Team Formation: Balancing Task Coverage and Expert Workload

ABSTRACT

In the classical team-formation problem the goal is to identify a team of experts such that the skills of these experts cover *all* the skills required by a given task. Based on this original formulation different variants of the problem have been studied with the majority of them focusing on complete coverage of tasks by the assigned experts.

In this paper, we deviate from this setting and we propose a variant of the problem, where all the skills of every task need not be covered; rather we aim to cover the skills of every task as well as possible, while also trying to minimize the maximum workload among the experts. Instead of setting the coverage constraint and minimizing the maximum load, we combine these two objectives into one objective function and we call the corresponding assignment problem the Balanced Coverage problem. We show that this problem is NP-hard. However, the nature of its objective function, which may also take negative values, does not allow us to design approximation algorithms with multiplicative guarantees. For that, we adopt a weaker notion of approximation [9, 19] and we show that under this notion we can design a polynomial-time algorithm with provable approximation guarantees. We also describe a set of computational speedups that we can apply to the algorithm and make it scale for reasonably large datasets. From the practical point of view, we also demonstrate how the nature of the objective function allows us to efficiently tune the two parts of the objective and tailor their importance to a particular application. Our experiments with a variety of real datasets demonstrate the practical utility of our problem formulation as well as the efficacy and the efficiency of our algorithm in practice.

ACM Reference Format:
. 2022. Team Formation: Balancing Task Coverage and Expert Workload. In *Proceedings of 31st ACM International Conference on Information and Knowledge Management (CIKM '22).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The abundance of online and offline labor markets (such as Amazon mechanical turks, Guru, Freelancer, scientific collaborations etc.) has motivated a lot of work on *team formation* in the data-mining community. The common characteristics of the majority of prior research are the following: there is a task (or a collection of tasks) that requires a set of skills, and a set of experts who also have a set of skills. The goal is to identify a "*good*" team (i.e., a subset of the experts) such that experts in the team collectively cover the skills required by the tasks – i.e., for every skill of every task there is at least one expert with this skill assigned to the task. Thus, the majority of the work on team formation requires *complete* coverage of the skills of the input tasks [1–3, 7, 10–17, 23, 25]. Their differences lie in the way they define the "goodness" of a team. For example, in some cases they optimize for the communication cost, while in other cases they optimize for the load of the experts or their compensation cost.

CIKM '22, Oct 17–22, 2022, Atlanta, GA
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In this paper, we deviate from this complete-coverage framework and we propose a team-formation problem where the goal is to assign experts to a set of input tasks such that we maximize the task coverage while, at the same time, we minimize the maximum load among the experts used. Hence, we do not aim to cover the skills of every task completely, but rather we want to cover a large fraction of the skills of the input tasks. Also, given that overworked experts do not perform well we penalize for expert overloading by trying to minimize the maximum number of tasks assigned to an expert. Therefore, for an assignment A of experts to tasks our goal is to maximize the combined objective:

$$F(A) = C(A) - L_{\max}(A), \tag{1}$$

where $C(A)$ is the sum of the fraction of the skills of the tasks being covered by their assigned experts and $L_{\max}(A)$ is the maximum number of tasks a single expert has been assigned to. We call this problem Balanced Coverage and we show that it is NP-hard.

From the application point of view, the Balanced Coverage problem makes sense because often skills in tasks are repetitive. For example, consider a task with the following skills: *advertising, internet advertising, facebook advertising, online marketing, social network platforms.* Clearly these skills are repetitive and not all of them need to be covered. Additionally, minimizing the maximum expert workload is desirable for better team performance.

Although we show that the two terms of the objective are comparable, we believe that there may be applications where coverage is more important than maximum load or vice versa. Therefore, we expand our framework with a *balancing coefficient* which enables an effective tuning of the importance of the two parts of the objective. We extensively discuss the impact of this coefficient in our algorithm design and in our experiments.

From the algorithmic point of view, optimizing the above objective is a challenging task. First of all, the function itself may potentially take negative values. Therefore, it does not admit multiplicative approximation algorithms. Although the coverage part of the objective ($C()$) is a monotone submodular function, the maximum load part does not have a predictable form (i.e., it is not linear or convex). Therefore, existing techniques [9, 19], which have been developed recently, cannot be applied. However, what we can adopt from these works is a weaker notion of approximation they propose. In that case the goal is to find an assignment A such that:

$$C(A) - L_{\max}(A) \geq \alpha C(\text{OPT}) - L_{\max}(\text{OPT}), \tag{2}$$

where OPT is the optimal solution to the Balanced Coverage problem and $\alpha \leq 1$ is an approximation guarantee that better fits functions like ours. In this paper, we show that we can design a polynomial-time algorithm with $\alpha = (1 - 1/e)$, which is probably the best we can hope for our objective given that the $C()$ is monotone and submodular. We also show that our algorithm admits a lot of practical speedups, which are a consequence of the structure of our objective function.

Our experimental results demonstrate that our algorithm is practical in terms of its running time due to the computational speedup methods we propose. Additionally, our algorithm outputs assignments with high task coverage and very low maximum load. When compared with a number of baselines inspired by existing work, we show that our algorithm consistently outperforms them.

There is some other recent work that also formalizes the team formation problem with partial coverages [8, 21, 22]. Although we discuss the connection between those works and ours in the related-work section (Section 2), we point out some key differences here. Dorn and Dustar [8] consider the single-task team-formation problem and their goal is to balance the task coverage and the team's communication cost on a graph; we assume no graph and therefore their methods are not related to ours. Nikolakaki et al. [22] also consider the partial coverage problem for a single task and their goal is to balance task coverage and the cost of the team – where the cost is the *summation* of the costs of its members. Therefore, they employ a framework for optimizing a submodular minus a linear function [9, 19]. Our objective function is different because $L_{\max}$ is not a linear function and we also consider *multiple* tasks. Finally, Nikolakaki et al. [21] in their arXiv paper propose a minimization version of our problem – their goal is to minimize load and uncovered fractions of tasks. Posing the problem as a minimization problem requires the design of different approximation algorithms. Moreover, the algorithms they propose are all heuristics, with high running times and no approximation guarantees.

Clearly, none of the above works consider the Balanced Coverage problem that allows for partial task coverage and tries to optimize coverage and maximum workload simultaneously. To the best of our knowledge, we are the first to design an algorithm with the relaxed approximation guarantees as captured in Equation (2) for this problem.

Contributions: In summary, our contributions are:

- We introduce the Balanced Coverage problem, which is a team formation problem with multiple tasks. In Balanced Coverage we seek an assignment of experts to tasks such that we maximize the coverage of tasks in terms of skills and minimize the maximum load of the experts in the assignment.
- We show that the Balanced Coverage problem is NP-hard and we design an efficient polynomial-time algorithm for solving it.
- We show how to adopt a recently proposed weaker notion of approximation and prove that our algorithm is indeed an approximation algorithm within this framework.
- We further show how to efficiently tune the importance of the two parts in the objective and thus make our objective flexible, and our algorithm versatile in different application scenarios.
- In a thorough experimental evaluation with a variety of datasets, we demonstrate the efficiency and the practical utility of our algorithm.

Roadmap: The rest of the paper is organized as follows: in the next section we give a thorough description of the related work. In Section 3 we give our notation and problem definitions. A detailed discussion of our algorithm and speedups is given in Section 4. In Section 5 we discuss how to efficiently balance and tune the importance of the two parts of the objective function, and in Section 6 we present our experiments on real-world data. We conclude the paper in Section 7.

2 RELATED WORK

In this section, we highlight some related work in team formation and discuss its relationship to the problem and the algorithmic techniques we propose in this paper.

Team formation with complete task coverage: The majority of the work in team formation assumes that there is a single task, or a collection of tasks, which require a set of skills. Additionally there are experts who possess a subset of skills. The goal is to identify a "good" subset of the experts that collectively cover *all* the skills required by the task – or each task of the collection. In the majority of the research[1–3, 7, 10–17, 23, 25], the requirement that all skills of the tasks are covered is strict – therefore, many of these team-formation problems contain an instance of the set cover problem. Different problem formulations arise from the different definitions of the "goodness" of a team (i.e., small communication cost, small maximum workload etc.).

Among the works that require that all skills of the input tasks need to be covered, the most related to ours is the work by Anagnostopoulos et al. [1]. They consider multiple tasks and multiple experts; every task has a set of required skills and every expert also has a set of skills. Their goal is to assign experts to tasks so that every task is completely covered by the skills of the experts assigned to it, while minimizing the maximum workload among the experts. They consider both the offline and the online version of this problem. The main difference between this setting and ours is that we consider a combined objective of maximizing coverage and minimizing maximum workload. Thus, the *complete* coverage of the tasks is not a strict constraint for us. Also, their problem is a minimization problem, while ours is a maximization problem. Therefore, the algorithmic techniques we develop in order to solve our problem are different from the ones they use.

Team formation with partial task coverage: More recently there has been some work, which similar to ours, focuses on partial task coverage. In this case, the goal is to cover as many of the skills of the input task while also trying to accommodate some other objective [8, 21, 22]. For example, the work of Dorn and Dustar [8] balances coverage with the team's communication cost on a graph. In our setting we have no graph and therefore their techniques are orthogonal to ours. The more recent work of Nikolakaki et al. [22] balances coverage of a *single* task with the total cost of experts, i.e., the *summation* of their weights (e.g., salaries). Our framework examines team formation for *multiple* tasks and minimizes the *maximum workload* of the experts. Although we adopt the same approximation framework adopted by Nikolakaki et al. [22], our objective is different and thus leads to different algorithmic techniques for solving our problem.

Also related is the work of Nikolakaki et al. [21]. Similar to the problem we study here, that work also considers partial task coverage and aims to optimize for coverage and maximum workload of the experts simultaneously. However, that problem is different from ours because it is is stated as a minimization problem: the goal there is to minimize the maximum workload and the fraction of the tasks that remain uncovered. In contrast, we express the Balanced Coverage problem as a maximization problem. While we find our objective function more intuitive, it is also challenging to optimize – because it can potentially take negative values. Consequently, we need to adopt the approximation framework that is more appropriate for such functions and then design approximation algorithms tailored to this maximization objective. On the other hand, Nikolakaki et al. provide a set of intuitive, yet computationally inefficient, heuristics and do not provide any formal approximation guarantees.

Approximation framework: As we have already discussed, one of the intricacies of our maximization objective function is that it can potentially take negative values. The approximation of such functions requires a weaker notion of approximation that is different from the multiplicative approximation bounds [9, 19]. Although we adopt this framework in our case, our objective function does not fall into any of the categories that have been studied before. Therefore, we need to design new algorithms for optimizing it.

3 PROBLEM DEFINITION

In this section, we first describe our notation and basic definitions and then we formally define our problem.

3.1 Preliminaries

Throughout the paper we assume a set of m tasks $\mathcal{J} = \{J_1, \ldots, J_m\}$ and a set of n experts $\mathcal{E} = \{E_1, \ldots, E_n\}$. We also assume that there is a set of skills S such that every task *requires* a set of skills and every expert *masters* a set of skills. That is, for every J_j we have that $J_j \subseteq S$ and for every expert we have that $E_i \subseteq S$.

An *assignment* of experts to tasks, is represented by a binary matrix A, such that $A(i, j) = 1$ if expert E_i is assigned to task J_j; otherwise $A(i, j) = 0$. Alternatively, one can view an assignment A as a bipartite graph with the nodes on the one side corresponding to the experts and the nodes on the other side corresponding to the tasks; edge (i, j) exists iff $A(i, j) = 1$. Finally, we often view an assignment A as a *set* of its 1-entries.

Given an assignment A, we define the *coverage* of task J_j as the fraction of the skills in J_j covered by the experts assigned to J_j. Formally,

$$C(J_j \mid A) = \frac{|(\cup_{i:A(i,j)=1} E_i) \cap J_j|}{|J_j|}.$$

Note that $0 \leq C(J_j \mid A) \leq 1$.

Given an assignment A, and the individual task coverages $C(J_j \mid A)$, we define the *overall coverage* as the sum of the individual task coverages:

$$C(A) = \sum_{j=1}^{m} C(J_j \mid A).$$

Finally, given an assignment A, we define the *load* of expert E_i in A as the number of tasks that E_i is assigned to. Formally:

$$L(E_i \mid A) = \sum_j A(i, j).$$

Given an assignment A, we can also compute the *maximum load* among all experts to be

$$L_{\max}(A) = \max_i L(E_i \mid A).$$

3.2 Problem definition

Given the above definitions, we now define the Balanced Coverage problem:

Problem 1 (Balanced Coverage). *Given a set of m tasks $\mathcal{J} = \{J_1, \ldots, J_m\}$ and a set of n experts $\mathcal{E} = \{E_1, \ldots E_n\}$ find an assignment A of experts to tasks such that*

$$F(A) = C(A) - L_{max}(A) \tag{3}$$

is maximized.

Observations: A few observations related to this problem definition are in order.

Observation 1: The objective function $F()$ is a summation of two quantities: coverage and maximum load. The coverage is a sum of normalized coverages and therefore it is a quantity that takes real values between $[0, m]$; the value of 0 is achieved when no task is covered and the value m is achieved when all tasks are fully covered. The maximum load is also a term that takes integer values between $\{0, m\}$, as the maximum load of an expert is between 0 and the total number of tasks. Therefore, the values of the two quantities are comparable and they can be added (or subtracted). However, depending on the dataset one may need a balancing coefficient, say in front of the overall coverage, that will tune the importance of the two parts of the objective. A detailed discussion on this issue is provided in Section 5.

Observation 2: The objective of the Balanced Coverage problem consists of two terms: the coverage, which we want to maximize, and the maximum load, which we want to minimize. These two terms act in opposition to one another and a good solution needs to identify a "balance point" between the experts being used and the coverage being achieved. This is the reason why the number of experts that are going to be used is not constrained in the definition of the Balanced Coverage problem itself.

Observation 3: Another observation that will prove useful is that the first part of the objective, i.e., $C(A)$, is a monotone and submodular function. We state that in the following proposition:

Proposition 3.1. *The part of the objective $F()$ that corresponds to the overall coverage: $C(A) = \sum_{j=1}^{m} C(J_j \mid A)$ is a monotone and submodular function.*

The monotonicity proof is based on the idea that for any two assignments A and A' if $A \subseteq A'$, then $C(A) \leq C(A')$.

The proof for the submodularity is based on the idea that for any two assignments A and A' such that $A \subseteq A'$ and for any (i, j) such that $A'(i, j) \neq 1$, then

$$C\left(A \cup (i, j)\right) - C\left(A\right) \geq C\left(A' \cup (i, j)\right) - C\left(A'\right).$$

The details of the proofs are omitted due to space constraints and because all arguments are standard.

Problem complexity: Clearly there are cases where our problem is easy to solve: for example, if there is only one task then the best solution is the one assigning every expert to this one task. However, our problem is NP-hard in general. Using similar observations as the ones made by Anagnostopoulos et al. [1] we can show that the Balanced Coverage problem is NP-hard: for example, for tasks that require a single skill (the same across all tasks) and experts that have this skill, we can show that the Balanced Coverage problem is NP-hard as it becomes identical to *load balancing with restricted assignment* [4, 5]. This problem is usually formulated on a bipartite graph with jobs on one side and machines on the other side, and edge (i, j) exists if job i can be processed on machine j. That is, each job can only be assigned to one of its adjacent machines in the bipartite graph. The goal is to find an assignment of jobs to machines that minimizes maximum load. In fact, using the results of Anagnostopoulos et al. [1] we can show that our problem is NP-hard even when there are only two tasks.

Theorem 3.2. *The Balanced Coverage problem is NP-hard even for $m = 2$.*

A sketch of the proof works as follows: In the case of Anagnostopoulos et al. the balance task-covering problem for two tasks corresponds to the problem where you have a set of experts (who possess certain skills) and a set of tasks (that require skills) and the goal is to assign experts to tasks such that the assignment guarantees that *all skills of every task* are covered and the objective is to minimize the maximum load among the experts. For two tasks this happens when the maximum load of every expert is equal to 1. In our case, this happens iff there is an assignment of experts to tasks such that $F(A) \geq 1$.

4 ALGORITHM

The overall objective function $F()$ of the Balanced Coverage problem is defined as the difference between a submodular function *(coverage)* and another function *(maximum load)*. Consequently, it does not have a concrete form i.e., it is not linear or convex. Therefore, existing results on optimizing a submodular function [20] or a submodular plus a linear or convex function [9, 19, 22] are not applicable.

Here, we describe ThresholdGreedy, a polynomial-time algorithm that is designed specifically for the Balanced Coverage problem. We also show that the ThresholdGreedy algorithm provides a solution assignment A that has the following property:

$$C(A) - L_{\max}(A) \geq \left(1 - \frac{1}{e}\right) C(\mathrm{OPT}) - L_{\max}(\mathrm{OPT}), \tag{4}$$

where OPT is the optimal solution to the Balanced Coverage problem.

The approximation guarantee described in (4) is a weaker form of approximation than standard multiplicative approximation guarantees. However, this is standard in cases, like ours, where the objective function is not guaranteed to be positive [9, 19, 22].

4.1 The ThresholdGreedy algorithm

A key observation is that the value of $L_{\max}$ is an integer in the range of $\{0, m\}$, where m is the total number of tasks. Therefore, ThresholdGreedy proceeds by finding an assignment for each possible value of $L_{\max}$ and then returns the assignment with the best value of $F()$. The pseudocode of the algorithm is given in Algorithm 1.

Algorithm 1 The ThresholdGreedy algorithm.

```
Input: Set of $m$ tasks $\mathcal{J}$ and $n$ experts $\mathcal{E}$
Output: An assignment of experts to tasks $A$
1: $A \leftarrow \emptyset, F_{\max} = 0$
2: for $\tau = 1, ..., m$ do
3:     Create the set of experts $\mathcal{E}_\tau$, with $\tau$ copies of each expert
4:     $A_\tau$ = Greedy($\mathcal{E}_\tau, \mathcal{J}$)
5:     Compute $F_\tau = C(A_\tau) - \tau$
6:     if $F_\tau \geq F_{max}$ then
7:         $F_{max} = F_\tau$
8:         $A \leftarrow A_\tau$
9:     end if
10: end for
11: return $A$
```

In more detail, given a threshold τ on the value of $L_{\max}$, any expert can be used at most τ times. Conceptually, this means that there are τ copies of every expert and we find A_τ to be the Greedy assignment corresponding to τ. This assignment A_τ is found by invoking the standard Greedy algorithm [24] – for optimizing a monotone submodular function – in order to optimize the total coverage function, i.e. $C()$ – see next paragraph for further explanation. After trying all possible m values of τ, we pick the assignment A_τ that has the maximum value of the objective $F(A_\tau)$.

In Line 4 of Algorithm 1, we call the Greedy algorithm for solving the coverage problem for input experts $\mathcal{E}_\tau$ and tasks $\mathcal{J}$. In this case, the algorithm greedily assigns experts in $\mathcal{E}_\tau$ to tasks until there are no more experts available. At step $\ell + 1$, the Greedy algorithm finds assignment $A_\tau^{\ell+1}$ by extending A_τ^{ℓ} with the addition of expert i assigned to task j so that its *marginal gain*

$$\tilde{C}((i,j) \mid A^\ell) = C\left(A_\tau^\ell \cup (i,j)\right) - C\left(A_\tau^\ell\right) \tag{5}$$

is maximized. In this process, each one of the τ copies of every expert is considered as a different expert and once it is assigned to a task it is removed from the candidate experts.

4.2 Approximation properties

We now prove our approximation result for ThresholdGreedy, as captured already in Equation (4). However, before proving the main theorem we need the following lemma, which we state here:

Lemma 4.1. *Let A_τ be the assignment of experts to tasks returned by* Greedy *(Line 4 of Algorithm 1) for fixed threshold workload τ. Let OPT_τ be the optimal assignment of experts $\mathcal{E}_\tau$ to tasks $\mathcal{J}$ with respect to the coverage objective $C(OPT_\tau)$. Then, it holds that:*

$$C\left(A_\tau\right) \geq \left(1 - \frac{1}{e}\right) C\left(OPT_\tau\right).$$

The proof of this lemma follows the standard proof of the Greedy algorithm being an $\left(1-\frac{1}{e}\right)$-approximation algorithm to the coverage problem [24] and is thus omitted.

This lemma says that for every threshold τ (i.e., for every iteration of ThresholdGreedy), the Greedy subroutine is guaranteed to return a solution that has a good coverage, with respect to the optimal solution for the coverage problem for this threshold τ. The lemma does not mention anything about the final solution picked by ThresholdGreedy, neither does it say anything about the approximation that this solution has with respect to the overall objective function $F()$. We build upon the lemma and state the following theorem.

Theorem 4.2. *Let A be the assignment returned by* ThresholdGreedy *and let OPT be the optimal assignment for the Balanced Coverage problem. Then we have the following approximation:*

$$C(A) - L_{max}(A) \geq \left(1-\frac{1}{e}\right) C(OPT) - L_{max}(OPT).$$

Proof. Let's assume that $L_{\max}(\mathrm{OPT}) = \tau$; note that $L_{\max}(A)$ may or may not be equal to τ. Then, we have the following:

$$\begin{aligned} F(A) &\geq F(A_\tau) \quad \text{(True for any } \tau) \\ &= C(A_\tau) - \tau \\ &\geq \left(1-\frac{1}{e}\right) C(\mathrm{OPT}_\tau) - \tau \quad \text{(Lemma 4.1)} \\ &\geq \left(1-\frac{1}{e}\right) C(\mathrm{OPT}) - \tau \quad (\mathrm{OPT}_\tau \text{ is optimal for threshold } \tau) \\ &= \left(1-\frac{1}{e}\right) C(\mathrm{OPT}) - L_{\max}(\mathrm{OPT}). \end{aligned}$$

□

4.3 Running time and speedups

In this section we examine the running time and speedup techniques for ThresholdGreedy.

First observe that a naive implementation of ThresholdGreedy takes time $O(m^2n^2)$; it requires m calls to the Greedy routine in Line 4, which also if implemented naively takes time $O(mn^2)$. Such a running time would make ThresholdGreedy impractical. Below, we discuss a two methods that help us significantly improve the running time of our algorithm and allow us to experiment with reasonably large datasets.

Lazy greedy instead of greedy: First, instead of using the naive implementation of Greedy, we deploy the lazy-evaluation technique introduced by Minoux [18] for the standard Greedy algorithm. The computational bottleneck in Greedy is that at every iteration ℓ, the algorithm needs to compute for every expert-task pair (i,j) it's marginal gain $\tilde{C}((i,j) \mid A^\ell)$ (see Equation (5)). To speed these computations up by avoiding unnecessary evaluations, we store each pair (i,j) in a maximum priority queue with key $v(i,j)$. We initialize the keys to $v(i,j) = \tilde{C}((i,j) \mid A^0)$, where A^0 is the assignment with all entries equal to 0. As the algorithm runs, the keys in the queue could store potentially outdated marginal gains, but the algorithm only updates them in a lazy fashion. Since $C()$ is a submodular function, marginal gains can only decrease as the number of 1-entries in the formed assignment increases. Therefore, the keys are always an upper bound on the corresponding marginal gains. In each iteration, the Greedy algorithm finds the element in the queue that has the maximum marginal gain as follows: first it removes from the queue the pair (i,j) with the maximum key value and computes it's updated marginal gain: $\tilde{C}((i,j) \mid A^\ell)$. Then, it compares $\tilde{C}((i,j) \mid A^\ell)$ to the key $v(i',j')$ of pair (i',j') that became the new top of the queue (after removing (i,j)). If $\tilde{C}((i,j) \mid A^\ell) > v(i',j')$, then (i,j) is the element with the largest marginal gain. Otherwise, we reinsert (i,j) in the queue with its updated key value being $v(i,j) = \tilde{C}((i,j) \mid A^\ell)$, and repeat this process with the new maximum key value.

Throughout, we use this lazy-evaluation version of Greedy, which improves the overall running time of ThresholdGreedy significantly.

Early termination of ThresholdGreedy: Another computational bottleneck for ThresholdGreedy is its outer loop (see line 2 in Algorithm 1), which needs to be repeated m times, where m is the total number of tasks. Here we show that not all m values of τ need to be considered. We show that the value of the objective function as computed by ThresholdGreedy for the different values of τ is a unimodal function, which initially increases and then starts decreasing. Therefore, once a maximum is found for some value of τ, then the algorithm can safely terminate as the value of the objective will not improve for larger values of τ.

To prove this, we rely on the properties of ThresholdGreedy as well as on the fact that the coverage function, $C()$, is monotone and submodular (See Proposition 3.1). More specifically, let A_τ be the assignment produced at the τ-th iteration of ThresholdGreedy. Then, we denote $F_\tau = F(A_\tau)$ and $C_\tau = C(A_\tau)$; by definition: $F_\tau = C_\tau - \tau$. Moreover, the monotonicity and submodularity of the coverage function imply the following observations[1]:

Observation 1. *The monotonicity of the overall coverage function implies that for every $\tau \in \{1,\ldots,m\}$: $C_\tau \geq C_{\tau-1}$.*

Observation 2. *The submodularity of the overall coverage function implies that for every $\tau \in \{1,\ldots,m-1\}$: $C_\tau - C_{\tau-1} \geq C_{\tau+1} - C_\tau$.*

These observations rely on the fact that in every iteration τ, ThresholdGreedy produces assignment A_τ, which has the property that $A_\tau \subseteq A_{\tau+1}$. That is, the 1-entries in A_τ are a superset of the 1-entries in $A_{\tau+1}$.

Theorem 4.3. *If there is a value of the threshold τ^*, such that $F_{\tau^*} \geq F_{\tau^*-1}$ and $F_{\tau^*} \geq F_{\tau^*+1}$, then the values of the objective function $F_\tau = F(A_\tau)$ as computed by* ThresholdGreedy *(line 5) for $\tau = 1,\ldots,m$ are unimodal. That is, $F_1 \leq F_2 \leq \ldots \leq F_{\tau^*}$ and $F_{\tau^*} \geq F_{\tau^*+1} \geq \ldots \geq F_m$.*

Proof. Let's assume that there is a threshold τ^* such that $F_{\tau^*} \geq F_{\tau^*-1}$ and $F_{\tau^*} \geq F_{\tau^*+1}$. Since $F_{\tau^*} \geq F_{\tau^*-1}$, we have

$$\begin{aligned} C_{\tau^*} - \tau^* &\geq C_{\tau^*-1} - (\tau^* - 1) \\ (C_{\tau^*} - C_{\tau^*-1}) &\geq 1. \end{aligned} \tag{6}$$

Using (6) and Observation 2, we have that

$$C_1 - C_0 \geq C_2 - C_1 \geq \ldots \geq C_{\tau^*} - C_{\tau^*-1} \geq 1.$$

[1] $C_0 = 0$ since it is the coverage of the empty assignment.

Thus, for every $\tau \leq \tau^*$ it holds that

$$\begin{aligned} C_\tau - C_{\tau-1} &\geq 1 \\ C_\tau - \tau &\geq C_{\tau-1} - (\tau - 1) \\ F_\tau &\geq F_{\tau-1}. \end{aligned}$$

The proof is symmetric for the values of $\tau > \tau^*$. That is, since $F_{\tau^*} \geq F_{\tau^*+1}$, we have

$$\begin{aligned} C_{\tau^*} - \tau^* &\geq C_{\tau^*+1} - (\tau^* + 1) \\ (C_{\tau^*+1} - C_{\tau^*}) &\leq 1. \end{aligned} \tag{7}$$

Using (7) and Observation 2, we have that

$$C_m - C_{m-1} \leq C_{m-1} - C_{m-2} \leq \ldots \leq C_{\tau^*+1} - C_{\tau^*} \leq 1.$$

Thus, for every $\tau > \tau^*$ it holds that

$$\begin{aligned} C_{\tau+1} - C_\tau &\leq 1 \\ C_{\tau+1} - (\tau - 1) &\leq C_\tau - \tau \\ F_{\tau+1} &\leq F_\tau. \end{aligned}$$

□

Throughout the rest of the paper we will call the value of τ for which $F()$ gets maximized in the iterations of the ThresholdGreedy algorithm the *best-greedy workload* and the corresponding value of the objective the *best-greedy objective.*

5 TUNING THE IMPORTANCE OF COVERAGE AND WORKLOAD

Up to this point, we have assumed that the objective function is of the form $F(A) = C(A) - L_{\max}(A)$. As we discussed in Section 3 these two terms are measured in terms of tasks (i.e., tasks being covered and tasks being assigned) and therefore it makes sense to combine them in a single objective. However, as we hinted previously, sometimes one might need to weight one of the two terms (the coverage or the maximum load) more heavily than the other. We can enhance our framework to do so by adding a *balancing coefficient* $\lambda > 0$, to the coverage function so that the new enhanced objective becomes:

$$F^\lambda(A) = \lambda C(A) - L_{\max}(A). \tag{8}$$

Values of $\lambda > 1$ assign a higher weight to the coverage; the larger the value of λ the larger the importance of the coverage. Conversely, small values of $\lambda \in (0, 1)$ assign more importance to the maximum load.

Finding the assignment that optimizes the enhanced objective as given in Equation (8) can still be done using ThresholdGreedy; for a given λ the only modification one has to make to the algorithm is to change line 5 to take the value of λ into account.

However, when trying to optimize for $F^\lambda()$ one must choose an appropriate value of λ such that it balances the relative importance of task coverage and expert workload as desired. This value typically depends on the application domain and the particular characteristics of the dataset. Finding a suitable value would require a search over the possible values of λ to identify a value that gives reasonable results.

In Section 6.3 we implement a methodology for choosing the right value of λ by examining different values of λ and then picking the one that gives the most intuitive trade-off between the coverage and the load of the corresponding solutions. Such methodology requires a search over different values of λ. There are two "naive" ways of implementing such a search process: The first is to run ThresholdGreedy (with all the speedups we proposed in Section 4.3) for the different values of λ. The second is to run ThresholdGreedy *without* the early termination technique we discussed in Section 4.3 and for $\lambda = 1$. This would mean that we would have to go over all possible values of τ, and for each threshold τ store independently the value of the coverage C_τ for this threshold. Then make a pass over all these values and weigh them appropriately with different λs. The first solution, requires us to run ThresholdGreedy as many times as the different λs we want to explore. The second solution requires us to run ThresholdGreedy once, but for *all* possible values of threshold $\tau = m$. Both these solutions are infeasible in practice even for datasets of moderate size.

An efficient search on the values of λ: We show that if we want to explore the solutions of ThresholdGreedy over different values of $\lambda \in \Lambda$, where $\Lambda \subseteq \mathbb{R}_+$, then we can do this efficiently, by running ThresholdGreedy only once and – at the same time – exploiting the early termination trick we discussed in Section 4.3.

This speedup is achieved by the following observation:

Proposition 5.1. *Assume that $\lambda_1 > \lambda_2$ and let the best-greedy objectives achieved for those values be $F^{\lambda_1}_{\tau_1}$ and $F^{\lambda_2}_{\tau_2}$ respectively. Then, for the corresponding best-greedy workloads we have that $\tau_1 \geq \tau_2$.*

Proof. Since $\lambda_1 > \lambda_2$, there exists an $\alpha > 1$ such that $\lambda_1 = \alpha\lambda_2$. Our proof will be by contradiction: suppose that $\tau_1 < \tau_2$. By Observation 1 we have that $C_{\tau_2} \geq C_{\tau_1}$. Since τ_2 corresponds to the best-greedy workload for F^{λ_2} we have that $F^{\lambda_2}_{\tau_2} \geq F^{\lambda_2}_{\tau_1}$ and thus:

$$\begin{aligned} \lambda_2 C_{\tau_2} - \tau_2 &\geq \lambda_2 C_{\tau_1} - \tau_1 \\ \lambda_2(C_{\tau_2} - C_{\tau_1}) &\geq (\tau_2 - \tau_1) \end{aligned}$$

Since τ_1 corresponds to the best-greedy workload for F^{λ_1} we have that $F^{\lambda_1}_{\tau_2} \leq F^{\lambda_1}_{\tau_1}$

$$\begin{aligned} \lambda_1 C_{\tau_2} - \tau_2 &\leq \lambda_1 C_{\tau_1} - \tau_1 \\ \lambda_1(C_{\tau_2} - C_{\tau_1}) &\leq (\tau_2 - \tau_1) \\ \alpha\lambda_2(C_{\tau_2} - C_{\tau_1}) &\leq (\tau_2 - \tau_1) \end{aligned}$$

Combining these two results we get

$$\alpha\lambda_2(C_{\tau_2} - C_{\tau_1}) \leq (\tau_2 - \tau_1) \leq \lambda_2(C_{\tau_2} - C_{\tau_1}),$$

which implies that $\alpha \leq 1$, which is a contradiction. □

6 EXPERIMENTS

In this section, we examine the performance of the ThresholdGreedy algorithm on real-world datasets. We compare its performance with three baseline algorithms in terms of the objective function $F()$, the maximum workload, $L_{\max}$, and the running time. We observe that ThresholdGreedy consistently performs the best in terms of the objective across all datasets. Additionally, it finds an assignment with a low maximum workload and it runs in a reasonable amount of time, even for datasets with several thousand experts and tasks.

Our code is implemented in Python and for all our experiments we used single process implementations on a 64-bit MacBook Pro with an Apple M1 Pro CPU and 16 GB RAM. We will make all our code, datasets and chosen hyperparameters available online for replication purposes.

Before presenting our results, we describe the datasets and the baselines we used for our experiments.

6.1 Datasets

In order to showcase the efficacy and the efficiency of our solution we use several real-world datasets, which have also been used in the past in the papers that are the most related to ours [1, 21, 22]. A summary of the characteristics of these datasets is shown in Table 1.

The IMDB data: The data is obtained from the International Movie Database [2]. The original dataset contains detailed information about movies, TV shows and documentaries along with the associated actors, directors, movie year and movie genre(s). We wish to simulate a team formation setting where movie directors conduct auditions for movie actors. For the purposes of our experiments, movie genres correspond to skills, movie directors ti experts and actors to tasks. The set of skills possessed by a director or an actor are the union of movie genres of the movies they have participated in. In order to experiment with datasets of different sizes, we created three instances from the original data by selecting all movies created since 2015, 2018 and 2020 and extract their directors and actors. We refer to these datasets as *IMDB*-2015, *IMDB*-2018 and *IMDB*-2020 respectively.

The Bibsonomy data: Our second dataset comes from bibsonomy, a social bookmark and publication sharing system [6]. The original dataset consists of a large number of publications, each of which is written by a set of one or more authors. Each publication is associated by a set of *tags*. We filter tags for stopwords and use the 1000 most common tags as skills in for our experiments. We use this data to simulate a setting where certain expert authors conduct interviews for other less prolific authors. Thus the prolific authors become the experts and the rest are the tasks. An author's skills are the union of the tags associated with their publications. Upon inspection of the distribution of skills among all authors we determine prolific authors (experts) to be those authors with at least 15 skills. We created three datasets for our experiments by selecting all publications since 2010, 2015 and 2020 and extracting their authors. We refer to these datasets as *Bibsonomy*-2010, *Bibsonomy*-2015, *Bibsonomy*-2020 respectively.

The Freelancer data: This dataset consists of a random sample from a large set of real tasks that are posted by users in the *Freelancer* online labor marketplace freelancer.com. The data consists of tasks that require skills and experts that have skills. We refer to this dataset as *Freelancer*.

The Guru data: Similar to *Freelancer*, this dataset also consists of a random sample from a large set of real projects that are posted by users in the *Guru* online labor marketplace guru.com. Again projects require skills and experts have skills and therefore the data fit our purposes. We refer to this dataset as *Guru*.

6.2 Baselines

In our experiments, we used the following three intuitive baselines.

[2]https://www.imdb.com/interfaces/

Table 1: Summary statistics of our datasets.

Dataset	Experts	Tasks	Skills	skills/ expert	skills/ task
IMDB-2015	5551	18109	26	2.4	3.1
IMDB-2018	3871	13183	26	2.1	2.7
IMDB-2020	2176	7858	25	1.9	2.4
Bibsonomy-2010	3044	21981	1000	13.7	4.8
Bibsonomy-2015	1904	9061	1000	10.9	4.3
Bibsonomy-2020	177	834	858	11.5	3.6
Freelancer	1212	993	175	1.5	2.9
Guru	6120	3195	1639	13.1	5.2

TaskGreedy: This algorithm is inspired by the previous work of Nikolakaki et al. [21]. TaskGreedy goes over all tasks sequentially and for each task it greedily assigns experts to maximize the task's coverage. To balance the maximum workload with the total task coverage successfully, we implement two heuristics. First we randomize the order in which experts are greedily assigned to tasks in each iteration. This ensures an even distribution of experts in a setting in which several experts might be equivalently good for a task. Second, we only assign experts if they yield a significant increase in the task coverage. We quantify this coverage amount by a hyperparameter, β, which we specifically grid search and optimize for each dataset. Excluding the grid search, the TaskGreedy algorithm has a running time of $O(mn)$ since there are n experts available for each of the m tasks.

NoUpdateGreedy: This algorithm corresponds to a simple modification of the ThresholdGreedy algorithm: we use a maximum priority queue but do not update the queue in each iteration of the algorithm. For each expert-task pair (i, j), we initialize the keys in the priority queue to $v(i, j) = \tilde{C}((i, j) \mid A^0)$, where A^0 is the assignment with all entries equal to 0. We then use these initial marginal gain values to iteratively add expert-task edges (i, j) from the top of the priority queue to our solution. In order to improve the performance of NoUpdateGreedy, we only use an expert if $v(i, j) > \beta$, where β is a hyperparameter. NoUpdateGreedy has a running time of $O(mn \log(mn))$, since there are mn total expert-task edges, and sorting these edges for the priority queue takes $O(\log(mn))$ time.

SmartRandom: This baseline algorithm assigns each expert to several tasks in a semi-random manner. More specifically, it works as follows: we add an expert-task assignment to the solution if the task is not yet fully covered. For each expert, we limit the number of tasks that the expert can be assigned to using a hyper-parameter μ. Finally, we only assign an expert to a task if they yield a significant increase in the task coverage. Similar to other baseline algorithms, we quantify this coverage amount by a hyperparameter, β. We specifically grid search and optimize the hyperparameters μ and β for each dataset. Excluding the grid search, The SmartRandom algorithm has a running time of $O(mn)$.

For all our experiments, we do grid search over the values of all hyperparameters and we only report the results for the best value we obtained for each algorithm and each dataset.

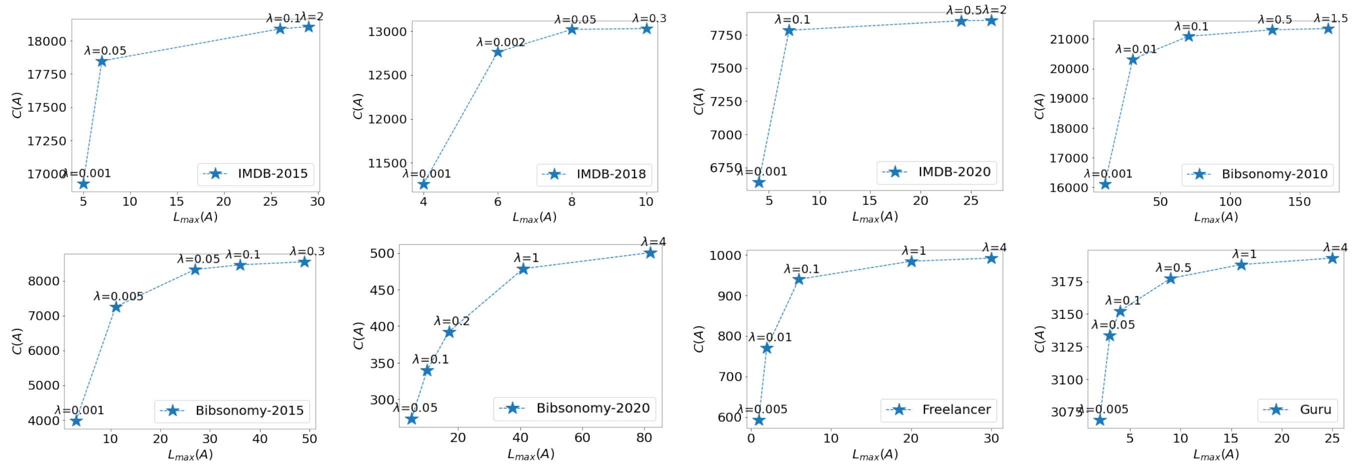

Figure 1: The best-greedy workload $L_{\max}(A)$ value and the coverage $C(A)$ corresponding to the best-greedy objective $F^{\lambda}(A)$ computed by `ThresholdGreedy`. Each subplot shows a range of values of the balancing coefficient λ for each dataset.

6.3 Tuning coverage and workload importance

Before showing our experimental results, we discuss how we set the balancing coefficient λ. For this, we use the techniques we described in Section 5. That is, we first run `ThresholdGreedy` with a large value of λ, and determine the *best-greedy workload* and the corresponding value of the *best-greedy objective*. We then compute the corresponding *best-greedy* values for smaller values of λ, and plot the corresponding values of $C(A)$ and $L_{\max}(A)$ for each λ value. Figure 1 shows these scatterplots for each dataset. In most of our datasets we experimented with relatively small values of $\lambda \in (0, 5]$.

From these plots, we employ the *elbow method* to identify a suitable value of λ such that the best-greedy workload and best-greedy objective values yield a high value for the overall coverage, $C(A)$ while simultaneously giving a reasonably low value for the $L_{\max}(A)$. Graphically, the best λ value for each dataset corresponds to the λ value observed at the "elbow" of the plot, where further increase of λ does not give significant increase of coverage. The values of λ we picked for the different datasets are shown besides the dataset name in Table 2.

6.4 Evaluation

We compare the performance of all four algorithms in the experiments conducted on real-world data. Intuitively, a *good* solution to an instance of Balanced Coverage problem is an assignment A that not only maximizes the overall task coverage but also minimizes the maximum load of the assignment. It is also important that the solution be computed in a reasonable amount of time, even for large numbers of experts and tasks.

A summary of the results of our experiments is laid out in Table 2. We observe that the `ThresholdGreedy` algorithm outperforms the other algorithms both in terms of the objective (consistently higher) and in terms of the workload (predominantly lower). Additionally, we observe a reasonable running time for the algorithm, even on the *Bibsonomy*-2010 and *IMDB*-2015 datasets, which have several thousand tasks and experts.

Comparison of the objective values (F): `ThresholdGreedy` consistently finds the assignment with the best objective value across all our experiments. On average across all datasets `ThresholdGreedy` performs about 55% better than `TaskGreedy` and `NoUpdateGreedy`, and more than 100% better than `SmartRandom`. Additionally, we observe that as the datasets get larger, the superior performance of `ThresholdGreedy` becomes more evident.

`TaskGreedy` and `NoUpdateGreedy` perform relatively well on the *IMDB*-2020 and *Bibsonomy*-2020 datasets: they return objective values that are within 20% of the objective value of `ThresholdGreedy`. This is because the smaller datasets lend themselves better to heuristic choices made by these baselines: the pool of suitable experts available to `TaskGreedy` is small and the initial marginal gain values used by `NoUpdateGreedy` are good estimators of the true marginal gain values in subsequent iterations. On the other hand, even for these datasets the baseline algorithms have significantly higher workloads of over 75 for *Bibsonomy*-2020 and over 100 for *IMDB*-2020. These workloads are significantly larger than the 41 and 7 workloads achieved by `ThresholdGreedy`.

`SmartRandom` and `NoUpdateGreedy` perform poorly on the *Freelancer* and *Guru* datasets: they return objective values that are less than 50% of the objective value returned by `ThresholdGreedy`. Since these datasets have more experts than tasks, the lower objective values could be attributed to the smaller number of suitable task options available to each expert in the execution of `SmartRandom`, and the lack of an accurate marginal gain value for expert-task edges in the execution of `NoUpdateGreedy`. We see that `TaskGreedy` and `NoUpdateGreedy` achieve comparable performance across the *IMDB* and *Bibsonomy* datasets. However, `TaskGreedy` performs about 50% better on the *Bibsonomy*-2010 dataset, and `NoUpdateGreedy` performs about 30% better on the *IMDB*-2015 dataset.

Comparison of maximum load values ($L_{\max}$): We observe that the `ThresholdGreedy` algorithm consistently finds the assignment with the lowest maximum workload value across all our experiments. Additionally, the baselines return maximum load values that are significantly larger than those returned by `ThresholdGreedy`.

Table 2: Experimental performance of the ThresholdGreedy and baseline algorithms on real-world datasets in terms of the objective F^λ and the maximum load $L_{\max}$.

Dataset	ThresholdGreedy		TaskGreedy		NoUpdateGreedy		SmartRandom	
	$F^\lambda(A)$	$L_{\max}(A)$	$F^\lambda(A)$	$L_{\max}(A)$	$F^\lambda(A)$	$L_{\max}(A)$	$F^\lambda(A)$	$L_{\max}(A)$
IMDB-2015 ($\lambda = 0.05$)	885	7	475	362	720	150	456	81
IMDB-2018 ($\lambda = 0.05$)	643	8	339	264	448	200	186	345
IMDB-2020 ($\lambda = 0.1$)	771	7	644	118	650	100	445	160
Bibsonomy-2010 ($\lambda = 0.1$)	2039	70	1319	243	1097	200	711	272
Bibsonomy-2015 ($\lambda = 0.05$)	389	27	96	126	129	250	92	79
Bibsonomy-2020 ($\lambda = 1$)	438	41	402	93	408	94	359	78
Freelancer ($\lambda = 0.1$)	88	6	63	36	25	50	17	33
Guru ($\lambda = 0.1$)	311	4	225	30	152	119	82	60

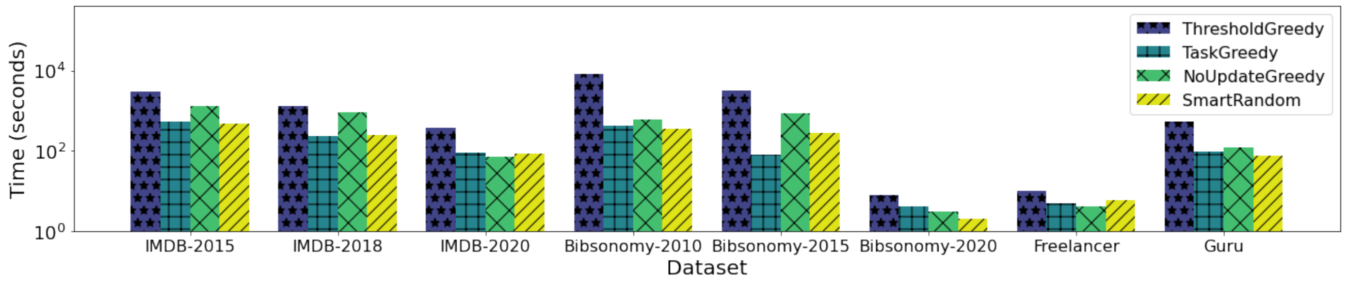

Figure 2: Running times (in seconds) of ThresholdGreedy and baselines, in logarithmic scale.

On average across all datasets ThresholdGreedy finds a maximum load value that is 85% smaller than the maximum workload values returned by the baselines. This is because, in an attempt to maximize the overall task coverage, the baseline algorithms end up assigning certain expert-task edges that are very costly. While we do see some examples of reasonable workload values (for example TaskGreedy returns $L_{\max} = 30$ for the *Guru* dataset), we see that in most cases the work load values returned by the baselines would be infeasible in practice.

Running times: While ThresholdGreedy has a theoretical running time of $O(m^2n^2)$, the speed-up techniques discussed in Section 4.3 and Section 5 lead to significantly lower running times in practice. Figure 2 shows a bar plot with the running times of all algorithms for each dataset in logscale. For the smaller datasets (e.g. *Freelancer* and *Bibsonomy*-2020), we observe that the running time of ThresholdGreedy is in the order of a few minutes, while the baselines run within a few seconds. Even for the largest datasets (e.g., *Bibsonomy*-2010 and *IMDB*-2015) the running time of our algorithm is within a few hours, while the baselines run in a few minutes. Note that the running times of the baselines as we report them here do not include the grid search we performed in order to tune their hyperparameters. While the running times of baselines are significantly smaller, recall that the solutions they return have much higher workloads and significantly lower coverages. Additionally, these baselines lack any theoretically proven approximation guarantees as the ones we have for ThresholdGreedy.

7 CONCLUSIONS

In this paper, we introduced the Balanced Coverage problem, a new problem in team formation, where the goal is to assign experts to tasks such that the total coverage of the tasks (in terms of their skills) is maximized and the maximum workload of any expert in the assignment is minimized. To the best of our knowledge we are the first to introduce and study this problem.

We showed that Balanced Coverage is an NP-hard problem and then we designed a polynomial-time approximation algorithm for solving it. One of the main technical challenges when solving the Balanced Coverage problem is that its objective function can potentially take negative values and therefore it is not amenable to approximation algorithms with the standard multiplicative guarantees. As a result, we adopted a weaker notion of approximation, which is tailored for such objective and has been introduced only recently [9, 19]. Within this framework, we showed that we can design a polynomial-time approximation algorithm for our problem. We also showed that despite the fact that a naive implementation of this algorithm is computationally expensive we can exploit the structure of our objective function and design speedups that work extremely well in practice. We also developed a more general framework where we can efficiently tune the importance of the two parts of our objective and therefore make our framework applicable to a wide set of applications. Our experimental results with a variety of datasets from various domains demonstrated the utility of our framework and the efficiency and efficacy of our algorithm.

REFERENCES

[1] Aris Anagnostopoulos, Luca Becchetti, Carlos Castillo, Aristides Gionis, and Stefano Leonardi. 2010. Power in unity: forming teams in large-scale community systems. In *Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010*. ACM, 599–608.

[2] Aris Anagnostopoulos, Luca Becchetti, Carlos Castillo, Aristides Gionis, and Stefano Leonardi. 2012. Online team formation in social networks. In *WWW*.

[3] Aris Anagnostopoulos, Carlos Castillo, Adriano Fazzone, Stefano Leonardi, and Evimaria Terzi. 2018. Algorithms for Hiring and Outsourcing in the Online Labor Market. In *ACM SIGKDD*, Yike Guo and Faisal Farooq (Eds.). ACM, 1109–1118.

[4] Yossi Azar, Andrei Z. Broder, and Anna R. Karlin. 1994. On-Line Load Balancing. *Theor. Comput. Sci.* 130, 1 (1994), 73–84.

[5] Yossi Azar, Joseph Naor, and Raphael Rom. 1995. The Competitiveness of On-Line Assignments. *J. Algorithms* 18, 2 (1995), 221–237.

[6] Dominik Benz, Andreas Hotho, Robert Jäschke, Beate Krause, Folke Mitzlaff, Christoph Schmitz, and Gerd Stumme. 2010. The Social Bookmark and Publication Management System BibSonomy. *The VLDB Journal* 19, 6 (Dec. 2010), 849–875. https://doi.org/10.1007/s00778-010-0208-4

[7] Avradeep Bhowmik, Vivek Borkar, Dinesh Garg, and Madhavan Pallan. 2014. Submodularity in team formation problem. In *SDM*.

[8] C. Dorn and S. Dustdar. 2010. Composing near-optimal expert teams: a trade-off between skills and connectivity. In *CoopIS*.

[9] Chris Harshaw, Moran Feldman, Justin Ward, and Amin Karbasi. 2019. Submodular Maximization beyond Non-negativity: Guarantees, Fast Algorithms, and Applications. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 2634–2643.

[10] Mehdi Kargar and Aijun An. 2011. Discovering top-k teams of experts with/without a leader in social networks. In *CIKM*.

[11] Mehdi Kargar, Aijun An, and Morteza Zihayat. 2012. Efficient bi-objective team formation in social networks. In *ECML PKDD*.

[12] Mehdi Kargar, Morteza Zihayat, and Aijun An. 2013. Finding affordable and collaborative teams from a network of experts. In *SDM*.

[13] T. Lappas, K. Liu, and E. Terzi. 2009. Finding a team of experts in social networks. In *KDD*.

[14] Cheng-Te Li, Man-Kwan Shan, and Shou-De Lin. 2015. On team formation with expertise query in collaborative social networks. *KAIS* (2015).

[15] L. Li, H. Tong, N. Cao, K. Ehrlich, Y.-R. Lin, and N. Bucher. 2017. Enhancing team composition in professional networks: Problem definitions and fast solutions. *TKDE* (2017).

[16] Liangyue Li, Hanghang Tong, Nan Cao, Kate Ehrlich, Yu-Ru Lin, and Norbou Buchler. 2015. Replacing the irreplaceable: Fast algorithms for team member recommendation. In *WWW*.

[17] Anirban Majumder, Samik Datta, and KVM Naidu. 2012. Capacitated team formation problem on social networks. In *KDD*.

[18] Michel Minoux. 1978. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization techniques*. Springer, 234–243.

[19] Siddharth Mitra, Moran Feldman, and Amin Karbasi. 2021. Submodular + Concave. *CoRR* abs/2106.04769 (2021).

[20] G.L Nemhauser and L. A. Wolsey. 1978. Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Research* 3, 3 (1978), 177–188.

[21] Sofia Maria Nikolakaki, Mingxiang Cai, and Evimaria Terzi. 2020. Finding teams that balance expert load and task coverage. *CoRR* abs/2011.04428 (2020).

[22] Sofia Maria Nikolakaki, Alina Ene, and Evimaria Terzi. 2021. An Efficient Framework for Balancing Submodularity and Cost. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, Feida Zhu, Beng Chin Ooi, and Chunyan Miao (Eds.). ACM, 1256–1266.

[23] Syama Sundar Rangapuram, Thomas Bühler, and Matthias Hein. 2013. Towards realistic team formation in social networks based on densest subgraphs. In *WWW*.

[24] Vijay V Vazirani. 2013. *Approximation algorithms*. Springer Science & Business Media.

[25] Xiaoyan Yin, Chao Qu, Qianqian Wang, Fan Wu, Baoying Liu, Feng Chen, Xiaojiang Chen, and Dingyi Fang. 2018. Social Connection Aware Team Formation for Participatory Tasks. *IEEE Access* (2018).