since both synergy graph models attained the same results.

Thus, the synergy graph model outperforms IQ-ASyMTRe. The results are compelling in that only observations of 2 and 3 agents were used for training, but teams of 4 and 5 agents were formed. Further, no assumptions of the agents were used and they were treated as black-boxes in the synergy graph model. Thus, the efficacy of the synergy graph model, the learning and team formation algorithms have been demonstrated on the RoboCup Rescue domain and can be extended to many other domains.

8. Conclusions

We contributed a representation of team performance that goes beyond the sum of single-agent capabilities; we formally introduced the weighted synergy graph model, where the inherent differences in the agents' capabilities are modeled with Normally-distributed variables, and the task-based relationship is modeled as a connected weighted graph. Weighted synergy graphs are applicable to many multi-agent domains, and we presented an example of a rescue task, and showed how different agents and their task-based relationships are modeled effectively.

To apply weighted synergy graphs on problem domains, we contributed an algorithm that learns a weighted synergy graph from observations, and two team formation algorithms that use a learned weighted synergy graph to form an effective multi-agent team. These algorithms enable the weighted synergy graph model to be applied to a variety of problems, since the only input are the training observations. The learning algorithm learns the structure of the weighted graph by iteratively improving the graph structure, and learns the agents' capabilities by solving a system of equations derived from the observations and the graph structure. Our first team formation algorithm uses branch-and-bound to find the optimal team, and our second team formation algorithm explores possible teams and approximates the optimal team. Although the approximation algorithm does not have formal guarantees on the formed team, we found that the team that is formed is close to optimal while only exploring a small subset of the space. The optimal team formation algorithm searches a large amount of space in order to guarantee optimality, so we believe the approximation algorithm is more applicable due to scalability issues with the optimal team formation algorithm in real problems.

We extensively evaluated our learning algorithm. First, we created random weighted synergy graphs with representative graph structure types, and demonstrated that our learning algorithm learned weighted synergy graphs with structures similar to the types. Second, we generated random weighted synergy graphs that were hidden from the learning algorithm, and showed that the log-likelihood of the learned weighted synergy graph closely matches that of the hidden ones. Thus, the agent capabilities and graph structures were effectively learned using our learning algorithm. Although we evaluated the learning algorithm with data derived from weighted synergy graphs, the learning algorithm is applicable to data from any source and will learn the closest weighted synergy graph that matches the data. The learning algorithm does not have guaranteed bounds on the learned result but from our experiments, we believe that a small number of iterations of simulated annealing (compared to the entire space of weighted graphs) is sufficient to learn a model that matches the observed data well.

We applied the weighted synergy graph model to a problem domain: urban search-and-rescue. Using the RoboCup Rescue simulator, we made minor modifications to six algorithms written by RoboCup participants, such that combinations of these algorithms can be run in parallel to perform the RoboCup Rescue task. We treated each of these six algorithms as separate agents, and modeled their capabilities and task-based relationships when combined to form an ad hoc team. We showed that the weighted synergy graph model forms a near-optimal team, outperforming the teams selected with IQ-ASyMTRe. Hence, the weighted synergy graph model effectively learned and modeled the synergy of the RoboCup Rescue algorithms.

The weighted synergy graph effectively models agent capabilities and their task-based relationships. Our algorithms only require the assumption of retrieving observations of the agents' performance at the task, which makes them applicable to many multi-agent and multi-robot domains. In particular, as long as problem domains can be treated as a black-box where the input is a multi-agent team, and the output is an observed performance, the weighted synergy graph model is applicable. We have applied the weighted synergy graph model to multi-robot problems such as foraging [27] and configuring robots from modules [30], and we continue to apply our model to other domains. Modeling and learning the synergy of multi-agent teams is a novel contribution, and we are actively pursuing this area of research. We have successfully applied weighted synergy graphs on actual robots performing role assignment [27] and forming teams that are robust to failures [28], and we have introduced an algorithm to learn the synergy of a new teammate [29]. We are using synergy graphs on other interesting problems, such as modeling task-based relationships that are non-transitive, and modeling agents that learn to coordinate better over time.

Acknowledgements

This work was partially supported by the Air Force Research Laboratory under grant number 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] N. Agmon, P. Stone, Leading multiple ad hoc teammates in joint action settings, in: AAAI Workshop: Interactive Decision Theory and Game Theory, 2011.
[2] B. Banerjee, 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.
[3] S. Barrett, P. Stone, Ad hoc teamwork modeled with multi-armed bandits: An extension to discounted infinite rewards, in: Proc. Int. Conf. Autonomous Agents and Multiagent Systems – Adaptive Learning Agents Workshop, 2011.
[4] S. Barrett, P. Stone, S. Kraus, Empirical evaluation of ad hoc teamwork in the pursuit domain, in: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2011, pp. 567–574.
[5] B. Bulka, M. Gaston, M. desJardins, Local strategy learning in networked multi-agent team formation, J. Autonom. Agents Multi-Agent Syst. 15 (2007) 29–45.
[6] J. Chen, D. Sun, Resource constrained multirobot task allocation based on leader-follower coalition methodology, J. Robot. Res. 30 (12) (2011) 1423–1434.
[7] M. de Weerdt, Y. Zhang, T. Klos, Distributed task allocation in social networks, in: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2007, pp. 500–507.
[8] M.B. Dias, TraderBots: A new paradigm for robust and efficient multirobot coordination in dynamic environments, PhD thesis, The Robotics Institute, Carnegie Mellon University, 2004.
[9] M.B. Dias, A. Stentz, A free market architecture for distributed control of a multirobot system, in: Proceedings of the International Conference on Intelligent Autonomous Systems, 2000, pp. 115–122.
[10] M.B. Dias, A. Stentz, Multi-robot exploration controlled by a market economy, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002, pp. 2714–2720.
[11] C. Dorn, S. Dustdar, Composing near-optimal expert teams: A trade-off between skills and connectivity, in: Proceedings of the International Conference on Cooperative Information Systems, 2010, pp. 472–489.
[12] Fernando dos Santos, Ana L.C. Bazzan, Towards efficient multiagent task allocation in the robocup rescue: A biologically-inspired approach, J. Autonom. Agents Multi-Agent Syst. 22 (2011) 465–486.
[13] P. Ferreira, F. Dos Santos, A.L. Bazzan, D. Epstein, S.J. Waskow, RoboCup Rescue as multiagent task allocation among teams: experiments with task interdependencies, J. Autonom. Agents Multi-Agent Syst. 20 (2010) 421–443.
[14] M. Gaston, M. desJardins, Agent-organized networks for dynamic team formation, in: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2005, pp. 230–237.
[15] K. Genter, N. Agmon, 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.
[16] J.M. George, J. Pinto, P.B. Sujit, J.B. Sousa, Multiple UAV coalition formation strategies (extended abstract), in: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2010, pp. 1503–1504.
[17] B.P. Gerkey, M.J. Mataric, A formal analysis and taxonomy of task allocation in multi-robot systems, J. Robot. Res. 23 (9) (2004) 939–954.
[18] M. Isik, F. Stulp, G. Mayer, H. Utz, Coordination without negotiation in teams of heterogeneous robots, in: Proceedings of the RoboCup International Symposium, 2006, pp. 355–362.
[19] R.M. Karp, Reducibility among combinatorial problems, in: Complexity of Computer Computations, 1972, pp. 85–103.
[20] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, S. Shimada, RoboCup Rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research, in: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 1999, pp. 739–743.
[21] A. Kleiner, M. Brenner, T. Bräuer, C. Dornhege, M. Göbelbecker, M. Luber, J. Prediger, J. Stückler, B. Nebel, Successful search and rescue in simulated disaster areas, in: RoboCup 2005: Robot Soccer World Cup IX, vol. 4020, 2006, pp. 323–334.
[22] T. Lappas, K. Liu, E. Terzi, Finding a team of experts in social networks, in: Proceedings of the International Conference on Knowledge Discovery and Data Mining, 2009, pp. 467–476.
[23] C. Li, M. Shan, Team formation for generalized tasks in expertise social networks, in: Proceedings of the International Conference on Social Computing, 2010, pp. 9–16.
[24] S. Liemhetcharat, M. Veloso, Mutual state capability-based role assignment model (extended abstract), in: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2010, pp. 1509–1510.
[25] S. Liemhetcharat, M. Veloso, Modeling mutual capabilities in heterogeneous teams for role assignment, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011, pp. 3638–3644.
[26] S. Liemhetcharat, 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. 365–375 (nominated for Best Student Paper Award).
[27] S. Liemhetcharat, 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.
[28] S. Liemhetcharat, M. Veloso, Forming an effective multi-robot team robust to failures, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2013, pp. 5240–5245.
[29] S. Liemhetcharat, M. Veloso, Learning the synergy of a new teammate, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2013, pp. 5246–5251.

[30] S. Liemhetcharat, M. Veloso, Synergy graphs for configuring robot team members, in: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2013, pp. 111–118.
[31] C. Lim, R. Mamat, T. Braunl, Market-based approach for multi-team robot cooperation, in: Proceedings of the International Conference on Autonomous Robots and Agents, 2009, pp. 62–67.
[32] T. Michalak, D. Marciniak, M. Szamotulski, T. Rahwan, M. Wooldridge, P. McBurney, 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.
[33] L. Parker, F. Tang, Building multirobot coalitions through automated task solution synthesis, Proc. IEEE 94 (7) (2006) 1289–1305.
[34] T. Rahwan, T. Michalak, N. Jennings, M. Wooldridge, P. McBurney, Coalition formation with spatial and temporal constraints, in: Proceedings of the International Joint Conference on Artificial Intelligence, 2009, pp. 257–263.
[35] S. Ramchurn, M. Polukarov, A. Farinelli, C. Truong, Coalition formation with spatial and temporal constraints, in: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2010, pp. 1181–1188.
[36] RoboCupRescue, RoboCup Rescue Simulation League, http://roborescue.sourceforge.net/, 2011 (online).
[37] T. Sandholm, K. Larson, M. Andersson, O. Shehory, F. Tohme, Coalition structure generation with worst case guarantees, J. Artif. Intell. 111 (1999) 209–238.
[38] T. Service, J. Adams, Coalition formation for task allocation: theory and algorithms, J. Autonom. Agents Multi-Agent Syst. 22 (2011) 225–248.
[39] O. Shehory, S. Kraus, Task allocation via agent coalition formation, J. Artif. Intell. 101 (1–2) (1998) 165–200.
[40] P. Stone, G. Kaminka, S. Kraus, J. Rosenschein, Ad hoc autonomous agent teams: Collaboration without pre-coordination, in: Proceedings of the International Conference on Artificial Intelligence, 2010.
[41] P. Stone, G. Kaminka, J. Rosenschein, Leading a best-response teammate in an ad hoc team, in: Agent-Mediated Electronic Commerce: Designing Trading Strategies and Mechanisms for Electronic Markets, 2009, pp. 132–146.
[42] P. Stone, S. Kraus, To teach or not to teach? Decision making under uncertainty in ad hoc teams, in: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2010, pp. 117–124.
[43] F. Tang, L. Parker, A complete methodology for generating multi-robot task solutions using ASyMTRe-D and market-based task allocation, in: Proceedings of the IEEE International Conference on Robotics and Automation, 2007, pp. 3351–3358.
[44] P. Tosic, G. Agha, Maximal clique based distributed coalition formation for task allocation in large-scale multi-agent systems, in: Proceedings of the International Workshop on Massively Multi-Agent Systems, 2004, pp. 104–120.
[45] L. Vig, J. Adams, Market-based multi-robot coalition formation, in: Proceedings of the International Symposium on Distributed Autonomous Robotics Systems, 2006, pp. 227–236.
[46] L. Vig, J. Adams, Coalition formation: From software agents to robots, J. Intell. Robot. Syst. 50 (2007) 85–118.
[47] F. Wu, S. Zilberstein, X. Chen, Online planning for ad hoc autonomous agent teams, in: Proceedings of the International Joint Conference on Artificial Intelligence, 2011, pp. 439–445.
[48] Y. Zhang, L. Parker, Task allocation with executable coalitions in multirobot tasks, in: Proceedings of the IEEE International Conference on Robotics and Automation, 2012.
[49] R. Zlot, A. Stentz, M.B. Dias, S. Thayer, Multi-robot exploration controlled by a market economy, in: Proceedings of the IEEE International Conference on Robotics and Automation, 2002, pp. 3016–3023.