Allocating teams to tasks: an anytime heuristic competence-based approach[⋆]

Athina Georgara[1,2], Juan A. Rodríguez-Aguilar[1], and Carles Sierra[1]

[1] Artificial Intelligence Research Institute, CSIC, Barcelona, Spain
[2] Enzyme Advising Group, Barcelona, Spain

Abstract. Many practical applications often need to form a team of agents to solve a task since no agent alone has the full set of required competencies to complete the task on time. Here we address the problem of distributing individuals in non-overlapping teams, each team in charge of a specific task. We provide the formalisation of the problem, we encode it as a linear program and show how hard it is to solve it. Given this, we propose an anytime heuristic algorithm that yields feasible team allocations that are good enough solutions. Finally, we report the results of an experimental evaluation over the concrete problem of matching teams of students to internship programs in companies.

Keywords: Team Formation· Task allocation· Optimisation· Heuristics

1 Introduction

Many real-world problems require allocating teams of individuals to tasks. For instance, forming teams of robots for search and rescue missions [6], forming teams of Unmanned Aerial Vehicles (UAVs) for surveillance [28], building teams of people to perform projects in a company [31], or grouping students to undertake school projects [4]. In this paper, we study the allocation of many teams to many tasks with size constraints, permitting no overlaps. That is, each agent can be part of at most one team, each team can be allocated to at most one task, and each task must be solved by at most one team. We illustrate our results in the domain of education, where it is very common that students shall form teams and collaborate with their teammates towards some common goal. For example, in primary and secondary schools teachers usually need to divide their students into study groups (teams) to carry out some school projects. Similarly, in universities, students are usually requested to work in teams in order to carry out semester projects. Moreover, educational authorities often need to form student teams and match them with internship programs, as it is more and more common that students spend time with companies to gain experience in the industry. Currently, teachers and education authorities obtain such allocations mainly by hand, but given the combinatorial nature of the problem,

[⋆] Research supported by projects AI4EU (H2020-825619), TAILOR (H2020-952215), 2019DI17, Humane-AI-Net (H2020-952026), Crowd4SDG (H2020-872944), and grant PID2019-104156GB-I00 funded by MCIN/AEI/10.13039/501100011033.

manual allocation requires a large amount of work. Beyond the activities within classrooms, the problem of allocating non-overlapping teams to tasks can also be found in events and competitions where participants need to work in teams and compete with each other, such as hackathons; or in situations where different teams need to work in parallel towards a common goal, and individuals cannot be in more than one team at a time, such as in search and rescue missions.

The multi-agent systems (MAS) literature has tackled the problem of allocating teams to tasks in several ways. The existing literature includes research on how to form a *single team* and allocate it to a *single* task [1,23,2]; how to form a *single team* and match it with *multiple tasks* [11]; and how to form *multiple teams* to solve the *very same* task [4]. Moreover, there is a handful of research works on forming *multiple teams* to match with *multiple tasks*, either by allowing *agent overlaps* (agents participate in multiple teams [6]), and/or *task overlaps* (different teams jointly solve a task [5]). However, the problem of distributing agents in non-overlapping teams, each to solve a different task, has deserved little attention, with the exception of [12,29,30]. This *non-overlapping many teams to many tasks* (NOMTMT) allocation problem is the one we address in this paper.

In most works, regardless of the type of team allocation problem, the allocation is decided based on due to agents' competences. As noted in [4], the literature on team composition and formation considers either a Boolean model of competences (an agent has or has not a competence) [1,23,2,12], or a graded model (an agent has a competence up to some degree) [3,4,7]. In many cases, competences are not explicitly considered, but they are 'concealed' behind some utility function [6,5,29]. Common to all these models is the assumption that a team assigned to a task must possess the competences *exactly* as required by the task. This is rather limiting to cope with real-world problems. For instance, even if a student does not posses a specific competence, they might still be qualified for a task if any of their already-acquired competences is similar enough. However, the semantic relationship between competences has been disregarded when matching teams to tasks. This prevents, for instance, that a team is allocated to a task requiring competences *similar* to those offered by the team. Against this background, here we make headway in the non-overlapping "many teams to many tasks" matching problem through the following novel contributions:[3]

1. A method for computing the semantic matching between a task and a team based on an ontology of competences.
2. A formalisation of the NOMTMT allocation problem as an optimisation problem together with a complexity analysis.
3. An integer linear programming (ILP) encoding for solving optimally the NOMTMT allocation problem.
4. An anytime heuristic algorithm to solve the NOMTMT allocation problem.
5. A threefold empirical evaluation: *(a)* we compare our heuristic algorithm against CPLEX [20] using synthetic data, and show that it outperforms CPLEX in terms of solving time; *(b)* we use real-world data from students that must be allocated to internships, and show that our heuristic algorithm solves large

[3] This work is an extended version of our earlier work presented in [16,17].

problem instances that CPLEX cannot handle; and *(c)* a group of experts in education confirm that the allocations produced by our heuristic algorithm are better than those manually produced by experienced teachers.

2 The NOMTMT Allocation Problem

This section formally casts our problem as an optimisation one. To do so, we first refer to the basic concepts of the problem, then we discuss the competence model to be used, and finally we define the problem as an optimisation problem.

2.1 Basic Concepts

A *competence* corresponds to a specified capability, skill, or knowledge. We assume there is a known, predefined and fixed set of competences, denoted by $\mathcal{C}$. A *task* is characterised by a set of requirements on agents' competences and team size constraints. For instance, an internship program in a computer tech company might require three competences (ML principles, coding in Python, and web development), and a team of size four. Thus, the company needs four employees that together possess the three required competences. In general, there might be further constraints, such as temporal or spatial constraints (i.e., when and where the task can be realised). However, within the scope of this paper, we only focus on team size constraints. The competences' relative importance is often part of the task description. Formally, a *task* τ is a tuple $\langle \text{t\_id}, C, w, s \rangle$, where t_id is a unique task identifier, $C \subseteq \mathcal{C}$ is the set of required competences, $w : C \to (0, 1]$ is a function that weighs the importance of competences, and $s \in \mathbb{N}_+$ is the required team size. The set of all tasks to perform is denoted by T, with $|T| = m$. We describe each *agent* via its acquired competences. Thus, an agent a is given by a tuple $\langle \text{a\_id}, C' \rangle$, where a_id is a unique agent identifier, and $C' \subseteq \mathcal{C}$ is a set of acquired competences. The set of agents is denoted by A, with $|A| = n$. Given $\tau \in T$, we denote the set of all size-compliant teams for τ as $\mathcal{K}_\tau = \{K \subseteq A : |K| = s_\tau\}$[4], where s_τ is the team size required by task τ.

2.2 Competence coverage and affinity

To match a team with a task, it is essential that the team is capable of solving the task. That is, before allocating a team of agents K to some task τ, we need to verify whether the agents, as a team, are equipped with the necessary competences, as determined by C_τ. Given a task τ and a team of agents K, we say that K is suitable for τ if K can *cover the required competences* of τ. That is, for each required competence $c \in C_\tau$, there is at least one agent $a \in K$ with competence c. As mentioned above, the existing literature considers competences as Boolean or graded features and determines the 'matching quality' of a team

[4] Note: we use subscript a to refer to the set of competences and the identifier of an agent $a \in A$, and subscript τ to refer to the elements of task $\tau \in T$.

when assigned to a task through some function, usually expressed as a utility function. As already mentioned, the existing models are limiting. To overcome such limits we consider the matching quality in terms of the semantic similarity between competences required by tasks and offered by teams. Here we present an intuitive way of determining this 'matching quality' as *competence coverage*.

Given some domains, competences are usually structured by an ontology which determines similarities between different competences. For example, competences c_1 (coding in Python) and c_2 (coding in Java), are different, but share essential principles (e.g. both are object-oriented languages). We can therefore assume that an agent with competence c_1 can somewhat be adequate for a task requiring competence c_2.

We assume that there is a known competence ontology which structures the competences in $\mathcal{C}$ according to their semantics, and for every pair of competences c, c' provides a *similarity degree* $\mathrm{sim}(c, c') \in [0, 1]$. Let assume that the ontology is structured as an acyclic directed graph, where each node is a specialised, refined version of its parent node. In section 5, we exploit well-established ontologies with such properties. We compute the semantic similarity between two competences c, c' as: $\mathrm{sim}(c, c') = e^{-\lambda l} \frac{e^{\kappa h} - e^{-\kappa h}}{e^{\kappa h} + e^{-\kappa h}}$ if $l \neq 0$ and 1 otherwise, where: l is the shortest path in the graph between c and c'; h is the depth of the deepest competence subsuming both c and c'; and $\kappa, \lambda \in [1, 2]$ are parameters regulating the influence of l and h on the similarity metric. Our semantic similarity function is a variation of the metric introduced in [24], which guarantees the reflexive property of similarity: a node is maximally similar to itself, independently of its depth. In other words, nodes at zero distance ($l = 0$) have maximum similarity. Similarly to [27], the values of semantic similarities lie in $[0, 1]$.

Given this, we assume that an agent a can cover competence c with degree $\mathrm{cvg}(c, a) = \max_{c' \in C_a} \mathrm{sim}(c, c')$. Then, given a task τ with required competences C_τ and an agent a with acquired competences C_a, the competence coverage of task τ by agent a is: $\mathrm{cvg}(a, C_\tau) = \prod_{c \in C_\tau} \mathrm{cvg}(c, a) = \prod_{c \in C_\tau} \max_{c' \in C_a} \{\mathrm{sim}(c, c')\}$. The product captures a's competence coverage over *all* competences.

Moving now from a single agent to a team of agents, allocating a team to a task requires to solve a *competence assignment problem*.[5] That is, given a task τ, we need to assign to each agent a in a team K a subset of competences of C_τ for which it will be responsible. As such, for each pair $\langle \tau, K \rangle$ we need a *competence assignment function* $\eta_{\tau \to K} : K \to 2^C_\tau$ that maps each agent in K with a subset of the required competencies. According to [4], a competence assignment function (CAF) for a size-compliant team of agents $K \in \mathcal{K}_\tau$ and a task τ is such that $\bigcup_{a \in K} \eta_{\tau \to K}(a) = C_\tau$. Moreover, we consider the reverse competence assignment function (r-CAF), denoted by $\theta_{\tau \to K} : C_\tau \to 2^K$, where $\theta_{\tau \to K}(c)$ indicates the agents that are assigned to cover competence c. Note that $\Theta_{\tau \to K}$ contains $|C_\tau| \cdot (2^{|K|-1})$ different CAFs. However, not all CAFs are equivalent or equally desired. Hence, here we adopt the concept of *fair competence assignment function (FCAF)* following the notion of *inclusive assignments* in [3] by adding

[5] As noted by [22], recent definitions on the term team refer to the specific subtask/ competences that will be performed by each agent.

an upper bound on the number of competences that can be assigned to an agent. An FCAF ensures that each competence required by the task τ is covered by at least one agent, and each agent covers at least one and at most $\left\lceil \frac{|C_\tau|}{|K|} \right\rceil$ competences. This bound avoids overloading a few very competent agents with excessive responsibilities.

Therefore, given an FCAF $\eta_{\tau \to K}$, we want to evaluate the suitability of a given team K for task τ. To do so, we first define a team's *competence affinity* with respect to a task taking into consideration the importance of each competence and the agents' assigned competences while satisfying the following requirements: *(i)* the higher the coverage of an assigned competence, the higher the competence affinity; *(ii)* the lower the importance of an assigned competence, the higher the competence affinity; and *(iii)* the competence affinity is at most equal to the coverage of any assigned competence with maximal importance. Formally, we define an agent's competence affinity as:

Definition 1 (*Agents' Competence Affinity*). Given an agent $a \in A$, a task $\tau \in T$, and a competence assignment function $\eta_{\tau \to K}$, the competence affinity of a to τ is:

$$\mathrm{aff}(a, \tau, \eta_{\tau \to K}) = \prod_{c \in \eta_{\tau \to K}(a)} \max \left\{ \left(1 - w_\tau(c)\right), \mathrm{cvg}(c, a) \right\}. \tag{1}$$

Since we are targeting FCAFs, i.e., we want balanced assignments of responsibilities, the competence affinity of a team of agents $K \subseteq A$ to task τ is defined a-la-Nash, as the product[6] of the competence affinity of the individuals in K to τ with respect to some fair competence assignment function $\eta_{\tau \to K}$ The product assigns a larger value to teams where all agents equally contribute to a task, rather than to teams with unbalanced contributions.

Definition 2 (*Team's Competence Affinity*). Given a team of agents $K \subseteq A$, a task $\tau \in T$, and a fair competence assignment $\eta_{\tau \to K}$, the competence affinity of K to τ is:

$$\mathrm{aff}(K, \tau, \eta_{\tau \to K}) = \prod_{a \in K} \mathrm{aff}(a, \tau, \eta_{\tau \to K}). \tag{2}$$

Observe that the competence affinity of a team to a task depends on the competence assignment function. In other words, for a given team K and a given task τ, different competence assignment functions result in different competence affinities. Finding the competence assignment function that yields the highest competence affinity is in fact an optimisation problem in itself:

$$\eta^*_{\tau \to K} = \underset{\eta \in \Theta_{\tau \to K}}{\arg\max}\ \mathrm{aff}(K, \tau, \eta) \tag{3}$$

where $\Theta_{\tau \to K}$ denotes the family of all CAFs for task τ and team K.

However, considering that, in practice, for a task τ both team size s_τ and the number of required competences $|C_\tau|$ are relatively small ($2 - 5$ members, and ≤ 10, respectively), solving this sub-problem optimally, e.g. by means of

[6] As noted in [9], the product favours both increases in overall team allocation utility and inequality-reducing distributions of team values.

a linear program, is rather inexpensive. With this in mind, the optimum team for task τ is the one that: *(i)* maximises competence affinity; and *(ii)* satisfies the team size requirement. Note that the size of $\mathcal{K}_\tau$, which is the set of all size-compliant teams, is $\binom{n}{s_\tau}$. The optimum team K^* is the one that maximises the competence affinity, under an optimal competence assignment function: $K^* = \arg\max_{K \in \mathcal{K}_\tau} \operatorname{aff}(K, \tau, \eta^*_{\tau \to K})$.

2.3 The Optimisation Problem

Finding a good allocation of agents for a collection of tasks is yet another optimisation problem that tries to maximise the *overall* competence affinity of all teams for their assigned tasks. For a single task τ, the best candidate would be the team that maximises the competence affinity, that is, $K^* = \arg\max_{K \in \mathcal{K}_\tau} \operatorname{aff}(K, \tau, \eta^*_{\tau \to K})$. For a collection of tasks T, with $|T| > 1$, we must maximise the competence affinity of all candidate teams with the tasks each one is matched to, given that each agent can participate in at most one team, each team can be allocated to at most one task, and each task can be assigned to at most one team. First we need to formally define what is a Feasible Team Allocation Function (FTAF), and then proceed on finding the optimum one, i.e., the one that maximises the competence affinity.

Definition 3 (***Feasible Team Allocation Function (FTAF)).*** Given a set of tasks T, and a set of agents A, a feasible team allocation function g is a function $g : T \to 2^A$ such that: (1) every task $\tau \in T$ is allocated its requested number of agents, so that $|g(\tau)| = s_\tau$; and (2) an agent can only be assigned to one team: for every pair of tasks $\tau, \tau' \in T$, such that $\tau \neq \tau'$, it holds that $g(\tau) \cap g(\tau') = \emptyset$.

The family of all feasible team allocation functions is denoted by G. To achieve balanced allocations, the optimum team allocation function g^* maximises the product of competence affinities of the teams to their assigned tasks.

Definition 4 (***Non-Overlapping Many Teams to Many Tasks (NOMTMT) Allocation Problem).*** Given a set of tasks T, and a set of agents A, the *Non-Overlapping Many Teams to Many Tasks Allocation Problem* is to find a team allocation function $g^* \in G$ that maximises the overall team affinity:

$$g^* = \arg\max_{g \in G} \prod_{\tau \in T} \operatorname{aff}(g(\tau), \tau, \eta^*_{\tau \to g(\tau)}) \qquad (4)$$

Note that in a NOMTMT allocation problem, for each team allocation $g \in G$ and each task τ we need to find the competence assignment function with the highest competence affinity for team $g(\tau)$, namely $\eta^*_{\tau \to g(\tau)}$. Thus, for each team allocation we need to solve $|T|$ optimisation problems (one per task) in order to determine $\eta^*_{\tau \to g(\tau)}$.[7] Here we want to highlight that the problem we address here

[7] Note that the NOMTMT allocation problem is interrelated with the $|T|$ optimisation problems. However, for a fixed team allocation, the inner optimisation problems are independent from one another.

is a non-trivial generalisation of the problem tackled in [4], which unlike us only copes with forming teams for a single task. Next, we show that the NOMTMT allocation problem is $\mathcal{NP}$-complete by reduction to a well-known problem.

Theorem 1. *The NOMTMT allocation problem for more than one task is $\mathcal{NP}$-complete.*

Proof. Due to space limitations, the proof can be found in: https://bit.ly/3A2uoSK

Notably, the problem we solve here can be cast as a cooperative game [8] where the agents and the tasks correspond to the players—with the constraint that exactly one task-player must exist in each coalition—and the competence affinity comprise the game's utility function. Therefore, we would seek for the a coalition structure that maximizes the *Nash Social Welfare* [26].

3 Solving the NOMTMT allocation problem optimally

Here we encode the NOMTMT allocation problem (Def. 4) as a linear program. First, for each task $\tau \in T$ and each size-compliant team $K \in \mathcal{K}_\tau$, we use a binary decision variable x_K^τ to indicate whether team K is assigned to task τ in the solution. Then, solving a NOMTMT allocation problem amounts to solving the following non-linear program:

$$\max \prod_{\tau \in T} \prod_{K \in \mathcal{K}_\tau} \left(\mathrm{aff}(K, \tau, \eta^*_{\tau \to K})\right)^{x_K^\tau} \tag{5}$$

subject to:

$$\sum_{K \subseteq \mathcal{K}_\tau} x_K^\tau \leq 1 \qquad \forall \tau \in T \tag{5a}$$

$$\sum_{\tau \in T} \sum_{\substack{K \subseteq \mathcal{K}_\tau \\ a \in A}} x_K^\tau \leq 1 \qquad \forall a \in A \tag{5b}$$

$$x_K^\tau \in \{0, 1\} \qquad \forall K \subseteq A, \tau \in T \tag{5c}$$

Constraints (5b) ensure that each agent will be assigned to at most one task; while constraints (5a) guarantee that each task is assigned to at most one team. Notice that the objective function (Eq (5)) is non-linear. Nevertheless, we linearise it by maximising the logarithm of $\prod_{\tau \in T} \prod_{K \in \mathcal{K}_\tau} \left(\mathrm{aff}(K, \tau, \eta^*_{\tau \to K})\right)^{x_K^\tau}$. Thus, solving the non-linear program above is equivalent to solving the following binary linear program:

$$\max \sum_{\tau \in T} \sum_{K \in \mathcal{K}_\tau} x_K^\tau \cdot \log\left(1 + \mathrm{aff}(K, \tau, \eta^*_{\tau \to K})\right) \tag{6}$$

subject to: equations (5a), (5b), and (5c). Note that the above is an equivalent optimisation problem: without affecting the monotonicity of the function *(i)* we use the $\log(\cdot)$ to convert the double product to double sum, and the powered factor into a product; and *(ii)* we change the function's domain to avoid $\log(0)$. We can solve this LP with the aid of an off-the-shelf solver (e.g. CPLEX [20], Gurobi [19]), GLPK [18], or SCIP [15]). Given sufficient time, an LP solver will return an optimal solution to the NOMTMT allocation problem.

Note that building such an LP requires to pre-compute the values of $\mathrm{aff}(K, \tau, \eta^*_{\tau \to K})$, which amounts to solving an optimisation problem for each pair of team and task. This leads to large LPs as the number of agents and tasks grow.

4 An algorithm for the NOMTMT allocation problem

Our proposed algorithm consists of two stages in a similar manner as in [4]—as we already said, our problem is a generalisation. The first stage finds an initial feasible allocation of teams to tasks. The second one iteratively improves the allocation by swapping agents between pairs of teams using different strategies.

4.1 Building an initial team allocation

The algorithm finds an initial, feasible, and *promising* team allocation. It sequentially picks up a team for each task, starting from the 'hardest' task to the 'lightest' one. We consider that a task is 'hard' if there are just a few agents that can cover its competences. Picking teams for the harder tasks first is a heuristic to avoid that the few agents that can cover it are picked by other 'simpler' tasks.

Computing the allocation hardness of tasks. We measure the *allocation hardness* of each task (referred as 'hardness' hereafter) by considering the competences required by the task with respect to the capabilities of *all* available agents. Intuitively, the more agents offering high coverage of competence c, the less hard a task requiring c is. Specifically, to characterise the hardness of competences, and therefore the hardness of tasks, we exploit the notion of *moment of inertia* based on [25]. We measure the hardness of a task as the hardness to cover its competences based on the agent's competences. That is, each agent can cover each competence with an affinity in range $[0, 1]$. Thus, we capture the effort to cover a competence as best as possible similar to the effort to rotate a rigid body around some axis. In other words, we see the distribution of all agents' coverage of a competence as the mass distribution of a rigid body. In our case, the chosen axis to rotate around represents the ideal competence coverage, i.e., where all agents cover the competence with utmost affinity. We compute the moment of inertia for c as: $I(c) = \sum_{J \in \mathcal{I}} n^c_J \cdot \left(1 - mid(J)\right)^2$, where: (i) $\mathcal{I} = \{[0, 0.1), [0.1, 0.2), [0.2, 0.3), [0.3, 0.4), [0.4, 0.5), [0.5, 0.6), [0.6, 0.7), [0.7, 0.8), [0.8, 0.9), [0.9, 1]\}$ is an interval partition of the domain of competence coverage $[0, 1]$; (ii) $n^c_J = |\{a \in A | \mathrm{cvg}(c, a) \in J\}|$ is the number of agents in A whose coverage of competence c lies within interval J, and hence represents the *mass* of c in the interval; and (iii) $mid(J)$ corresponds to the midpoint of interval J.

Now, we compute the hardness of each task from the hardness of each one of the competences that it requires (inversely proportional to the moment of inertia of its competences) as well as their relative importance weights. Thus, given task τ, we define its hardness as $h(\tau) = \omega \cdot \sum_{c \in C_\tau} w_\tau(c) \cdot I(c)$, where $\omega = \frac{1}{\sum_{c \in C_\tau} w_\tau(c)}$, is a normalising factor over the weights.

Building an initial team allocation. Our algorithm sorts tasks according to their hardness and proceeds by sequentially allocating a team for each task starting from the hardest one. Let $A_\tau \subseteq A$ be the set of available agents to allocate to τ. First, the algorithm sorts the task's competences, C_τ, based on their relative importance, into a sequence $\bar{C}_\tau$. We note as $\bar{C}^i_\tau$ the i-th competence in $\bar{C}_\tau$. To allocate an agent to the top competence in the sequence, $\bar{C}^1_\tau$, we select the agent that best covers the competence. Formally, we compute the agent to allocate to $\bar{C}^1_\tau$ as $\sigma(\bar{C}^1_\tau) = \arg\max_{a \in A_\tau}\{\text{cvg}(\bar{C}^1_\tau, a)\}$. After allocating that agent to $\bar{C}^1_\tau$, the set of agents available to allocate to the rest of competences is $A_\tau - \{\sigma(\bar{C}^1_\tau)\}$. In general, given the i-th competence $\bar{C}^i_\tau$, we obtain the agent to allocate to the comptence as: $\sigma(\bar{C}^i_\tau) = \arg\max_{a \in A_\tau - \Sigma_{i-1}}\{\text{cvg}(\bar{C}^i_\tau, a)\}$, where $\Sigma_{i-1} = \bigcup_{k=1}^{i-1}\{\sigma(\bar{C}^k_\tau)\}$ stands for the agents allocated so far up to competence $\bar{C}^{i-1}_\tau$. The selected team for task τ is $K = \bigcup_{i=1}^{s_\tau} \sigma(\bar{C}^i_\tau)$. The agents in K are no longer available for being chosen to participate in another team.

4.2 Improving team allocation

The second stage of our algorithm applies several heuristics implemented as *agent swaps*. This stage is similar to the approach proposed in [4], with the addition of an exploring step. The heuristics are applied until either: (1) the global maximum competence affinity is reached; (2) no solution improvement occurs for a number of iterations; or (3) the algorithm is stopped by the user. In all cases, the most recently found solution is returned. This stage performs two types of iterations:

1. **Single pairing.** We randomly select two tasks, and we apply over them the following swaps:
   (a) **Exploiting swap.** Find the optimal team allocation just considering the agents in the teams currently allocated to both tasks.
   (b) **Exploring swap.** Try a maximum of k times the following: (i) randomly select one of the two tasks, one agent within that task and an unassigned agent (if any); (ii) swap them; (iii) if the competence affinity is improved, keep the change and stop the exploring swaps.
2. **Exhaustive pairing.** For *every* pair of tasks, swap every possible pair of agents within them. If the competence affinity is improved, keep the change and stop the exhaustive pairing.

5 Empirical Analysis

We evaluated our algorithm regarding: (1) the quality of solutions; (2) the time required to produce optimal solutions; (3) its performance when solving a real-

Scenario	**Time Savings *wrt.* CPLEX(%)**
Small (10 Tasks)	60%
Medium (15 Tasks)	55%
Large (20 Tasks)	71%

Table 1: Time savings to reach optimality *wrt.* CPLEX.

world problem. Importantly, our algorithm was validated by experts on team formation by comparing the allocations computed by our algorithm with respect to the allocations provided by teachers with expertise in team formation.

We ran all the experiments on a PC with Intel Core i7 CPU, 8 cores, and 8GB RAM. The implementation of our algorithm, along with all the necessary supporting code, was made in Python3.7. In all experiments, we set our algorithm's parameters as follows: to compute similarities we used $\kappa = 0.35, \lambda = 0.75$; we performed one exhaustive-pairing every 50 single-pairings; we stopped the algorithm after completing two rounds of 50 single-pairings and after two rounds of exhaustive pairings elapsed with no improvements. In what follows, in Section 5.1, we pitch our algorithm against CPLEX to study its quality, as well as its runtime and anytime performance. In Section 5.2, we solve a real-world problem and study our algorithm's behaviour as the team-size parameter changes. Finally, in Section 5.3, we compare the quality of our algorithm's allocations with the ones obtained by experts in team formation.

5.1 Quality, runtime and anytime analysis

Generating problem instances. In this analysis, we used as competence ontology the taxonomy developed by the Institute for the Development of Vocational Training for Workers (ISFOL) [21]. For comparison purposes, we built 3 families of problem instances of different sizes (small, medium, large) that could all be solved by CPLEX within acceptable time limits. We synthetically generated agents, competencies and tasks as follows. First, we generated the tasks to perform. We started by fixing a number of tasks from $\{10, 15, 20\}$. After that, for each task τ we sampled: its required team size $m_\tau \sim \mathcal{U}\{1, 3\}$; its number of required competencies $|C_\tau| \sim \mathcal{U}\{2, 5\}$; and its importance weights $c \in C_\tau$ is $w_\tau(c) \sim \mathcal{N}\big(\mu = \mathcal{U}(0, 1), \sigma \sim \mathcal{U}(0.01, 0.1)\big)$. Second, we generated agents to perform tasks. For each task τ, we generated m_τ agents such that the competencies of each agent contain competencies that are either identical or a child-node in the ISFOL taxonomy of some required competence in τ. Our experiments involve 60 problem instances distributed in three families: (1) 20 instances with 10 tasks and $\sim$20.5 agents (average number of agents over 20 problem instances); (2) 20 instances with 15 tasks and $\sim$30.6 agents (average over 20); and (3) 20 instances with 20 tasks and $\sim$41.35 agents (average over 20).

Quality analysis. Figure (1a) shows the evolution of the *quality* of our heuristic algorithm calculated as the ratio between the competence affinity of the solutions computed by our algorithm and the optimal competence affinity computed by CPLEX. The figure plots the average of the quality ratio achieved by our algorithm along time over 20 problem instances per scenario: low-size (10 tasks), medium-size (15 tasks), large-size (20 tasks). Variances for all cases are insignificant, $\leq 5 \cdot 10^{-4}$, and hence we do not plot them. The timestamps are also averages over the 20 problem instances. Notice that our heuritic algorithm reaches optimality (quality 1), likewise CPLEX, in the three scenarios.

Runtime analysis. The greatest advantage of our heuristic algorithm is that it is much faster than CPLEX. Table 1 shows the time we can save with respect to CPLEX to reach optimality. Overall, using our heuristic can save from ~55% to ~71% time with respect to CPLEX. Specifically, for problem instances with 10 tasks (small scenario), we save 60% time *wrt.* CPLEX; for problem instances with 15 tasks (medium scenario), we save 55% time; and, for problem instances with 20 tasks (large scenario) we save 71% time *wrt.* CPLEX. Note that the main time consuming task for CPLEX is building of the LP encoding the problem.

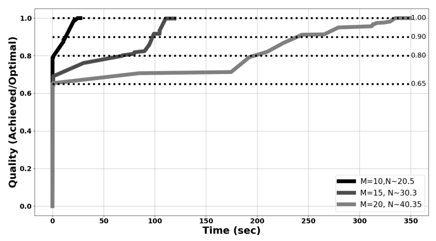
Anytime analysis. Let t_{opt} be the seconds required by CPLEX to reach optimality. Our algorithm finds the first solution: (1) after $10^{-3} \cdot t_{opt}$ (= 0.06) seconds with a quality at 80% of the quality of the optimal in the small scenario, (2) after $25 \cdot 10^{-3} \cdot t_{opt}$ (= 0.12) seconds and 70% in the medium scenario, and (3) after $2 \cdot 10^{-4} \cdot t_{opt}$ (= 0.29) seconds and 65% in the large scenario (see Table 2).

5.2 Solving a real-world problem

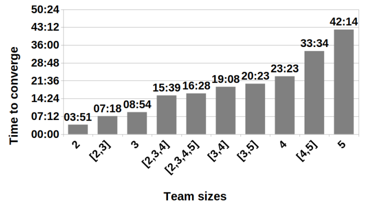
As mentioned earlier, in the domain of education there is a need to allocate teams of students to internship programs. Each student is equipped with competencies, determined through the student's educational background (type of school, enrolment year, completed courses, past educational activities, etc.), while each internship specifies a set of required competencies for the team. In this part of the experimental analysis, we used *real-world data.* Specifically, we count on a collection of 100 students whose competencies are described in the European Skills/Competences qualifications and Occupations (ESCO) [13] ontology. ESCO

Scenario	**Quality**	**Time in sec**	**Proportion of t_{opt}**
Small (10 Tasks)	80%	0.06 sec	10^{-3}
Medium (15 Tasks)	70%	0.12 sec	$25 \cdot 10^{-3}$
Large (20 Tasks)	65%	0.29 sec	$2 \cdot 10^{-4}$

Table 2: Quality of the initial solution, the time needed in seconds, and the proportion of time compared to the time required by CPLEX (t_{opt}).



(a) Our algorithm: Competence quality *wrt* time.



(b) Time required *wrt* team size

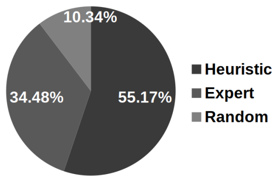
Fig. 1

consists of a dictionary that describes, identifies and classifies professional occupations, skills, and qualifications relevant for the EU labour market and education and training. The ESCO ontology is a directed acyclic graph structure of 6547 different competencies, with 7 levels, and an average branching factor of 1.26 (maximum branching factor 15). The collection of students in our dataset is equipped with 118 different competencies, on average with 11.98 each. We also used a collection of 50 real internship programs, whose competencies are also described in ESCO. All 50 internships required 34 different competencies, while each internship required on average 4 competencies.

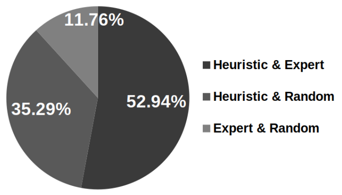
Our following analysis shows the problem's scalability as required team sizes grow, and investigates the ability of our algorithm to handle the problem. Due to the fact that the actual data regarding tasks (internship programs) did not specify a required team size, we synthetically created problem instances of certain team sizes. Specifically, we used datasets where all tasks required the same team size, i.e., where all tasks required *equal* team sizes (either 2, 3, 4, or 5). These team sizes are based on the following observation in [3]: "teams that are formed within an educational environment shall not exceed 5 members." Moreover, we also created problem instances where tasks required *varying* team sizes (team sizes in $[2, 3]$, in $[2, 4]$, in $[2, 5]$, in $[3, 4]$, in $[3, 5]$, and in $[4, 5]$), where the team sizes are equally distributed across the tasks within each problem instance. Consider the scenario requiring teams of equal size 5 and 100 agents. The search space has $\sim 7.5 \cdot 10^7$ different teams of size 5. To solve such a hard scenario instance optimally (e.g., by using CPLEX), we would need to produce ~75 millions of decision variables just for a single internship with team size 5. Thus, generating the LP encodings for the problem instances considered here is totally infeasible.
Analysis. Figure (1b) shows the time required to converge to a solution as the average over 20 different problem instances with 100 agents and 20 tasks. The bars illustrate the average time (in minutes:seconds) needed by our algorithm to output an allocation per team size. As expected, settings with smaller team sizes require much less time until they converge to a solution. In general, the time needed by a problem instance requiring team sizes in $[a, b]$ falls between the times needed *(a)* by the problems requiring team sizes a and b, and *(b)* by the problems requiring team sizes in $[a, b-1]$ and $[a+1, b]$. Notably, we need less than 50 minutes to yield a solution in settings where each task requires a team of size 5, which is the hardest scenario. We deem this is acceptable considering that this process is not required to run in real time with very demanding time constraints. Note that the current practice is to match students to internships by hand, which is much more time consuming, while LP solvers cannot even generate the program in time. Hence, our results show the feasibility of employing our algorithm to perform team allocation in the education scenario that we address.

5.3 Validation

Our last analysis focuses on having our algorithm validated by teachers experienced in team formation. For that, we pitched our algorithm against some experts (teachers) with experience in allocating students to internships. Specifically, we



(a) **Single Winner**: Percentage out of 29 tournaments.



(b) **Tie with 2 Winners**: Percentage out of 17 tournaments.

synthesised an instance of a task allocation problem involving 50 internships ($m = 50$) and 100 students ($n = 100$) with team sizes within $\{1, 2, 3\}$. Notice that such settings are similar to those employed in our actual-world evaluation in Section 5.2. The problem instance used here is the largest one regarding the number of tasks that we can generate with the 100 student profiles at hand. Notably, to solve this problem optimally (e.g. by using CPLEX) would require more than 1.8 million decision variables.

Thereafter, we proceed as follows. For the very same problem instance: (1) we task an expert with matching by hand teams of students with tasks; (2) we employ our algorithm to compute an allocation; and (3) we compute a random allocation of teams of students to tasks. Henceforth, we note those three allocation methods as g_{expert}, $g_{\text{heuristic}}$, and g_{random} respectively. Then, eight evaluators (teachers as well), who are regularly engaged with the process of allocating student to internships, were tasked to compare the quality of the three allocations, without knowing the method that produced each allocation. Notably, our algorithm yielded an allocation in less than 1 hour and 45 minutes, while the experts reported that they approximately needed a whole working week in order to study and analyse the students and internship data, and manually build an allocation.

Evaluation Process. Each evaluator was asked to mark the internship assignments produced by each one of the three allocation methods. Thus, each evaluator marked each internship assignment with one of the following marks: 1 for first option, 2 for second option, and 3 for third option. Notice that we allowed the evaluators to mark two assignments produced by two different allocation methods with the same value if they considered them to be equivalent.

Handling missing data Here we want to point out that during this final analysis, we faced the problem of missing data. That is, the expert did not manage to find a team for every internship. Specifically, the expert did not provide a team assignment to 13 internships (out of 50), leaving 23 students (out of 100) without internship. This led the evaluators work with incomplete data (two complete allocations, and a partial one), and, in their turn, provide incomplete evaluations. In particular, since for some tasks g_{expert} was missing, the evaluators were unable to mark the three allocation methods (g_{expert}, $g_{\text{heuristic}}$, and g_{random}). For this reason we used the auxiliary mark 4 indicating absence, which is considered worse than third option (mark 3). Therefore, any missing allocation was marked with a 4 by all evaluators. Moreover, eventually the evaluators missed marking some internships (different interships for each evaluator). In that case, we gen-

erated a third-option mark (3) for missing evaluations.
Analysis. Our analysis is founded on finding the best allocation method for each internship assignment based on the evaluators' assessments. We consider the evaluation of each internship assignment as a *tournament* consisting of three competing rounds between pairs of allocation methods: (1) Heuristic vs Expert; (2) Heuristic vs Random; (3) Expert vs Random.

The marks set by evaluators allow to pick the winning allocation method of each round and of the tournament as a whole.[8] The winning allocation method of each round results from the aggregated marks of evaluators: the internship assignment with greater aggregated mark wins one point for its allocation method. In case there is a tie between two internship assignments in a round, their corresponding allocation methods earn half a point each. Using the points accumulated from each round of a tournament, we apply a Copeland$_\alpha$ voting rule [10] (with $\alpha = 0.5$) to declare the winner of the tournament. As shown in [14] this voting rule is "resistant to all the standard types of (constructive) electoral control". In short, the allocation method that accumulates more points throughout the three rounds wins the tournament. Again, in case of a tie between two allocation methods, each one earns half a point. As an illustratory example, say that for a given tournament: the 8 evaluators considered that our heuristic algorithm provided the best assignment, 5 evaluators considered that the human expert provided a better assignment than random, and 2 evaluators equally preferred the assignments produced by the human expert and random. That would lead to the following scores: our heuristic algorithm would get $8 \cdot 1$ points, the expert's allocation would get $5 \cdot 1 + 2 \cdot 0.5 = 6$ points, and random would get $2 \cdot 0.5 + 1 \cdot 1 = 2$ points. Therefore, the winner of this tournament would be our algorithm.

Each tournament may have *a single winner*, *a tie with two winners*, or *no winner*. In our evaluation, we encounter 58% over 50 tournaments (i.e., internship assignments) that announced a single winner, and 34% that announced two winners in a tie. Considering only the tournaments that announce a **single winner**, in Figure 2a we observe that 55.17% of these tournaments announced as winner the allocation yielded by our heuristic algorithm, while 34.48% of the tournaments announced as winner the allocation provided by the human expert. The random allocation method only won 10.34% of the tournaments. Therefore, the evaluators preferred the assignments produced by our algorithm to those produced by a human expert. Consider now the tournaments declaring **2 winners (tie)**. Figure 2b shows that, as expected, our heuristic algorithm and the human expert jointly won more than half of the tournaments (52.94%). Overall, regarding the tournaments declaring a tie with two winners, our heuristic algorithm was part of the winning tie 88.23% of the times. To summarise, our analysis indicates that expert evaluators deem our proposed heuristic algorithm as the method of choice to assign teams of students to internships.

[8] Notably, the marks applied by the evaluators indicate rankings and therefore these numbers are meaningless; thus we turn to tournaments.

6 Related Work

Team formation has received much attention by the AI and MAS community. Anagnostopoulos et al. in [1] thoroughly study the problem of forming a single team to resolve a single task, and show the employability of several algorithms in large scale communities. Lappas et al. in [23] tackle the problem of finding a single team of experts for a given task in an attempt to minimize the communication cost among the team. [2] study an online version of the team formation problem and propose algorithms in order to form teams as a stream of tasks sequentially arrives (one task at a time). Notably, [2] form a single team for a single task at a time; while agents can be 'reused' in teams of different tasks, permitting overlapping teams. Kurtan et al. [22] study the dependencies between subtasks of a given task, and propose algorithms for building a single team for a single task considering some desired qualities, such as preserving privacy.

Chad et al. [11] add a new dimension to the problem by considering robustness, and focus on finding a single robust team to perform several tasks. Andrejczuck et al. [4] tackle the many teams to single task problem and present algorithms for partitioning a set of agents into equal-size teams in order to perform resolve the very same task. Capezzuto et al. in [6] tackle the many teams to many tasks team formation problem considering temporal and spacial constraints, and propose an anytime, efficient algorithm. However, compared to the problem we tackle here, the proposed algorithm in [6] provides solutions with overlapping teams, and aims to maximise the number of tasks solved per team.

Regarding the many teams to many tasks team formation problem with no overlaps—i.e., problems for which different teams share no common agents, each team can be allocated to only one task, and each task can be assigned to only one team—we can find a handful of works in the literature. Specifically, we have singled out two works, namely [12,29], which can be considered as the most directly related to ours. Although these papers tackle the general *many teams to many tasks* problem, their version of the problem is essentially different to ours, hence preventing us from conducting meaningful comparisons. In more detail, [29] propose a branch-and-bound technique to determine the optimal team size structure and then they proceed with a brute-force search. Given that the problem we tackle in this work assumes that team sizes are known a priori (team size is part of each task's requirements), comparing against [29] would be equivalent to compare against brute-force search. Notably, brute-force search becomes prohibitive as the number of agents and tasks rise; and considering the problem instances in our analysis such a comparison would be infeasible.

On the other hand, Czatnecki and Dutta [12] propose an algorithm for matching non-overlapping teams of robots with tasks. Similarly to [29], [12] sets no constraints on the team sizes. However, even if we could 'bypass' the team size misalignment (by allowing [12] to yield a result, and use these team sizes in our version), there is yet another essential difference between [12] and our approach. Our algorithm pursues to *optimise* the competence affinity between all teams with their assigned tasks while targeting at balanced allocations (i.e., all teams are more or less equally competent for their task). Instead, [12] targets at finding *Nash stable* teams, i.e., teams whose agents have not incentive to unilaterally abandon their current team and task without harming the others. As such, [12] and our approach differ notably in their objectives.

Regardless of the type of team allocation problem, all the works above use a rather simplistic competence model. That is, following the observation in [3], all these works assume either that an agent may have or have not a competence (Boolean) [1,23,2,12,22]; or that an agent may have a competence up to a degree (Graded) [3,4,7]. Nonetheless, all works consider that a team must collectively possess all the required competences, *exactly as requested*. However, in this work we identify that an agent, and therefore a team, can perform a task when they count on competences that are *similar* to the ones required, even if they are not exactly the same. This is natural, especially when the agents correspond to humans. Given that ontologies such as ESCO [13] describe semantic relations among competences, not having a specific competence for a tasks is not an obstacle provided that agents have similar enough competences. For example, when students move from school to industry, they count on competences, acquired at the school, which are not exactly the same as those required by industry. And yet, these student can be considered adequate for jobs in industry. As such, in this work we put forward a methodology to resolve such issues.

7 Conclusions and future work

In this work, we studied a particular type of team formation problem, and hence we focused on the *Non-Overlapping Many Teams to Many Tasks* (NOMTMT) allocation problem. First, we provided the formulation of the problem. At this point, we identified and tackled an existing issue regarding the competence models that we find essential when solving real-world cases. As such, we introduced a new ontology-based competence model, and proposed a methodology to compute semantic similarities between the competencies required by a task and those offered by a team. Then, we cast the NOMTMT allocation problem as an optimisation one, and show how to solve optimally it by the means of LP. Thereafter, motivated by the practical limitations of solving the problem optimally, we introduced a novel anytime, heuristic algorithm. Finally, we conducted a three-fold evaluation of our proposed algorithm. Our results: *(i)* showed that our heuristic algorithm can reach optimality in notably less time than an LP solver, saving up to 71% of time; *(ii)* showed that our algorithm can handle large, real-world problems with 100 agents and 20 tasks in less than an hour, while solving the problem optimally is infeasible; and *(iii)* our algorithm outperformed experts while requiring much less time (one hour and half vs a whole working week). Notably, besides the problem's size, another time consuming factor for the human experts is the need of manually discerning the similarities between the competences required by a task and those offered by a team. As future work, we plan to relax our team size constraints, and use instead allowable intervals—e.g., at least 2 and at most 5 members—since, these assumptions are not fundamental to our model. Moreover, we will address the notion of "robustness", and work towards not only forming good allocations, but also forming robust allocations.

References

1. Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., Leonardi, S.: Power in unity: Forming teams in large-scale community systems. pp. 599–608 (01 2010)
2. Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., Leonardi, S.: Online team formation in social networks. In: Proceedings of the 21st International Conference on World Wide Web. p. 839–848. WWW '12, Association for Computing Machinery, New York, NY, USA (2012)
3. Andrejczuk, E., Berger, R., Rodríguez-Aguilar, J.A., Sierra, C., Marín-Puchades, V.: The composition and formation of effective teams: computer science meets organizational psychology. Knowledge Eng. Review **33**, e17 (2018)
4. Andrejczuk, E., Bistaffa, F., Blum, C., Rodríguez-Aguilar, J.A., Sierra, C.: Synergistic team composition: A computational approach to foster diversity in teams. Knowledge-Based Systems **182**, 104799 (2019)
5. Bachrach, Y., Meir, R., Jung, K., Kohli, P.: Coalitional structure generation in skill games. vol. 2 (01 2010)
6. Capezzuto, L., Tarapore, D., Ramchurn, S.D.: Anytime and efficient coalition formation with spatial and temporal constraints (2020)
7. Chalkiadakis, G., Boutilier, C.: Sequentially optimal repeated coalition formation under uncertainty. Autonomous Agents and Multi-Agent Systems **24**(3), 441–484 (May 2012)
8. Chalkiadakis, G., Elkind, E., Wooldridge, M.: Computational Aspects of Cooperative Game Theory (Synthesis Lectures on Artificial Inetlligence and Machine Learning). Morgan & Claypool Publishers, 1st edn. (2011)
9. Chevaleyre, Y., Dunne, P., Endriss, U., Lang, J., Lemaitre, M., Maudet, N., Padget, J., Phelps, S., Rodrguez-Aguilar, J., Sousa, P.: Issues in multiagent resource allocation. Informatica **30**, 3–31 (2006)
10. Conitzer, V., Sandholm, T.: Complexity of manipulating elections with few candidates. In: Eighteenth National Conference on Artificial Intelligence. p. 314–319. American Association for Artificial Intelligence, USA (2002)
11. Crawford, C., Rahaman, Z., Sen, S.: Evaluating the efficiency of robust team formation algorithms. In: Osman, N., Sierra, C. (eds.) Autonomous Agents and Multiagent Systems. pp. 14–29. Springer International Publishing, Cham (2016)
12. Czatnecki, E., Dutta, A.: Hedonic coalition formation for task allocation with heterogeneous robots. In: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC). pp. 1024–1029 (2019)
13. ESCO: European skills, competences, qualifications and occupations. https://ec.europa.eu/esco/portal (2010)
14. Faliszewski, P., Hemaspaandra, E., Hemaspaandra, L.A., Rothe, J.: Llull and copeland voting broadly resist bribery and control. p. 724–730. AAAI'07, AAAI Press (2007)
15. Gamrath, G., Anderson, D., Bestuzheva, K., Chen, W.K., Eifler, L., Gasse, M., Gemander, P., Gleixner, A., Gottwald, L., Halbig, K., Hendel, G., Hojny, C., Koch, T., Le Bodic, P., Maher, S.J., Matter, F., Miltenberger, M., Mühmer, E., Müller, B., Pfetsch, M.E., Schlösser, F., Serrano, F., Shinano, Y., Tawfik, C., Vigerske, S., Wegscheider, F., Weninger, D., Witzig, J.: The SCIP Optimization Suite 7.0. Technical report, Optimization Online (March 2020), http://www.optimization-online.org/DB_HTML/2020/03/7705.html
16. Georgara, A., Rodríguez-Aguilar, J.A., Sierra, C.: Towards a Competence-Based Approach to Allocate Teams to Tasks, p. 1504–1506. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2021)

17. Georgara, A., Rodríguez-Aguilar, J.A., Sierra, C., Mich, O., Kazhamiakin, R., Palmero Approsio, A., Pazzaglia, J.C.: An anytime heuristic algorithm for allocating many teams to many tasks. In: Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems. p. 1598–1600. AAMAS '22, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2022)
18. GLPK: Glpk: Gnu linear programming kit). https://www.gnu.org/software/glpk/ (2018)
19. GUROBI: Gurobi optimizer 8.0. https://www.gurobi.com/ (2018)
20. IBM: Ibm ilog cplex optimization studio 12.10. https://www.ibm.com/us-en/marketplace/ibm-ilog-cplex (2019)
21. ISFOL: Professioni, occupazione, fabbisogni. https://fabbisogni.isfol.it (2017)
22. Kurtan, C., Yolum, P., Dastani, M.: An ideal team is more than a team of ideal agents. In: ECAI (2020)
23. Lappas, T., Liu, K., Terzi, E.: Finding a team of experts in social networks. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 467–476. KDD '09, Association for Computing Machinery, New York, NY, USA (2009)
24. Li, Y., Bandar, Z.A., Mclean, D.: An approach for measuring semantic similarity between words using multiple information sources. IEEE Transactions on Knowledge and Data Engineering **15**(4), 871–882 (July 2003)
25. Morrison, R.W., De Jong, K.A.: Measurement of population diversity. In: Collet, P., Fonlupt, C., Hao, J.K., Lutton, E., Schoenauer, M. (eds.) Artificial Evolution. pp. 31–41. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
26. Nguyen, T.T., Rothe, J.: Minimizing envy and maximizing average nash social welfare in the allocation of indivisible goods. Discrete Applied Mathematics **179**, 54–68 (2014)
27. Osman, N., Sierra, C., Mcneill, F., Pane, J., Debenham, J.: Trust and matching algorithms for selecting suitable agents. ACM Trans. Intell. Syst. Technol. **5**(1), 16:1–16:39 (Jan 2014)
28. Ponda, S.S., Johnson, L.B., Geramifard, A., How, J.P.: Cooperative Mission Planning for Multi-UAV Teams, pp. 1447–1490. Springer Netherlands, Dordrecht (2015)
29. Präntare, F., Heintz, F.: An anytime algorithm for simultaneous coalition structure generation and assignment. In: Miller, T., Oren, N., Sakurai, Y., Noda, I., Savarimuthu, B.T.R., Cao Son, T. (eds.) PRIMA 2018: Principles and Practice of Multi-Agent Systems. pp. 158–174. Springer International Publishing, Cham (2018)
30. Präntare, F., Heintz, F.: An anytime algorithm for optimal simultaneous coalition structure generation and assignment. Autonomous Agents and Multi-Agent Systems **34**(1), 1–31 (2020)
31. Sa Silva, I.E., Krohling, R.A.: A fuzzy sociometric approach to human resource allocation. In: 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). pp. 1–8. IEEE (2018)