24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

An ideal team is more than a team of ideal agents

Can Kurtan and **Pınar Yolum** and **Mehdi Dastani**[1]

Abstract. The problem of building a team to perform a complex task is often more than an optimal assignment of subtasks to agents based on individual performances. Subtasks may have subtle dependencies and relations that affect the overall performance of the formed team. This paper investigates the dependencies between subtasks and introduces some desired qualities of teams, such as preserving privacy or fairness. It proposes algorithms to analyze and build teams by taking into account the dependencies of assigned subtasks and agent performances. The performance of the algorithms are evaluated experimentally based on a multiagent system that is developed to answer complex queries. We show that by improving an initial team iteratively, the algorithm obtains teams with higher performance.

1 INTRODUCTION

The problem of forming teams has been a central question in many disciplines [3]. Put simply, the problem is to form a set of agents (human or software) with required capabilities so that they can perform a task together. In the domain of service composition, this would correspond to a set of service providers, each performing a single service, which overall yields a composition of the desired service [9]. In the domain of query answering, this would correspond to data sources that can each answer part of a given question [8]. In the domain of rescue operations, this would correspond to a set of human and robot agents that work to help civilians [11].

While traditionally, the term team was used to refer to a set of agents [2, 15, 19], the newer definitions refer the specific subtask that will be performed by each agent [4, 10]. Following this, here we refer to teams as the set of agents and their assigned subtasks. Assigning subtasks to individual agents is typically modeled as an optimization problem, where coverage of the subtasks is maximized, or the number of agents involved is minimized. Many formalizations of the problem exist in the literature [13, 15, 19]. Earlier approaches model team formation with three important assumptions: (i) the overall task can be divided into independent subtasks (ii) agents' capabilities as to what subtask they can do is known or computed easily and (iii) the agents' capabilities of performing a subtask is binary (e.g., no variation in the quality of the subtask performance). Even under these assumptions, the problem of assigning subtasks to individual agents is known to be NP-hard [2, 10, 15].

In many domains, these assumptions do not hold. Consider the following simple example from a query answering domain, where the question is to find the title, author and the summary of books. Three agents separately provide book names, book authors, and book summaries but do not provide the information for the same books. An answer for the question cannot be found by using the information of these agents because the books do not match. Many agents provide the information of book titles, authors, or summaries at varying degrees; e.g., one agent knows three book titles and a thousand book summaries, while another agent knows a thousand book titles and three book summaries. The individual performance of an agent depends on which subquery the agent is assigned. The number of books returned in the answer depends on which agents are assigned to which subqueries. Hence, building a team based on finding best performing individuals does not always yield the best performing team. We tackle this problem as an iterative process where a given team can be improved by aiming desired qualities, such as better performance or fairness.

Performance is generally deemed the most important criteria for evaluating a team. However, in addition to performance, there are other qualities of team building that are necessary to operate successfully. For example, a team that carries out a critical task might be required to keep the task details private. Is it possible to quantify the teams in terms of privacy risks they pose? Or, a team building process might require to be fair in giving chances to the possible candidates in forming teams. How would the fairness of a team building algorithm be measured? Depending on the content of a task, diversity of the team might be required [14]. Is it possible to identify diverse agents from their working with other agents automatically?

This paper proposes algorithms for addressing the above challenges: building teams for tasks whose subtasks are dependent to each other and the performance of agents is affected by the dependencies. We represent our knowledge of the agents in an environment using *expertise graph*, which keeps information on how well agents perform tasks individually and how well they can support each other on common or different subtasks. Our algorithms make use of this expertise graph to approximate how likely the agents are to perform well in dependent subtasks. We provide metrics to measure various properties of team building including fairness, privacy, and diversity. We demonstrate the workings of our algorithms in a query answering multiagent system, where agents are data providers and tasks are queries. We evaluate the proposed algorithms in an experimental setup and show that by improving existing teams incrementally, we can build better performing teams.

The rest of this paper is organized as follows: Section 2 demonstrates the team formation problem for question-answering multiagent teams. Section 3 describes the representation of the expertise graph and metrics to analyze the graph. Section 4 provides both one shot and iterative algorithms for building teams. Section 5 evaluates our algorithms and shows that they can form well-performing teams. Finally, Section 6 discusses our work in relation to other multiagent team formation approaches in the literature.

[1] Utrecht University, The Netherlands, email: {a.c.kurtan, p.yolum, m.m.dastani}@uu.nl

2 QUERY ANSWERING AS A TEAM

We assume a set of agents $A = \{a_1, a_2, \dots a_n\}$ and a set of capabilities $C = \{c_1, c_2, \dots c_m\}$ exist. We use $a_C \in 2^C$ to denote the set of capabilities of agent $a \in A$. We define a task T as a set of capabilities, i.e., $T \in 2^C$, and a team K as a set of agents to each of which a task is assigned, i.e., $K = \{\langle a, T\rangle \mid a \in A \,\&\, T \in 2^C\}$. We use $\mathcal{K}$ to denote the set of all possible teams. A team is defined correctly if and only if $\forall \langle a, T\rangle \in K \;:\; T \subseteq a_C$. For notation convenience, we introduce three auxiliary functions $\mathsf{P_{cap}}$, $\mathsf{P_{agent}}$, and $\mathsf{P_{team}}$. The function $\mathsf{P_{cap}} : A \times C \to \mathbb{N}$ is assumed to measure the performance of an agent for an assigned capability, the function $\mathsf{P_{agent}} : A \times 2^C \to \mathbb{N}$ is assumed to measure the performance of an agent for an assigned task, and the function $\mathsf{P_{team}} : \mathcal{K} \to \mathbb{N}$ is assumed to measure the performance of a team. The performance of a team is of course measured with respect to the tasks assigned to the agents that are participating in a team.

An important domain for finding teams is that of information seeking. Information seeking is a challenging process for users, especially when searching answers for a complex query. Finding a source that can provide all the information required to answer every constraint in the query is generally not possible [18]. Therefore, answering queries requires dividing the query into subqueries, finding the right agents to ask each subquery, and collecting data from each agent.

We specifically target query answering in Semantic Web domain, where the structure and interlinks enable the use of distributed data sources through semantic queries [6]. Data sources are interconnected to each other by either using the Uniform Resource Identifiers (URIs) or linking to different URIs of the same entities. Thus, searching through various sources for the same entity to find more information becomes possible. We map the problem of semantic query answering as follows:

- **Agents** A is the set of data source agents that can answer queries.
- **Capabilities** C is the set of predicates corresponding to properties of entities, i.e., predicates used in the ontology.
- **Task** $T \in 2^C$ is the set of predicates used in a conjunctive SPARQL query. We assume that a conjunctive query can be constructed based on the predicates from T using one query variable.
- **Team** $K = \{\langle a, T\rangle \mid a \in A \,\&\, T \in 2^C\}$ is the set of agents with their assigned sub-queries.
- **Performance of a query answering team** ($\mathsf{P_{team}}$) is defined as the number of entities (URIs) returned as the answer of a query, i.e., $\mathsf{P_{team}}(K) = n$ where n is the number of returned entities.
- **Performance of a data source agent** ($\mathsf{P_{agent}}$) is defined as the number of entities (URIs) returned as the answer of a query assigned to an agent, i.e., $\mathsf{P_{agent}}(a, T) = n$ where n is the number of entities returned by agent $a \in A$ for task T.
- **Performance of a property** ($\mathsf{P_{cap}}$) is defined as the number of entities (URIs) returned as the answer of a query consisting of one single property assigned to an agent, i.e., $\mathsf{P_{cap}}(a, c) = n$ where n is the number of entities returned by agent $a \in A$ for property $c \in C$.

We would like to observe that performance of a query answering team decreases as the size of the task increases. This is due to the nature of tasks defined as conjunctive queries. Consequently, the performance of a query answering team is always less or equal to the performance of the worst performing agent in the team, i.e., $\mathsf{P_{team}}(K) \leq \min\{\mathsf{P_{agent}}(a, T) \mid \langle a, T\rangle \in K\}$.

We assume there is a dedicated agent that takes a query, assign sub-queries to source agents, and combines resulting data to answer the given query. To be able to build a team, this agent stores the required information, such as the capabilities of agents.

Example 2.1. Three agents $A = \{a, b, c\}$ are data providers of entities (books) for properties "name", "author", "summary" and "publisher". Capabilities and corresponding performance values are given below:

Agent	name	author	summary	publisher
a	10	5	7	-
b	11	-	7	8
c	-	6	10	1

Agent a, which has capabilities $a_C = \{name, author, summary\}$, can return ten books with name and five books with author, *i.e.,* $\mathsf{P_{cap}}(a, name) = 10$ and $\mathsf{P_{cap}}(a, author) = 5$. For a conjunctive query asking the books having both name and author information, we know that the answer would be at most five books because $\min(\mathsf{P_{cap}}(a, name), \mathsf{P_{cap}}(a, author)) = 5$.

For a conjunctive query that questions books whose name, author, summary, and publisher information present together, how can source agents be assigned to subqueries in order to maximize the number of results that will be produced by the team? A naive approach that considers only capability performances would assign agents to subtasks that has the highest performance for the capability, such as $K_1 = \{\langle b, \{name, publisher\}\rangle, \langle c, \{author, summary\}\rangle\}$.

When we evaluate the performance of the team on the real data, we see that the team cannot return any result for the given query, i.e., $\mathsf{P_{team}}(K_1) = 0$. On the other hand, team $K_2 = \{\langle a, \{summary, name\}\rangle, \langle b, \{publisher\}\rangle\}, \langle c, \{author\}\rangle\}$ performs much better, $\mathsf{P_{team}}(K_2) = 5$, because agent a can provide "name" and "summary" information for every book whose "publisher" information is provided by agent b and "author" information is provided by agent c.

3 EXPERTISE GRAPH

As seen above, the assignments of subtasks to agents in a team will be affected by the relations among subtasks, resulting in variance in the performance of the team. As is customary, we assume that we know the individual capabilities of agents. Further, we assume that we have an indication of how well two agents can carry out two subtasks together.

Inspired by social networks, we propose to represent the interplay between possible assignment as a graph model, called *expertise graph*, which stores the capability and cooperation performances of agents. Contrary to social networks where each node is an agent, here each node is a possible assignment, which denotes an agent exhibiting a particular capability; e.g., $(a, name)$ and the edge between two nodes denotes how well the two assignments would work together.

Definition 3.1 (Expertise Graph). $G = \langle V, E\rangle$ is an undirected graph that represents agent-capability pairs (assigned capability) as nodes V and pair-wise co-performances as edges E. An edge e between two nodes $v_1 = \mathsf{node}(a_i, c_k)$ and $v_2 = \mathsf{node}(a_j, c_l)$ can be one of the three types: *c*-edge, *s*-edge, or *j*-edge.

A relation between two nodes can be one of the three types: (i) The assignments in the two nodes could be carried out by the same agent, meaning that the agent is *competent* in carrying out a subtask that requires two capabilities. That is, $a_i = a_j$ and $c_k \neq c_l$, then the edge e is type *c*-edge, which corresponds to *competency* of agent a_i

to be responsible for capabilities c_k, c_l together. (ii) The assignments in the two nodes are done by different agents but the subtasks are the same, such that when carried out together one agent is *supporting* the other agent in the same subtask. That is, if $a_i \neq a_j$ and $c_k = c_l$, then the edge e is an *s*-edge. (iii) The nodes have different agents and subtasks, then the edge e is a *j*-edge, which represents *joint* work of two agents, where each is responsible for a different capability. Thus, the performance value on each edge is given a meaning based on the edge type, where the performance for the *c*-edge denotes how well the agent can carry out two subtasks, whereas for the *j*-edge the value denotes two agents working on two different tasks, and for the *s*-edges how well one agent supports the other on the same subtask. Formally, let $\mathsf{node} : A \times C \rightarrow V$ be a function that returns the node of an agent-capability pair. $\mathsf{pf} : V \cup E \rightarrow \mathbb{N}$ determines the performance corresponding to a vertex or an edge such that for a node $v = \mathsf{node}(a, c)$, $\mathsf{pf}(v) = P_{cap}(a, c)$ and for an edge e that connects two nodes $v_1 = \mathsf{node}(a_i, c_k)$ and $v_2 = \mathsf{node}(a_j, c_l)$,

$$\mathsf{pf}(e) = \begin{cases} \mathsf{P}_{\mathsf{agent}}(a_i, \{c_k, c_l\}), & \text{if } e \text{ is } c\text{-edge} \\ \mathsf{P}_{\mathsf{team}}(\{\langle a_i, \{c_k\}\rangle, \langle a_j, \{c_k\}\rangle\}), & \text{if } e \text{ is } s\text{-edge} \\ \mathsf{P}_{\mathsf{team}}(\{\langle a_i, \{c_k\}\rangle, \langle a_j, \{c_l\}\rangle\}), & \text{if } e \text{ is } j\text{-edge} \end{cases}$$

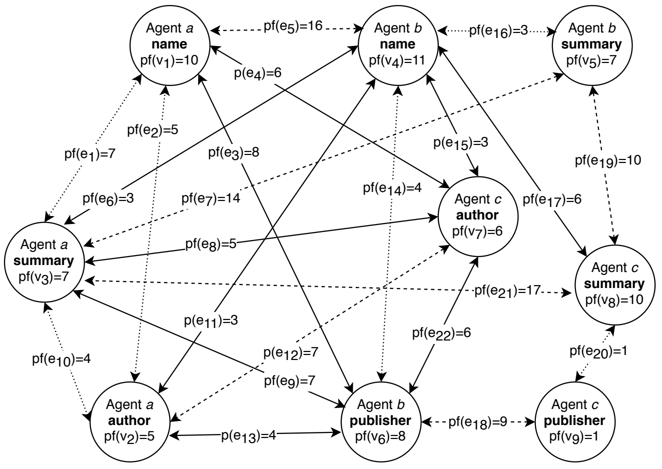

Figure 1. The expertise graph of the agents in Example 2.1

The expertise graph of the agents in Example 2.1 is constructed and presented in Figure 1, where the dotted lines denote *c*-edges, the dashed lines denote *s*-edges, and line edges denote *j*-edges. For instance, node v_1 corresponds to "name" capability of agent a and node v_6 corresponds to "publisher" capability of agent b. The edge between these two nodes represents their cooperation of agent a and b when they are assigned to capabilities specified as their nodes. The performance of this cooperation can be extracted from the edge, which is a *j*-edge, as $\mathsf{pf}(e_3) = 8$. Notice that some of the nodes do not have edges in between because those agents cannot work together when they are assigned to the corresponding capabilities. For instance, v_1 and v_9 do not have an edge in between because the agent a cannot work together with agent c when they perform capabilities "name" and "publisher", respectively.

The performance values that are stored in the graph as well as the graph properties can be combined to give meaning to different dynamics of working together. By defining appropriate metrics, it is possible to measure the cooperativeness, versatility, and so on and use these metrics as heuristics to build teams.

We define the following metrics on the expertise graph:

- *Cooperativeness* of a node measures how an agent a would perform c in relation to other capabilities performed by other agents. Note that the cooperativeness is not a metric of an agent but is specific to a particular capability. It is calculated based on *j*-edges of the node as follows:

$$\mathsf{cpt}(a, c) = \Big(\sum_{e \in E_j} \mathsf{pf}(e)\Big)/|E_j| \tag{1}$$

  where E_j is the set of *j*-edges connected to $\mathsf{node}(a, c)$.
- *Versatility* of a node measures how well an agent a can carry out other capabilities when it performs capability c. That is, it is an estimation of how an agent would perform in case of being responsible for multiple capabilities simultaneously. Again, versatility is not a metric of an agent as a whole but it is calculated for performing a particular capability. It is calculated as follows:

$$\mathsf{vst}(a, c) = \Big(\sum_{e \in E_c} \mathsf{pf}(e)\Big)/|E_c| \tag{2}$$

  where E_c is the set of *c*-edges connected to $\mathsf{node}(a, c)$.
- *Centrality* of a node in a graph denotes how well a node is linked to others in the graph. Contrary to above two metrics, here we are not concerned with the centrality of an assignment but the centrality of an agent. Hence, when calculating the centrality, we consider all the nodes that belong to the agent. Thus, the centrality becomes an estimation of how much an agent can be involved in teams with different agents.

$$\mathsf{dct}(a) = \Big(\sum_{v \in V} |\mathsf{edgesOf}(v)|\Big)/|V| \tag{3}$$

  where $V = \mathsf{nodesOf}(a)$ and $\mathsf{nodesOf} : A \rightarrow 2^V$ is a graph function that returns all nodes of a given agent. This measure corresponds to the average of degree centrality values [16] of all the nodes that belong to the agent.

Table 1. Graph metric values of the agents in Example 2.1

Agent	capability	cpt	vst	dct
a	name	2.80	6.00	5.67
a	author	1.40	4.5	5.67
a	summary	3.75	5.50	5.67
b	name	3.00	4.00	5.33
b	summary	0.00	2.00	5.33
b	publisher	5.00	2.00	5.33
c	author	4.00	0.00	6.33
c	summary	1.50	0.50	6.33
c	publisher	0.00	0.50	6.33

4 ALGORITHMS FOR BUILDING TEAMS

Team formation approaches usually focus on building a team from scratch in a way that one specific property of a team, such as communication [10], is maximized. Another interesting dimension is to start with an existing team for a task, which might be generated with a tool in hand and improve the team to yield a better team in an iterative manner. This requires a metric for a performance estimation of the generated team so that the algorithm can decide to continue to iterate to find a better team or to stop. Below, we first show how a simple algorithm for one shot team building can be used with the expertise graph. Next, we develop an iterative algorithm that improves an existing team.

4.1 One-Shot Team Building

Team building can be done in various ways using the expertise graph, such as selecting the agent that has the highest cooperativeness value for a required capability. Note that the fact that the agent has a high cooperativeness value does not guarantee that the agents will perform well in a team but increases the likelihood. This approach is generalized to make agents part of the team based on a given metric (Algorithm 1). $\mathsf{capables} : C \rightarrow 2^A$ is a function on the expertise graph, which returns all agents that can perform a given capability, such that $\mathsf{capables}(c) = \{a \mid a \in A \text{ and } c \in a_\mathsf{C}\}$ (line 3). After finding agents that can perform, it uses function max, which returns the agent having the highest metric value for the capability, to find which agent to assign (line 4). Finally, it adds the agent to the team by assigning to that capability (line 5).

```
Algorithm 1: buildTeam$(T, G, m)$
  Input: Task $T$, Expert Graph $G$, Metric $m$
  Output: Team $K$ in which metric $m$ is maximized
1 $K \leftarrow \emptyset$
2 foreach $c \in T$ do
3     $U \leftarrow \mathsf{capables}(c)$            // get all capables
4     $a \leftarrow \mathsf{max}(c, G, U, m)$         // find the highest
5     $K.add(a, \{c\})$
6 end
7 return $K$
```

The algorithm selects the best agents in terms of a given performance metric. By the means of the expertise graph, we can compare agents based on capability performance, cooperativeness value, and versatility. We introduce the corresponding teams as follows:

- HIP: Each capability is assigned to the agent that has the highest individual performance for the capability. The team is composed of the best agents based on their individual performance.
- HCPT: Each capability is assigned to the agent that has the highest cooperativeness value for the capability. The team is composed of agents that suit best to the cooperation with other agents for the assigned capability.
- HVST: Each capability is assigned to the agent that has the highest versatility value for the capability. The team is composed of agents that perform the best in case of being assigned to multiple capabilities.

4.2 Iterative Team Building

The team formation process can start with an initial team that has been already formed. However, the team may need further alterations for many reasons, such as improving the expected performance or exploring those that have not been used before.

4.2.1 Expected Team Performance

Since the individual capability performances and the pair-wise cooperation performances are known from the expertise graph, estimation of a team performance can be exactly calculated for teams of size two as each is assigned to a subtask of only one capability. However, it is still not possible to calculate the exact performance of bigger teams by only knowing the pair-wise performances. For bigger teams where only one agent is assigned to each capability, *expected* ranges of a team performance can be calculated to compare their performances. For query answering domain, an expected maximum performance for a conjunctive query is the maximum number of results that would be obtained, while the minimum performance is the minimum number of those results guaranteed to be returned.

The maximum performance of a team is bound by the cooperation performances that appear as the edge values. The edge with the lowest value is the bottleneck for the team such that the maximum performance of the team can at best be as much as the lowest of the edges. Thus, we estimate the maximum performance of the team by determining the minimum pair-wise cooperation performances value. Let $\mathsf{edges} : \mathcal{K} \rightarrow 2^E$ be a function that returns all edges covered by a team K. Then, $\mathsf{P_{max}} : \mathcal{K} \rightarrow \mathbb{N}$ returns the maximum number of results and is calculated as follows:

$$\mathsf{P_{max}}(K) = \mathsf{min}\{\mathsf{pf}(e) \mid e \in \mathsf{edges}(K)\} \tag{4}$$

Estimating the minimum performance of a team requires calculating the overlap between individual node performances and its associated edge performances through set intersection. We know the performance of each node v (e.g., the number of URIs it will return) of team K, though not the individual URIs. We represent this as a set s where the size matches the performance of the node. Similarly, we represent each edge e between the node v and the other nodes of the team as a set of edges E, where the set size matches the pair-wise performances for the assignment represented by the node v. Thus, we have $|E|$ sets, whose intersection yields the minimum number of results guaranteed to be returned, when compared with s. Note that the same intersection can be computed for each node in the team and the maximum of the estimations guarantees the minimum performance. Below, we show both computations.

Let $I : \mathcal{K} \times V \rightarrow \mathbb{N}$ be function that finds the minimum expected performance for a team K by taking an agent-capability pair represented by a node. That corresponds to the intersection of v with its edges and is calculated as follows:

$$I(K, v) = \sum_{e \in E} \mathsf{pf}(e) - (|E| - 1)\mathsf{pf}(v) \tag{5}$$

where $E = \mathsf{edges}(K) \cap \mathsf{edgesOf}(v)$. Let $\mathsf{nodes} : \mathcal{K} \rightarrow 2^V$ be a function that returns all nodes covered by a team K. Then, minimum expected performance $\mathsf{P_{min}} : \mathcal{K} \rightarrow \mathbb{N}$ can be calculated by using function I as follows:

$$\mathsf{P_{min}}(K) = \mathsf{max}\{I(K, v) \mid v \in \mathsf{nodes}(K)\} \tag{6}$$

Example 4.1. The nodes of the best performing team in Example 2.1 are $\mathsf{nodes}(K_2) = \{v_1, v_3, v_6, v_7\}$ and the edges of the team are $\mathsf{edges}(K_2) = \{e_1, e_3, e_4, e_8, e_9, e_{22}\}$. The maximum expected performance $\mathsf{P_{max}}(K_2) = 5$ because the minimum of edge values is equal to 5 for e_8. The result of Equation 5 is equal to 1 if we use v_1 as the input and it is equal to 5 if we use the other nodes, v_3, v_6, and v_7, as input. Equation 6 returns the maximum of those values as the minimum expected performance, which is $\mathsf{P_{min}}(K_2) = 5$. The expected performance boundaries of team K_2 are equal to each other and therefore, equal to the exact performance as well.

4.2.2 *Performance-Based Team Building*

The expected performance of the team may not be at satisfying levels because either the min performance is too poor or the max one is not promising enough. Rather than building a completely new team in these cases, the team can be altered in such a way that performance expectation changes in the intended direction. Algorithm 2 shows a procedure of altering a team to increase cooperativeness in the team. We know that performance is correlated with the pair-wise cooperative performances. Therefore, we can replace an assigned agent with an agent or set of agents that have better local cooperativeness in that specific team.

The algorithm finds first the average cooperativeness in a team (line 4) by using the Equation 1, where the set of edges $E = \mathsf{edges}(K)$ instead of all the edges in the graph. Then, it checks every subtask assignment by starting from the agent that has the minimum cooperativeness in the team. If it finds an alternative assignment for an agent-capability pair having cooperativeness value lower than the average, then it looks for alternative agents to replace the assignment. For an alternative agent, it is important to check if the agent has been a part of the team already. If that is the case, *versatility* of the possible new assignment has to be considered (line 15) rather than *cooperativeness* (line 13) to compare with others. Cooperativeness and versatility values are normalized by the node performance to compare alternatives based on their expected minimum performance in the team (see Equation 6). The agent with the highest normalized value is selected as an replacement (line 19). A team is repeatedly altered with the alternative agents as long as the expected performance improves, i.e., at least $\mathsf{P}_{\mathsf{min}}$ or $\mathsf{P}_{\mathsf{max}}$ increases.

Algorithm 2: $\mathsf{alterTeam}(K, G)$

```
Input: Team $K$, Expert Graph $G$
Result: Team $K$ is improved
1  while true do
2      $K_o \leftarrow K.copy()$                 // take a copy of team
3      $M \leftarrow \mathsf{teamCpt}(K, G)$        // cpt values in team
4      $avg \leftarrow \mathsf{avg}(M)$              // average cpt of team
5      foreach $v \in M$ do
6          $y \leftarrow M.\mathsf{getValue}(v)$        // cpt of $v$ in team
7          $a_o \leftarrow agent(v), c \leftarrow cap(v)$
8          if $y < avg$ then
9              $U \leftarrow \mathsf{capables}(c) \setminus \{a_o\}$
10             $M \leftarrow initOrderedMap$          // descending
11             foreach $a \in U$ do
12                 if not $K.has(a)$ and $\mathsf{cpt}(a, c) > avg$ then
13                     $M.add(a, \mathsf{cpt}(a, c)/\mathsf{pf}(\mathsf{getNode}(a, c)))$
14                 else if $K.has(a)$ and $vst(a, c) > avg$ then
15                     $M.add(a, \mathsf{vst}(a, c)/\mathsf{pf}(\mathsf{getNode}(a, c))$
16                 end
17             end
18             if $M \neq \emptyset$ then
19                 $K.replace(c, a_o, M.first())$
20             end
21         end
22     end
23     if $\mathsf{P}_{\mathsf{min}}(K) \leq \mathsf{P}_{\mathsf{min}}(K_o)$ and $\mathsf{P}_{\mathsf{max}}(K) \leq \mathsf{P}_{\mathsf{max}}(K_o)$ then
24         $K \leftarrow K_o$   // no improvement, last version
25         break
26     end
27 end
```

4.2.3 *Exploration-Based Team Building*

The above procedure does not take into account how often certain agents are used in teams as well as which agents have not been explored. This is an important concern when building teams as reusing the same set of agents might disrupt the flow of certain types of information. Furthermore, the agents that have not been used might provide information that is scarce and difficult to obtain otherwise. Hence, we provide a variation on Algorithm 2 that encourages exploration over performance. Notice that the exploration is not random and that still qualified agents are chosen for the team.

In Algorithm 2, agents are evaluated based on their performance (line 13 and 15). The algorithm always selects the agent with the highest performance (line 19). For most of the agents in a team, the individual performance exceeds the team performance (see Equation 4). This results in an extra performance that a team cannot get benefit from. In order to increase the exploration, the number of times the agents are consulted before is stored. This time, agents are added to the ordered map with their consultancy count ($M.add(a, getConsult(a, c))$). Then, from the map of candidate agents, which are expected to increase the team performance, the least consulted agent is selected ($M.last()$).

4.3 Qualities for Team Building

The performance of a team is the only required criteria for evaluating the team. However, in many settings, measuring other qualities of teams or the process of team building could be essential. We define three qualities and provide metrics to measure them.

Fairness: Fairness refers to the process of building teams, rather than the team itself. If certain agents are always chosen for the formed teams, some agents might never perform subtasks and thus "starve". A fair team building process would give chance to agents that can provide services to a team to take part. While fairness helps with unpopular agents to take part in teams, it also enables the teams to find these agents, enabling them to be reused more often in the future. Fairness can only be measured over the course of successive team buildings and based on individual subtasks. The algorithm is most fair, when each subtask for every team generated, is assigned uniformly to agents that can carry out the task. Standard deviation of the number of times that agents are assigned to perform their capabilities reflects to how uniform the assignment is. We measure the fairness of assignments for graph $F(G)$ as inverse of standard deviation.

Privacy: The subtasks that are carried out might reveal private information about the task owner. For example, the fact that certain pieces of data are being sought gives away the fact that the query owner is investigating the topic. It would be best if the task can be performed in a privacy-preserving manner, such that none of the subtask performers can leak information about the task to third parties. Intuitively, the more well-connected an agent is to others, the more likely it is to leak the information to others, yielding a high *privacy risk* [1, 5]. Given a team K, the agent that has the highest centrality poses the largest risk for privacy. We measure the privacy risk of a team $R(K)$ as follows:

$$R(K, G) = 1 - \frac{\max\{\mathsf{dct}(a) \mid a \in K\}}{\max\{\mathsf{dct}(a_g) | a_g \in G\}} \tag{7}$$

To mitigate this privacy risk, agents with lower connectivity values can be preferred over others. However, a simple swapping of two agents is not adequate as the relations of the new agent with the rest of the team are not accounted for.

Diversity: Another important quality of a team is *diversity*. Diversity of a team usually corresponds to the difference of individuals. Since in our case, each agent performs a different subtask in a team, the variation in subtasks is already in place. However, another important aspect of diversity is the different groups or communities from which assignments in a team come from. Note that this diversity does not refer to the diversity of individuals, but to the assignment of agents for subtasks. Current community detection algorithms [7] when applied on the expertise graph can identify the existing communities. We measure the diversity of a team as the number of assignments that come from different communities, e.g., a diversity of 1 denoting the minimum diversity where all assignments are from the same community.

5 EVALUATION

In order to evaluate the proposed algorithms, we have created an experimental setup, which is inspired by the case study. The requested task is a query, which describes books. However, this time there are ten different properties of books, corresponding to agents' capabilities. Agents provide varying number of books as an answer to tasks that require those capabilities. The capabilities are distributed to agents to mimic the fact that few agents are experts in a lot of capabilities, whereas many agents are experts in a few capabilities. More precisely, three agents have ten capabilities, four agents have nine capabilities, and so on, yielding a total of 52 agents.

The extent of the agents' capabilities correspond to the number of different books that agents return as answer for a query corresponding to the capability. For each capability, an expert is assigned to know between 10 to 100 different books from a pool of 200 books. When the number is larger than 50, the books are assigned randomly from the 200 books. For agents that know less than 50 books, books are assigned from one of two subsets each containing 50 of the 200 books. This ensures to have variations between the agents in such a way that the agents that know a lot are generalists, whereas the agent that know little are specialists. That means the agents that know less are more likely to have complete knowledge about a special area.

We have assigned the agents in the above setup to primitive tasks (tasks of single capabilities) to observe their individual and pair-wise cooperation performances. The resulting performances yield an expertise graph with 296 nodes and 43622 edges. The average value of nodes is 52.68, whereas the average value of edges is 24.27, more specifically 16.57 for j-edges, 16.41 for c-edges, and 89.18 for s-edges. Comparing the values of nodes to the values of edges, especially j-edges, shows that performance of agents would significantly decrease if they work together with arbitrarily assigned others. The expertise graph even lacks edges between some of the nodes because these agents cannot work together for the corresponding capability assignments.

Using the above setup, we evaluate how teams can be built. In order to evaluate team performances, we create tasks having vary from size three to seven. For each size, we have created ten different tasks that each require different set of capabilities to be performed. Each task is first evaluated with three one-shot teams formed by Algorithm 1, namely HIP, HCPT, and HVST. Next, the same set of tasks are evaluated with the two iterative algorithms, namely Performance-Based and Exploration-Based. When doing so, we first create 100 teams for each task by assigning one of the capable agents to each required capability randomly. This process results in 1000 different teams for each task size. Those teams are used by Algorithm 2 as input.

Table 2 shows the resulting performances for each task size and algorithm. Notice that as the task size increases, the performance of each algorithm decreases. This is expected as the performance corresponds to the average number of results returned and as the task includes more subtasks, the possible results are fewer. These tasks correspond to conjunctive queries and therefore, it is an expected outcome that the more constraints are introduced, the fewer number of books satisfy all the constraints. When comparing the individual performance of the algorithms, we see that for the smallest task size (3), the best results are obtained by HIP, while HCPT's performance is comparable. The proposed iterative approaches do not perform as well. However, starting from task size 4, performance-based iterative algorithm outperforms all the rest consistently. The main reason for this is that when the task size is large building a team by just adding "ideal" agents in do not capture the relations among them. It is necessary to consider how the team will perform as a whole and update the team when an agent does not fit the team.

Table 2. Performance of teams for different task sizes

	One-Shot			Iterative	
Size	HIP	HCPT	HVST	Pf-based	Exp-based
3	21.44	20	12.4	16.78	12.88
4	9.6	10.5	5.7	14.54	6.82
5	5	5.6	2.6	10.83	3.79
6	2.3	3.4	1.3	7.71	2.21
7	1.3	2	0.9	5.98	1.41

We explore the performance evolution of performance-based iterative algorithm more in Table 3. The columns under *initial* are the average performance of the randomly constructed teams at the beginning of the iteration process while the columns under *final* are the average performance of the improved teams at the end of the iterative process. It is important to note that the exact performance of initial teams are calculated only to compare the results and those values have not been used in the algorithm. The algorithm has been called repeatedly to improve a team only based on the expected performance, which always guarantees that the actual performance will be in the boundaries. We see that performance expectation of the resulting teams is given with a wider window. This is an expected outcome as there are more relations to account for in bigger teams.

Table 3. Performance evolution of teams

	Initial			Final		
Size	P_{min}	P_{max}	exact	P_{min}	P_{max}	exact
3	2.54	10.37	6.14	12.64	22.24	16.78
4	0.5	8.29	2.80	10.55	22.75	14.54
5	0.06	6.35	1.21	6.54	20.43	10.83
6	0	5.45	0.57	2.88	18.73	7.71
7	0	4.60	0.27	1.53	17.79	5.98

Lastly, we compare the two iterative algorithms in terms of fairness. We have calculated the fairness over number assigned tasks and presented the results in Figure 2. Exploration-based algorithm is expected to increase the fairness since it prefers least consulted agents in alternatives. Performance-based algorithm assigns better performing agents more than the others. Thus, intuitively, we expect some agents to be left at a disadvantage. This is reflected in the results where exploration-based algorithm achieves higher fairness than performance-based algorithm. Moreover, with performance-based algorithm the decrease in fairness is sharper whereas with the exploration-based algorithm the decrease is slow. Overall, even though the exploration-based approach does not attain as high performance as the performance-based approach, it can still form teams that perform relatively good while being more fair.

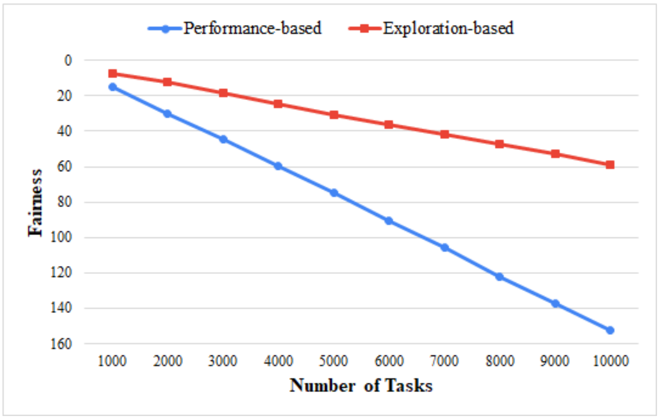

Figure 2. Fairness of iterative algorithms over tasks

6 DISCUSSION

The problem of team formation has been traditionally considered as an optimization problem, where subtasks are assigned to ideal candidates to carry out the task [2, 13]. Variations on this include addition of a budget for communication or agent costs. For example, Kargar *et al.* [10] tackle the problem of finding a team of experts from a social network to complete a task under a budget. Communication cost is considered to be pair-wise, such as physical distance between experts. They assume those values are independent and thus the communication cost of a team is calculated as the sum of distances. They consider both personnel and communication costs together and represent the problem as a bicriteria optimization problem, which is NP-hard. Two approaches are proposed: finding a team with bounded budget, which is an approximation algorithm, and finding Pareto optimal teams, which returns a set of teams in which each team is guaranteed to be not dominated by any other team in terms of personnel and communication costs. They evaluate the proposed algorithms on IMDB dataset and show that those are effective and efficient algorithms.

Rangapuram *et al.* [19] model the problem as a weight matching problem in a weighted bipartite graph where the weight between each pair of agents reflects their degree of compatibility to jointly solve tasks. These weights are updated along multiple encounters between agents. In the formalization, a task is given as a set of triples that specifies required amount for each skill to finish the given task. The proposed algorithm aims to maximize the collaborative compatibility and satisfy the skill requirement, budget constraints, bound on the team size and distance based constraint. When a team is formed, every agent is assumed to perform every skill it has. While these approaches provide the best assignment for agents to subtasks in terms of performance, they do not explicitly consider the relations among subtasks, the context in which the teams are formed, as well as the team building properties, such as privacy preserving or fairness [3].

More recent approaches to team formation consider factors apart from the optimization of tasks, such as the synergy of the team or the personal traits of the individuals. Liemhetcharat and Veloso [12] propose a model that learns capabilities of agents and constructs a synergy graph among them to solve the team composition problem using previous joint experiences. They define a synergy model as a graph where the distance between agents is an indicator of how well they work together. They form teams from capable agents to maximize team's internal synergy. Andrejczuk *et al.* [4] represent agents with personal properties and tasks with information on required congeniality, competence, and so on. They partition a given set of agents into teams using minimum costs so that within a task all competence requirements are covered and team members work well together. They propose a model that considers competence levels and personality measures together to compose synergistic teams. Peleteiro *et al.* [17] try to capture the quality of the solutions of team tasks via a model that besides using skills and compatibility between agents (called the strength of collaboration synergies within coalitions), calculates the reputation of teams (coalitions) as a whole and of single agents. These reputation values are used by the team composition process.

Our work differs from these approaches in that we consider how well two agents work together on a given subtask as well as the possible degradation of performance when multiple subtasks are performed by a single agent. Further, we develop methods for forming teams by modifying teams. With our proposed iterative algorithm, we can start with teams with low performance and improve them to yield performances that are far better than individually assigning agents to tasks. The proposed qualities for team building can be used further to measure and compare teams. We use the expertise graph as the representation of knowledge. The expertise graph could be readily available from the system. Alternatively, the expertise graph can be constructed at once, as we do in the experiments, or at run time as information becomes available. Whenever a new agent joins the network, it is possible to extend the graph by observing the performance of the new agent. The team building algorithms do not require the graph to contain the complete knowledge of the pair-wise performance values. Using a partial expertise graph is possible but the expected performance results are more accurate with the complete graph.

This work opens up interesting directions for further research. Currently, we are reusing existing teams that are available for the exact same task at hand. However, many times the requested task could resemble an existing task but still be different. Quantifying the similarity of the task, deciding on which previous team to reuse, and adapt the team appropriately are all important challenges to tackle. Further, our current team compositions assign a single subtask to each agent. However, it is possible in teams to have multiple agents to work on the same subtask. It would be possible to exploit the s-edges in the expertise graph to devise teams that collaborate on subtasks.

ACKNOWLEDGEMENTS

This work was done in the context of the Golden Agents project, funded by the Netherlands Organization of Science NWO – Large Investments program.

REFERENCES

[1] Jose Alemany, Elena Del Val, Juan M Alberola, and Ana Garćia-Fornes, 'Metrics for privacy assessment when sharing information in online social networks', *IEEE Access*, **7**, 143631–143645, (2019).

[2] Aris Anagnostopoulos, Luca Becchetti, Carlos Castillo, Aristides Gionis, and Stefano Leonardi, 'Online team formation in social networks', in *Proceedings of the 21st International Conference on World Wide Web*, pp. 839–848. ACM, (2012).

[3] Ewa Andrejczuk, Rita Berger, Juan A Rodriguez-Aguilar, Carles Sierra, and Víctor Marín-Puchades, 'The composition and formation of effective teams: computer science meets organizational psychology', *The Knowledge Engineering Review*, **33**, (2018).

[4] Ewa Andrejczuk, Filippo Bistaffa, Christian Blum, Juan A Rodriguez-Aguilar, and Carles Sierra, 'Synergistic team composition: A computational approach to foster diversity in teams', *Knowledge-Based Systems*, (2019).

[5] Michael S Bernstein, Eytan Bakshy, Moira Burke, and Brian Karrer, 'Quantifying the invisible audience in social networks', in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 21–30. ACM, (2013).

[6] Christian Bizer, Tom Heath, and Tim Berners-Lee, 'Linked data: The story so far', in *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, 205–227, IGI Global, (2011).

[7] Michelle Girvan and Mark EJ Newman, 'Community structure in social and biological networks', *Proceedings of the National Academy of Sciences*, **99**(12), 7821–7826, (2002).

[8] Birte Glimm, Carsten Lutz, Ian Horrocks, and Ulrike Sattler, 'Conjunctive query answering for the description logic shiq', *Journal of Artificial Intelligence Research*, **31**, 157–204, (2008).

[9] Michael N Huhns and Munindar P Singh, 'Service-oriented computing: Key concepts and principles', *IEEE Internet Computing*, **9**(1), 75–81, (2005).

[10] Mehdi Kargar, Morteza Zihayat, and Aijun An, 'Finding affordable and collaborative teams from a network of experts', in *Proceedings of the 2013 SIAM International Conference on Data Mining*, pp. 587–595. SIAM, (2013).

[11] Hiroaki Kitano and Satoshi Tadokoro, 'Robocup rescue: A grand challenge for multiagent and intelligent systems', *AI Magazine*, **22**(1), 39–52, (2001).

[12] Somchaya Liemhetcharat and Manuela Veloso, 'Modeling and learning synergy for team formation with heterogeneous agents', in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 365–374. International Foundation for Autonomous Agents and Multiagent Systems, (2012).

[13] Kathryn Sarah Macarthur, Ruben Stranders, Sarvapali Ramchurn, and Nicholas Jennings, 'A distributed anytime algorithm for dynamic task allocation in multi-agent systems', in *Twenty-Fifth AAAI Conference on Artificial Intelligence*, (2011).

[14] Leandro Soriano Marcolino, Albert Xin Jiang, and Milind Tambe, 'Multi-agent team formation: diversity beats strength?', in *Twenty-Third International Joint Conference on Artificial Intelligence*, (2013).

[15] Tenda Okimoto, Nicolas Schwind, Maxime Clement, Tony Ribeiro, Katsumi Inoue, and Pierre Marquis, 'How to form a task-oriented robust team', in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pp. 395–403. International Foundation for Autonomous Agents and Multiagent Systems, (2015).

[16] Tore Opsahl, Filip Agneessens, and John Skvoretz, 'Node centrality in weighted networks: Generalizing degree and shortest paths', *Social networks*, **32**(3), 245–251, (2010).

[17] Ana Peleteiro, Juan C Burguillo, Michael Luck, Josep Ll Arcos, and Juan A Rodríguez-Aguilar, 'Using reputation and adaptive coalitions to support collaboration in competitive environments', *Engineering Applications of Artificial Intelligence*, **45**, 325–338, (2015).

[18] Bastian Quilitz and Ulf Leser, 'Querying distributed RDF data sources with sparql', in *European Semantic Web Conference*, pp. 524–538. Springer, (2008).

[19] Syama Sundar Rangapuram, Thomas Bühler, and Matthias Hein, 'Towards realistic team formation in social networks based on densest subgraphs', in *Proceedings of the 22nd International Conference on World Wide Web*, pp. 1077–1088. ACM, (2013).