Forming Coordinated Teams that Balance Task Coverage and Expert Workload

Karan Vombatkere[1*], Aristides Gionis[2] and Evimaria Terzi[1]

[1]Department of Computer Science, Boston University, Boston, USA.
[2]Division of Theoretical Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden.

*Corresponding author(s). E-mail(s): kvombat@bu.edu;
Contributing authors: argioni@kth.se; evimaria@bu.edu;

Abstract

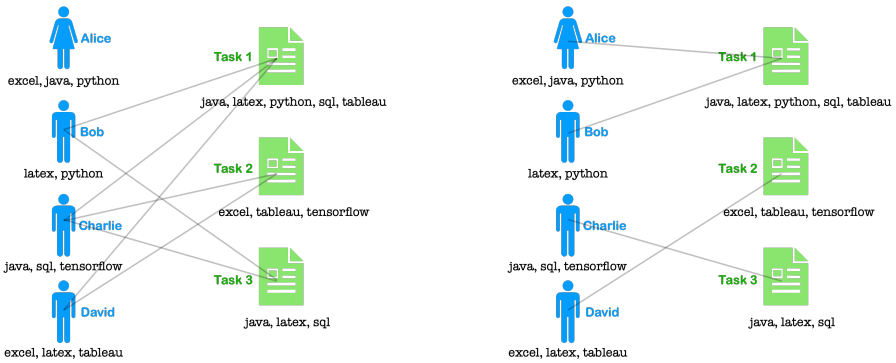
We study a new formulation of the team-formation problem, where the goal is to form teams to work on a given set of tasks requiring different skills. Deviating from the classic problem setting where one is asking to cover all skills of each given task, we aim to cover as many skills as possible while also trying to minimize the maximum workload among the experts. We do this by combining penalization terms for the coverage and load constraints into one objective. We call the corresponding assignment problem Balanced-Coverage, and show that it is **NP**-hard. We also consider a variant of this problem, where the experts are organized into a graph, which encodes how well they work together. Utilizing such a coordination graph, we aim to find teams to assign to tasks such that each team's radius does not exceed a given threshold. We refer to this problem as Network-Balanced-Coverage. We develop a generic template algorithm for approximating both problems in polynomial time, and we show that our template algorithm for Balanced-Coverage has provable guarantees. We describe a set of computational speedups that we can apply to our algorithms and make them scale for reasonably large datasets. From the practical point of view, we demonstrate how to efficiently tune the two parts of the objective and tailor their importance to a particular application. Our experiments with a variety of real-world datasets demonstrate the utility of our problem formulation as well as the efficiency of our algorithms in practice.

Keywords: team formation, submodular optimization, greedy, social network, data mining algorithms

1 Introduction

The abundance of online and offline labor markets (e.g., Guru, Freelancer, online scientific collaborations, etc.) has motivated a lot of work on the *team-formation* problem. In the team-formation setting, the input consists of (i) a task, or a collection of tasks, so that each task requires a set of skills, and (ii) a set of experts, where each expert is also associated with a set of skills. The objective is to identify one team, or one team for every task, such that all the skills in every task are covered by at least one team member. Notably, the majority of works in team-formation research require *complete* coverage of the skills of the input tasks (Anagnostopoulos et al., 2010, 2012, 2018; Bhowmik et al., 2014; Kargar et al., 2013; Kargar and An, 2011; Kargar et al., 2012; Lappas et al., 2009; Majumder et al., 2012; Li et al., 2015a,b, 2017; Rangapuram et al., 2013; Yin et al., 2018). The differences among existing papers lie in the way they define the "goodness" of a team. For example, in some cases they optimize the communication cost of the team, while in other cases they optimize the load of the experts, or their associated cost.

We motivate the inherent trade off between task coverage and expert workload using the example of Fig. 1. The assignment on the left, achieves 100% coverage for all three tasks; however Charlie has a workload of 3, Bob and David each have workload of 2 while Alice is not assigned to any task. However, the assignment on the right – which allows for partial coverage – does not cover any task 100%, yet it is more balanced in terms of expert workload; all experts now have a workload of 1.



(a) Assignment with full coverage for all tasks and unbalanced expert workload.

(b) Assignment with partial coverage and balanced expert workload.

Fig. 1: Motivating example with 4 experts and 3 tasks.

In this paper, we propose team-formation problems where the goal is to assign experts to a set of input tasks such that the task coverage is maximized, and at the same time, the maximum workload among the experts used is minimized. This trade-off suggests that we need not always cover the skills of every task completely, since covering a large fraction of their required skills might be sufficient. Also, given that overworked experts do not perform well, we penalize expert overloading by minimizing

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:

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

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 assigned to a single expert.

Although we normalize the two terms of the objective (Eq. (1)) and make them comparable, in certain applications we may want to aim for different trade off between the coverage and maximum-load terms. Thus, we incorporate the *balancing coefficient* λ, which enables an effective tuning of the importance of the two terms. We call this problem Balanced-Coverage.

Often, the experts are organized in a network, which encodes how well experts can work with each other. In the presence of such information, we extend the Balanced-Coverage problem so that the teams assigned to tasks have the property that their radius is not larger than a pre-specified threshold. The motivation is for teams to have small coordination cost and be able to work well with each other. We call this version of the problem Network-Balanced-Coverage.

We show that the two problems we define, Balanced-Coverage and Network-Balanced-Coverage, are **NP**-hard.

From the application point of view, it makes sense to relax the hard constraint of full coverage; in practice, skills in tasks are often overlapping. For example, consider a task requiring skills: *advertising, internet advertising, Facebook advertising, online marketing, social network platforms*. Clearly, these are overlapping and not all of them need to be covered. Additionally, minimizing the maximum expert workload is desirable for better team performance.

From the algorithmic point of view, optimizing the above objective, with or without the radius constraint in the teams, is challenging; the function itself may take negative values. Therefore, it does not admit multiplicative approximation guarantees. Although the coverage part of the objective ($C(\cdot)$) is a monotone submodular function, the maximum load part does not have a predictable form (i.e., it is not linear or convex). Therefore, recent techniques (Harshaw et al., 2019; Mitra et al., 2021) on submodularity optimization cannot be applied. However, we adopt from these works a weaker notion of approximation and aim to find an assignment A such that:

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

where OPT is the optimal solution to the Balanced-Coverage or the Network-Balanced-Coverage problems. In this case, $\alpha \leq 1$ is an approximation guarantee that better fits functions like ours. In this paper, we show that for the Balanced-Coverage problem, 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 $C(\cdot)$ is monotone and submodular. Unfortunately, the Network-Balanced-Coverage problem appears to be significantly harder and for that we only present a heuristic algorithm,

which works extremely well in our extensive experiments; designing an approximation algorithm for Network-Balanced-Coverage is an open problem. We note however, that both our algorithms follow the same generic design template — which we believe is interesting by itself. We also show that our algorithms admit a lot of practical speedups, which are a consequence of the structure of our objective function.

Our experimental results demonstrate that our algorithms are practical in terms of their running time, and they output assignments with high total task coverage and very low maximum load. Comparisons with a number of baselines inspired by existing works show that our algorithms consistently outperform them. In our experiments, we also compare the characteristics of the teams found by our algorithms for Balanced-Coverage and Network-Balanced-Coverage. Our findings are consistent with our expectation that the solutions to the Network-Balanced-Coverage problem are teams that are more cohesive in the graph that encodes the experts' ability to work together; that is, the teams found as solutions to Network-Balanced-Coverage have higher density in this graph.

2 Related Work

In this section, we highlight some related work in team formation and discuss its relationship to our problem and the algorithmic techniques we propose in this paper. To the best of our knowledge there is no other paper that addresses the exact Balanced-Coverage and Network-Balanced-Coverage problems we discuss here.

Team formation with a single task: A large body of work in team formation assumes that there is a single task, which requires 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. In the majority of this work (Bhowmik et al., 2014; Kargar et al., 2013; Kargar and An, 2011; Kargar et al., 2012; Lappas et al., 2009; Majumder et al., 2012; Li et al., 2015a,b, 2017; Rangapuram et al., 2013; Yin et al., 2018; Hamidi Rad et al., 2023; Kou et al., 2020; Berktaş and Yaman, 2021), the requirement that all skills of the tasks are covered is a hard constraint. Different problem formulations arise from the different definitions of the "goodness" of a team (i.e., small communication cost). The work by Kargar and An (2011) and Rangapuram et al. (2013) consider different graph communication costs in an offline setting to find a team of experts. However, these works consider single tasks with a complete coverage requirement, and consequently do not consider the trade-off between communication cost and expert workload. A subsequent related work by Kargar et al. (2013) considers a bi-criteria optimization for complete coverage of a single task, to minimize both the communication cost as well as the personnel cost of the teams formed. While this work has a similar flavor to ours, it is important to note that our Network-Balanced-Coverage problem formulation is a generalization of their work since we relax the complete coverage constraint and extend the offline scenario to forming teams for *multiple* tasks simultaneously.

More recently, there has been some work aiming to maximize a combined objective of task coverage minus the sum of the costs of the experts participating in the team (Nikolakaki et al., 2021; Dorn and Dustdar, 2010). In other words, the goal is to

maximize a submodular (i.e., coverage) minus a linear function. The setting is similar to ours and it could be expanded to consider multiple tasks. However, the linear part of the objective is more structured than the maximum load we are considering here. As a result, the algorithmic techniques that were developed by Nikolakaki et al. (2021) cannot be applied to our setting. On the other hand, the work of Dorn and Dustdar (2010) balances coverage with the team's communication cost on a graph. However, since their work considers only single tasks, their heuristics do not consider the workload of experts.

Team formation with multiple tasks: There is a number of papers that consider multiple tasks (Anagnostopoulos et al., 2010, 2012, 2018; Nikolakaki et al., 2020; Selvarajah et al., 2021), most of which focus on the *online* version of the problem, where tasks arrive in a streaming fashion. The offline versions of these problems are also **NP**-hard. Regardless of whether we study the offline or the online version of these problems, the setting is to minimize the load of the most loaded expert while covering completely all the skills in all tasks. Our setting is a relaxation of these problems aiming to maximize a combined objective of coverage minus load. Also, this line of work considers a minimization problem while in this paper we study a maximization problem, and therefore, the approximation bounds we seek are different.

Approximation framework: One of the intricacies of our objective function in the Balanced-Coverage and the Network-Balanced-Coverage problems 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 (Harshaw et al., 2019; Mitra et al., 2021). 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 our setting.

3 Problem Definitions

In this section, we describe our notation and basic concepts, and formally define the Balanced-Coverage and Network-Balanced-Coverage problems.

3.1 Preliminaries

Tasks, Experts and Skills. Throughout, we assume a set of m tasks $\mathcal{J} = \{J_1, \ldots, J_m\}$ and a set of n experts $\mathcal{X} = \{X_1, \ldots, X_n\}$. We also assume 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 task $J_j \subseteq S$ and for every expert $X_i \subseteq S$.

Assignments. An *assignment* of experts to tasks is represented by a binary matrix A, such that $A(i,j) = 1$ if expert X_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 one side corresponding to the experts and the nodes on the other side corresponding to the tasks; edge (i,j) exists if and only if $A(i,j) = 1$. Finally, we often view an assignment A as a *set* of its 1-entries.

Teams. Given an assignment A, we can find the set of m teams associated with A, denoted by $\mathcal{T}_A$, such that $T_j \in \mathcal{T}_A$ is the team of experts associated with task J_j: i.e.,

$T_j = \{X_i \mid A(i,j) = 1\}$. We use the additive skill model (Anagnostopoulos et al., 2010) to define the expertise of a team: a skill is covered by the team if there exists at least one member on the team who has that skill.

Task coverage. 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,

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

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:

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

Expert workload. Additionally, given an assignment A, we define the *load* of expert X_i in A as the number of tasks that X_i is assigned to. Formally,

$$L(X_i \mid A) = \sum_{j} A(i,j).$$

Given an assignment A, the *maximum load* among all experts is

$$L_{\max}(A) = \max_{i} L(X_i \mid A).$$

Coordination costs. We represent pairwise (symmetric) coordination costs between individual experts using edge weights on a graph $G = (\mathcal{X}, E)$. The vertices of G correspond to the set of experts, $\mathcal{X}$ and the edges, E are characterized by a metric distance function $d : E \to \mathbb{R}_{\geq 0}$. Although in the experimental section we discuss how $d(\cdot,\cdot)$ is computed, we point out here that we assume that there is a non-negative distance between any two experts; that is, $d(X_i, X_j) \geq 0$ for every $X_i \neq X_j$. We also assume that $d(\cdot,\cdot)$ is a metric.

Team radius and diameter. We first define the *radius* of a team T as $R(T) = \min_{X_i \in T} \max_{X_j \in T} d(X_i, X_j)$. The *diameter* $Diam(T)$ of a team T corresponds to the longest distance between any two experts on that team T, and is defined as $Diam(T) = \max_{X_i, X_j \in T} d(X_i, X_j)$. Since we consider a discrete metric space, it follows that: $\frac{1}{2} Diam(T) \leq R(T) \leq Diam(T)$.

Given an assignment A and the set of teams $\mathcal{T}_A$ associated with it, we define $R_{\max}(A) = \max_{T \in \mathcal{T}_A} R(T)$.

3.2 The Balanced-Coverage Problem

We now define the Balanced-Coverage problem as follows:

Problem 1 (Balanced-Coverage)**.** *Given a set of m tasks $\mathcal{J} = \{J_1, \ldots, J_m\}$ and a set of n experts $\mathcal{X} = \{X_1, \ldots X_n\}$ find an assignment A of experts to tasks such that*

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

is maximized.

The following observations provide some insight on our problem definition.

Observation 1: The objective function (see Eq. (3)) 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. Thus, the number of experts in the solution is not constrained in the definition of Balanced-Coverage itself.

Observation 2: The parameter λ is referred to as a *balancing coefficient*. Depending on the application, one may need to tune the importance of the two parts of the objective. The balancing coefficient λ should be thought of as a factor that adds flexibility to the model and allows for flexibility in the team-construction process. A detailed discussion on how we set the value of λ in practice is provided in Section 4.4.

Observation 3: The objective function $F(\cdot)$ is a summation of two quantities: coverage and maximum load. The coverage is a sum of normalized coverages multiplied by λ and therefore it is a quantity that takes real values between $[0, \lambda m]$; the value of 0 is achieved when no task is covered and the value λm is achieved when all tasks are fully covered. The maximum load is 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).

Observation 4: Finally, it can be shown that the first part of the objective, i.e., $C(A)$, is a monotone and submodular function. We state this in the following proposition:

Proposition 1. *The overall coverage function: $C(A) = \sum_{j=1}^{m} C(J_j \mid A)$ is a monotone and submodular function.*

The proof of this proposition is omitted as it is relatively simple: $C(\cdot)$ is a monotone submodular function as it is a summation of coverage functions that are known to be monotone and submodular (Krause and Golovin, 2014).

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. (2010) we can show that the Balanced-Coverage problem is **NP**-hard even when there are only two tasks.

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

Proof. We provide a proof of **NP**-hardness for $\lambda = 1$, via a reduction from the monotone satisfiability or MSat problem. The MSat problem is a version of satisfiability where clauses have only positive or only negative literals, and is known to be **NP**-hard (Lewis, 1983).

An instance of MSAT is specified by a set of clauses, each clause being a disjunction of literals that are all positive or all negative. Given an instance of the MSAT problem we create an instance of the BALANCED-COVERAGE problem, as follows.

- every clause C_ℓ in MSAT corresponds to a skill in our problem;
- every literal x_i in MSAT corresponds to an expert X_i in our problem; X_i has skills that correspond to the clauses in which x_i or its negation participates;
- we create two tasks $\mathcal{J} = \{J_1, J_2\}$; J_1 requires the skills that correspond to the clauses with positive literals and J_2 requires the skills that correspond to the clauses with negative literals.

We can show that the instance of the BALANCED-COVERAGE problem we have created has a solution of value 1 if and only if the corresponding instance of the MSAT problem has a satisfying assignment. For the one direction assume that there is a satisfying assignment in MSAT. For a literal x_i that is set to `true` the expert X_i is assigned only to J_1. For a literal x_i that is set to `false` the expert X_i is assigned only to J_2. All experts are assigned to exactly one task, and thus, $L_{\max} = 1$. Furthermore, both tasks are fully covered, and thus, the total coverage is 2. Therefore the value of the instance of the BALANCED-COVERAGE problem is $2 - 1 = 1$.

For the other direction assume that the BALANCED-COVERAGE objective is 1. Notice that the possible values for $L_{\max}$ are 0, 1, and 2. For the BALANCED-COVERAGE objective to be 1, the max load $L_{\max}$ can only be 1. Indeed, if $L_{\max} = 0$ or 2 the value of the objective is less than or equal to 0. When $L_{\max} = 1$ then for the objective to be 1, the total coverage should also be equal to 2. This only happens if there is an assignment of the experts to the two tasks such that each expert is assigned to exactly one task and each task is covered completely, which essentially means that there is a satisfying assignment to the MSAT problem. □

3.3 The Network-Balanced-Coverage Problem

We now define the NETWORK-BALANCED-COVERAGE problem as follows:

Problem 2 (NETWORK-BALANCED-COVERAGE)**.** *Given a set of m tasks $\mathcal{J} = \{J_1, \ldots, J_m\}$, a set of n experts $\mathcal{X} = \{X_1, \ldots, X_n\}$, a distance function $d(\cdot, \cdot)$ between any two experts, and a radius constraint r, find an assignment A of experts to tasks such that*

$$F(A) = \lambda C(A) - L_{\max}(A) \tag{4}$$

is maximized, and each task has a team of radius at most r, i.e., $R_{\max}(A) \leq r$.

Theorem 3. *The* NETWORK-BALANCED-COVERAGE *problem is* **NP***-hard even for $m = 2$ and any radius constraint r.*

The proof of Theorem 3 follows from the fact that the NETWORK-BALANCED-COVERAGE problem is a generalization of the BALANCED-COVERAGE problem.

4 Algorithms for Balanced-Coverage

The objective function $F(\cdot)$ of the BALANCED-COVERAGE problem is defined as the difference between a submodular function (*coverage*) and another function (*maximum*

Algorithm 1 The `ThresholdGreedy` algorithm.

Input: Set of m tasks $\mathcal{J}$, n experts $\mathcal{X}$, and λ
Output: An assignment of experts to tasks A

$A \leftarrow \emptyset, F_{\max} = 0$
for $\tau = 1, ..., m$ **do**
    Create the set of experts $\mathcal{X}_\tau$, with τ copies of each expert
    $A_\tau = \texttt{Greedy}(\mathcal{X}_\tau, \mathcal{J})$
    Compute $F_\tau = \lambda C(A_\tau) - \tau$
    if $F_\tau \geq F_{max}$ **then**
        $F_{max} = F_\tau$
        $A \leftarrow A_\tau$
    end if
end for
return A

load), which does not have a concrete form i.e., it is neither linear nor convex. Therefore, existing results on optimizing a submodular function (Nemhauser and Wolsey, 1978) or a submodular plus a linear or convex function (Harshaw et al., 2019; Mitra et al., 2021; Nikolakaki et al., 2021) are not applicable.

We describe `ThresholdGreedy`, a polynomial-time algorithm for the Balanced-Coverage problem. We show `ThresholdGreedy` outputs an assignment A such that:

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

or equivalently,

$$F(A) \geq \left(1 - \frac{1}{e}\right) F(OPT) - \frac{1}{e} L_{\max}(OPT). \tag{5}$$

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

The approximation guarantee described in Eq. (5) is a weaker form of approximation than standard multiplicative approximation guarantees. However, this is used in cases, like ours, where the objective function is not guaranteed to be positive (Harshaw et al., 2019; Mitra et al., 2021; Nikolakaki et al., 2021).

4.1 The `ThresholdGreedy` Algorithm

A key observation that `ThresholdGreedy` exploits is that the value of $L_{\max}$ is an integer in $[0, m]$, 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 $F(\cdot)$. The pseudocode is given in Algorithm 1.

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; A_τ is found by invoking the standard `Greedy` algorithm (Vazirani, 2013) — for optimizing a monotone submodular function — in order to optimize the overall coverage i.e., $C(\cdot)$. After trying all possible

m values of τ, we pick the assignment A_τ that has the maximum value of the objective $F(A_\tau)$.

The `Greedy` algorithm for solving the coverage problem for input experts $\mathcal{X}_\tau$ and tasks $\mathcal{J}$ (Line 4 of Algorithm 1) greedily assigns experts in $\mathcal{X}_\tau$ to tasks until there are no more experts available. At step $\ell+1$, `Greedy` 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*

$$\tilde{C}((i,j) \mid A^\ell) = C\left(A_\tau^\ell \cup (i,j)\right) - C\left(A_\tau^\ell\right) \tag{6}$$

is maximized. During this greedy assignment, each one of the τ copies of every expert is considered as a different expert and once a copy is assigned to a task the copy is removed from the candidate experts.

4.2 Approximation

Here, we prove our approximation result for `ThresholdGreedy`, as outlined already in Eq. (5). Before proving the main theorem we need the following lemma:

Lemma 1. *Let A_τ be the assignment of experts to tasks returned by* `Greedy` *(Line 4 of Alg. 1) for fixed threshold workload τ. Let OPT_τ be the optimal assignment of experts $\mathcal{X}_\tau$ 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 is similar to the proof that `Greedy` is an $\left(1-\frac{1}{e}\right)$-approximation algorithm to the coverage problem (Vazirani, 2013) and is thus omitted.

The above lemma states that for every threshold τ (i.e., for every iteration of `ThresholdGreedy`), the `Greedy` subroutine is guaranteed to return a solution that has good coverage with respect to the optimal solution for the coverage problem for this threshold τ. The lemma does not state anything about the final solution returned by `ThresholdGreedy`, or about the approximation with respect to the objective function $F(\cdot)$. We build upon the lemma and state the following theorem.

Theorem 4. *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:*

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

Proof. Let us assume that $L_{\max}(OPT) = \tau^*$. Note that $L_{\max}(A)$ may or may not be equal to τ^*. Then, we have the following:

$$F(A) \geq F(A_{\tau^*}) \qquad \text{(True for any } \tau\text{)}$$

$$
\begin{aligned}
&= \lambda C(A_{\tau^*}) - \tau^* \\
&\geq \left(1 - \frac{1}{e}\right) \lambda C(OPT_{\tau^*}) - \tau^* && \text{(Lemma 1)} \\
&\geq \left(1 - \frac{1}{e}\right) \lambda C(OPT) - \tau^* && (OPT_{\tau^*} \text{ is optimal for threshold } \tau^*) \\
&= \left(1 - \frac{1}{e}\right) \lambda C(OPT) - L_{\max}(OPT).
\end{aligned}
$$

□

4.3 Running Time and Speedup

A naive implementation of `ThresholdGreedy` has running time $\mathcal{O}(m^2n^2)$. It requires m calls to the `Greedy` routine in Line 4, which if implemented naively, takes time $\mathcal{O}(mn^2)$. Such a running time would make `ThresholdGreedy` impractical. Below, we discuss three methods that 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 (1978). In our experiments, we only use this lazy-evaluation version of `Greedy`.

Early termination of `ThresholdGreedy`: A computational bottleneck for `ThresholdGreedy` is its outer loop (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. This is because 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, the algorithm can safely terminate as the value of the objective will not improve for larger values of τ.

If we denote by A_τ the assignment produced at the τ-th iteration of `ThresholdGreedy` and by $C_\tau = C(A_\tau)$, then $F_\tau = C_\tau - \tau$. Using this notation, we have the following theorem.

Theorem 5. *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 $F_{\tau^*} \geq F_{\tau^*+1} \geq \ldots \geq F_m$.*

In order to prove Theorem 5, we rely on the properties of `ThresholdGreedy` as well as on the fact that the coverage function $C(\cdot)$ is monotone and submodular (Proposition 1). Recall that A_τ is the assignment produced at the τ-th iteration of `ThresholdGreedy` and $C_\tau = C(A_\tau)$. Then, by definition $F_\tau = C_\tau - \tau$. Moreover, the monotonicity and submodularity of the coverage function imply the following:[1]

Proposition 6. *The monotonicity of the overall coverage function implies that for every $\tau \in \{1, \ldots, m\}$: $C_\tau \geq C_{\tau-1}$.*

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

Proposition 7. *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 propositions 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 $A_{\tau+1}$.

We are now ready to prove Theorem 5.

Proof. Let us 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

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

Using Inequality (7) and Proposition 7, we have

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

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

$$\begin{aligned} C_{\tau^*} - \tau^* &\geq C_{\tau^*+1} - (\tau^* + 1) \\ (C_{\tau^*+1} - C_{\tau^*}) &\leq 1. \end{aligned} \tag{8}$$

Using Inequality (8) and Proposition 7, we have

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

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

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

□

We will call the value of τ for which $F(\cdot)$ gets maximized in the iterations of the `ThresholdGreedy` algorithm the *best-greedy workload* and the corresponding value of the objective the *best-greedy objective*.

Improving on linear search over workload values: The unimodality of the objective function as computed by `ThresholdGreedy` for the different values of τ, clearly allows us to try all possible values of τ starting from 1 until the value of F_τ stops increasing. This is a *linear search* over the different thresholds. We speedup this linear search by *combining an exponential with a linear search*. That is, we search over an exponentially increasing range of values of $\tau = 2^i$, for $i \geq 0$; once the objective function decreases for some i, we then perform a linear search over the range of workload values, $\tau \in [2^{i-1}, 2^i]$. In practice we observe that this technique significantly improves over the simple linear search.

Note that the unimodality of the objective function as computed by `ThresholdGreedy` for the different values of τ, would suggest a binary search over the values of τ. This type of search does not work well in practice because the running time of every iteration of `ThresholdGreedy` increases with the value of τ and the binary search requires trying (at least some) large values of τ. Thus in our experiments, we only use the combination of exponential and linear search we described above.

4.4 Tuning Coverage vs. Workload Importance

One must choose an appropriate value of the balancing coefficient, λ for each application, such that it tunes the relative importance of task coverage and expert workload as desired. In practice, we achieve this 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. There are two naive ways of implementing such a search process: The first is to run `ThresholdGreedy` (with all the speedup ideas 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 running `ThresholdGreedy` as many times as the different λs. The second solution requires running `ThresholdGreedy` once, but for *all* possible values of threshold $\tau = m$. Both these solutions are infeasible in practice even for datasets of moderate size. However, we make a key observation in Proposition 8, that enables us to efficiently search for an appropriate value for λ.

Proposition 8. *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 Proposition 6 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 $F^{\lambda_2}_{\tau_2} \geq F^{\lambda_2}_{\tau_1}$ and thus:

$$\lambda_2 C_{\tau_2} - \tau_2 \geq \lambda_2 C_{\tau_1} - \tau_1$$
$$\lambda_2 (C_{\tau_2} - C_{\tau_1}) \geq (\tau_2 - \tau_1).$$

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. □

An efficient search on the values of λ: Using Proposition 8 we can explore the solutions of `ThresholdGreedy` for different values of $\lambda \in \Lambda \subseteq \mathbb{R}_+$ efficiently, by running `ThresholdGreedy` only once and – at the same time – exploiting the early termination trick we discussed in Section 4.3.

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 *best-greedy* values for smaller values of λ, and plot the corresponding values of $C(A)$ and $L_{\max}(A)$ for each λ value. 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 result in a significant increase in coverage. Thus, a suitable value of λ can be identified by visual inspection, 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 $L_{\max}(A)$. Note that the λ value can be adjusted as needed, as per the requirements of the application domain.

5 Algorithms for Network-Balanced-Coverage

In this section, we introduce `NThreshold`, our algorithm for solving the Network-Balanced-Coverage problem. The pseudo-code of the algorithm is shown in Algorithm 2. Conceptually, the algorithm is similar to `ThresholdGreedy`. More specifically, `NThreshold` considers all values of load $\tau = 1, \ldots, m$. For each value τ, the algorithm forms candidate teams (`CandidateTeams`) that satisfy the radius constraint and then it assigns teams to tasks (`AssignTeams`). This assignment may cause some experts to violate the load constraint imposed by τ, thus, an additional pruning step (`TeamPruning`) is needed to ensure that the load constraint is not violated. Finally, `NThreshold` returns the assignment corresponding to the best objective found across the different workload values τ.

In the rest of the section, we describe each one of the steps of `NThreshold` in detail and discuss all computational issues that arise.

5.1 Forming Candidate Teams

First, we form a set of candidate teams $\mathcal{T}$ such that the each team in $\mathcal{T}$ has a radius that satisfies the specified radius constraint r; this is done in Line 2 of Algorithm 2. We

Algorithm 2 The `NThreshold` algorithm.

Input: Set of m tasks $\mathcal{J}$, n experts $\mathcal{X}$, graph $G = (\mathcal{X}, \mathbf{E})$ with coordination costs, radius constraint r, and λ.

Output: An assignment A of experts to tasks.

$A \leftarrow \emptyset, F_{\max} = 0$
$\mathcal{T} \leftarrow$ `CandidateTeams`$(\mathcal{X}, G, r)$
for $\tau = 1, ..., m$ **do**
    $A_\tau \leftarrow$ `AssignTeams`$(\mathcal{J}, \mathcal{T}_\tau, \tau)$
    $A'_\tau \leftarrow$ `TeamPruning`(A_τ, τ)
    $F_\tau \leftarrow \lambda C(A'_\tau) - \tau$
    if $F_\tau \geq F_{max}$ **then**
        $F_{max} = F_\tau$
        $A \leftarrow A'_\tau$
    end if
end for
return A

pursue two alternatives for forming candidate teams, which we call `CandidateTeams-R` and `CandidateTeams-AllR` and which we describe below.

`CandidateTeams-R`: Given a set of n experts $\mathcal{X}$, a graph $G = (\mathcal{X}, E)$ with their coordination costs, and a radius constraint r, `CandidateTeams-R`$(\mathcal{X}, G, r)$ forms n teams, one team T_i for each expert X_i. Team T_i consists of expert X_i and all other experts X_j with $d(X_i, X_j) \leq r$. That is, $T_i = X_i \cup \{X_j \mid d(X_i, X_j) \leq r\}$. This method runs in time $\mathcal{O}(n^2)$ and creates n candidate teams.

`CandidateTeams-AllR`: Here, we consider several different radii $0 < r' \leq r$; for each r' we invoke `CandidateTeams-R` and form n teams corresponding to radius constraint r'. In practice, we form teams of varying sizes by splitting the interval $(0, r]$ into k parts of size r/k, and choosing k different values for $r' \in \{r/k, 2r/k, \ldots, r\}$. `CandidateTeams-AllR` returns kn candidate teams, and its running time is $\mathcal{O}(kn^2)$.

5.2 Assigning Teams to Tasks

Before we describe our general algorithm for assigning teams to tasks, we consider a special case, where every team consists of one expert and the task is to assign experts to tasks. In this case, the team-assignment problem can be written as a linear program as follows: let $x_{ij} = 1$ if expert i is assigned to task j, and let C_{ij} denote the fraction of skills required by task J_j covered by expert i. The linear program (LP) is the following:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} C_{ij} x_{ij}, \\ \text{such that} \quad & \sum_{i=1}^{n} x_{ij} \leq 1, \qquad \text{for all } 1 \leq j \leq m, \end{aligned}$$

$$\sum_{j=1}^{m} x_{ij} \leq \tau \quad \text{for all } 1 \leq i \leq n, \text{ and}$$
$$0 \leq x_{ij} \leq 1.$$

Note that due to the *unimodular* nature of the constraints the above LP only has integer solutions, i.e., in the optimal solution it is $x_{ij} \in \{0, 1\}$, for all (i, j) (Papadimitriou and Steiglitz, 1998).

Therefore, when teams consist of one expert, the team-assignment problem can be solved optimally in polynomial time. Additionally, the above LP works in cases when there is a pre-specified set of teams $\mathcal{T} = \{T_1, \ldots, T_\ell\}$. The solution obtained by the LP in this case guarantees that each task is assigned to at most one team, and each team is assigned to at most τ tasks. However, since the teams may have arbitrary overlap among their experts, there is no guarantee for the number of tasks assigned to a single expert. We consider the solution of the above LP for teams, even if it violates the per-expert load constraint. To ensure compatibility with the load constraints, we then prune the teams so that each expert has load at most τ (see next section).

In practice, we solve the `AssignTeams` task shown in Algorithm 2 either by solving the LP we described above using a readily-available solver like Gurobi (Gurobi Optimization, LLC, 2023), or by a greedy algorithm that greedily matches a team to a task that maximizes the objective and does not violate any of the constraints. Such a greedy assignment is a 2-approximation algorithm to the problem described by the LP (Khan et al., 2016) and it runs in time $\mathcal{O}(m^2)$. We note though that the Gurobi solver works extremely well in practice.

Clearly, given an assignment of teams to tasks, we can generate a corresponding assignment A of experts to tasks as follows: for each task a team is assigned to, all experts on that team have a 1-entry in the corresponding column in A.

5.3 Pruning Teams

As the assignment A returned by `AssignTeams` may violate the load constraint τ for individual experts, we prune the assignment by removing experts from teams in order to guarantee that the load of each individual is τ or less. For this, we invoke the following `TeamPruning` step in Line 5 of Algorithm 2.

The pseudocode for the `TeamPruning` routine is presented in Algorithm 3. The pruning algorithm takes as input an assignment A of experts to tasks, and the load constraint τ. It then removes (or un-assigns) experts from tasks until all experts satisfy the workload constraint τ.

In order to explain `TeamPruning`, we introduce the idea of *coverage loss*, which we define to be the amount of coverage of a task that is lost when an expert is removed from the team assigned to that task. First, we obtain the set of all overloaded experts that need to be pruned. Then for each task that the expert is assigned to, we compute the loss in coverage by removing the expert from that team. We add these coverage-loss values to a priority queue. Subsequently, we prune experts from tasks in order of increasing coverage loss from the priority queue, until all experts satisfy the workload

Algorithm 3 The `TeamPruning` algorithm

Input: Assignment A and workload constraint τ
Output: Pruned Assignment A'

$A' \leftarrow A$
$\mathcal{X}_\tau \leftarrow$ Set of experts in A with workload greater than τ
Initialize a priority queue to store coverage losses for expert-task pairs
for each expert X in $\mathcal{X}_\tau$ **do**
    for each team T expert X is on **do**
        for each task J team T is assigned to **do**
            Compute loss in coverage of task J by removing expert X from team T.
            Insert expert-task coverage loss into priority queue.
        end for
    end for
end for
while Any expert X in $\mathcal{X}_\tau$ violates workload constraint τ **do**
    $A' \leftarrow$ Prune expert-task pair from A using priority queue.
    Recompute coverage losses of experts on pruned team.
end while
return A'

constraint, τ. Every time we remove an expert from a task, we recompute the coverage losses of all other experts that were assigned to that task.

The worst-case running time of `TeamPruning` is $\mathcal{O}(n^2 m)$; in practice, this is significantly faster as it is not usually necessary to prune the entire priority queue.

5.4 Approximation

Although `NThreshold` performs well in practice, we have no formal approximation guarantees for its performance. Part of the reason for this is that the subproblem of assigning a set of pre-formed teams (i.e., the ones formed by `CandidateTeams`) to tasks such that the coverage is maximized, while the load of each individual expert is below a threshold τ is an **NP**-hard problem itself. We prove this in Appendix A.

This observation does not mean that Network-Balanced-Coverage cannot be approximated; it simply means that `NThreshold` as it is designed in Algorithm 2 cannot have provable approximation bounds.

5.5 Running Time and Speedups

In this section, we discuss the running time of `NThreshold` and propose some practical speedups. Note that a naive implementation of the `NThreshold` algorithm would have a running time $\mathcal{O}(m^2 n^2)$. Since the `NThreshold` algorithm computes the same objective as `ThresholdGreedy`, we can exploit some of the speedup techniques from Sec. 4.3.

Early Termination of `NThreshold`: We make use of Theorem 5, and do not consider all m values of τ. The value of the objective function F_τ as computed by `NThreshold` for the different values of τ is a unimodal function, and once a maximum is found for

some value of τ, the algorithm can safely terminate as the value of the objective will not improve for larger values of τ.

Improving on Linear search over workload values: As in `ThresholdGreedy`, in Line 3 of Algorithm 2 we search over an exponentially increasing range of values of $\tau = 2^i$, for $i \geq 0$; once the objective function decreases for some i, we then perform a linear search in the range $\tau \in [2^{i-1}, 2^i]$. In practice, this technique significantly improves the performance of the method, over the simple linear search.

5.6 Instantiating the `NThreshold` Algorithm

We specify here the naming convention we use for different variants of the `NThreshold` algorithm, depending on how we choose to implement the subroutines: `CandidateTeams` (i.e., `CandidateTeams-R` or `CandidateTeams-All`) and `AssignTeams` (i.e., `AssignTeams-LP` or `AssignTeams-Greedy`), we call the corresponding versions of `NThreshold`: `NThreshold-R-LP`, `NThreshold-R-Greedy`, `NThreshold-All-LP` and `NThreshold-All-Greedy` respectively; `TeamPruning` is always invoked.

5.7 Tuning Coverage vs. Workload Importance

Similar to the technique used for the `ThresholdGreedy` algorithm in Section 4.4, depending on the application, we choose an appropriate value of the balancing coefficient, λ such that it balances the relative importance of task coverage and expert workload. We call the value of τ for which $F(\cdot)$ gets maximized in the iterations of the `NThreshold` algorithm the *best-network workload* and the corresponding value of the objective the *best-network objective*. We can then make use of Proposition 8, but modified with the best-network workload and best-network objective, and follow the technique in Section 4.4 to graphically select an appropriate λ value that gives the most desirable trade-off between the coverage and the workload.

6 Experiments

We experimentally evaluate our algorithms for both Balanced-Coverage and Network-Balanced-Coverage using real-world datasets. We compare our algorithms with other heuristics, inspired by related work. In the end of the section, we also compare the solutions obtained by `ThresholdGreedy` and `NThreshold`, aiming to provide additional insight on the differences and the similarities of the two methods.

Our implementation is in Python and available online.[2] For all our experiments we use single-process implementation on a 64-bit MacBookPro with an AppleM1Pro CPU and 16GB RAM.

6.1 Experiments for Balanced-Coverage

In this section we first introduce our datasets and baselines, and then discuss our experiments for the Balanced-Coverage problem. We show how we choose the

[2] https://github.com/kvombatkere/Team-Formation-Code

balancing coefficient λ for each dataset, and then evaluate the performance of `ThresholdGreedy` and baselines in terms of the objective, expert load and running time.

6.1.1 Datasets

We evaluate our methods on several real-world datasets; some of these datasets have been used in past team-formation papers (Anagnostopoulos et al., 2010; Nikolakaki et al., 2020, 2021). A short description of the datasets follows, while their statistics are shown in Table 1.

IMDB: The data is obtained from the International Movie Database.[3] We simulate a team-formation setting where movie directors conduct auditions for movie actors: we assume that movie genres correspond to skills, movie directors to experts, and actors to tasks. The set of skills possessed by a director or actor is the union of genres of the movies they have participated in. In order to experiment with datasets of different sizes, we create three data instances by selecting all movies created since 2020, 2018 and 2015. From these movies we select the directors that have at least one actor in common with at least one other director, and then randomly sample 1000, 3000 and 4000 directors, to form the set of experts in the 3 datasets. Then we randomly sample 4000, 10000 and 12000 actors, to form the set of tasks. We refer to these datasets as *IMDB-1*, *IMDB-2* and *IMDB-3*, respectively.

Bibsonomy: This dataset comes from a social bookmark and publication sharing system with a large number of publications, each of which is written by a set of authors (Benz et al., 2010). Each publication is associated with a set of *tags*; we filter tags for stopwords and use the 1000 most common tags as skills. We simulate a setting where certain prolific authors (experts) conduct interviews for other less prolific authors (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 to be those with at least 12 skills. We create three datasets by selecting all publications since 2020, 2015 and 2010. From these publications we select the prolific authors that have at least one paper in common with at least one other prolific author, and then randomly sample 500, 1500 and 2500 prolific authors to form the set of experts in the 3 datasets. Then we randomly sample 1000, 5000 and 9000 non-prolific authors, to form the set of tasks. We refer to these datasets as *Bbsm-1*, *Bbsm-2*, *Bbsm-3*, respectively.

Freelancer and *Guru:* These two datasets consist of random samples of real jobs that are posted by users online, and a random sample of real freelancers, in the *Freelancer*[4] and *Guru* [5] online labor marketplaces respectively. The data consists of tasks that require certain discrete skills, and experts who possess discrete skills. The Freelancer data we use consists of 993 jobs (i.e. tasks) that require skills and 1212 freelancers (i.e. experts) that have skills; we refer to this dataset as *Freelancer*. Similarly, the Guru data we use consists of 3195 tasks that require skills and 6120 experts that have certain skills; we refer to this dataset as *Guru*.

[3] https://www.imdb.com/interfaces/
[4] freelancer.com
[5] guru.com

Table 1: Summary statistics of our datasets.

Dataset	Experts	Tasks	Skills	skills/ expert	skills/ task
IMDB-1	1000	4000	24	2.2	2.0
IMDB-2	3000	10000	25	2.4	2.2
IMDB-3	4000	12000	26	2.8	2.8
Bbsm-1	500	1000	957	13.0	4.8
Bbsm-2	1500	5000	997	13.6	4.9
Bbsm-3	2500	9000	997	13.6	4.9
Freelancer	1212	993	175	1.5	2.9
Guru	6120	3195	1639	13.1	5.2

6.1.2 Baselines

Motivated by existing work, we use the following three algorithms as baselines:

`LPCover`: This algorithm is an application of the offline Linear Programming rounding (LP-rounding) algorithm discussed by Anagnostopoulos et al. (2010). Using their LP formulation, the goal is to obtain a fractional assignment of experts to tasks such that every task is fully covered and the maximum load is minimized. Once a fractional assignment is obtained (let X_{ij} be the fractional assignment of expert i to task j), a rounding scheme is provided that operates in logarithmic number of rounds; in each round we independently assign expert i to task j with probability X_{ij}. It can be shown that at the end of rounding each task is fully covered with high probability and the load achieved is a logarithmic approximation to the optimal load. In our case, we proceed with the same LP, but in every iteration of the rounding phase, we check the value of our objective and we only keep the solution that has the best value. Our LP has mn variables and $\mathcal{O}(mn)$ constraints. If T is the running time for the LP then the overall running time of `LPCover` is $\mathcal{O}(T + mn)$. For our experiments we use Gurobi (Gurobi Optimization, LLC, 2023) and we observe that `LPCover` is significantly slower than the other baselines.

`TaskGreedy`: This algorithm is inspired by the previous work of Nikolakaki et al. (2020). `TaskGreedy` iterates 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 $\mathcal{O}(mn)$ since there are n experts available for each of the m tasks.

`NoUpdateGreedy`: This algorithm is a simple modification of `ThresholdGreedy`: 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

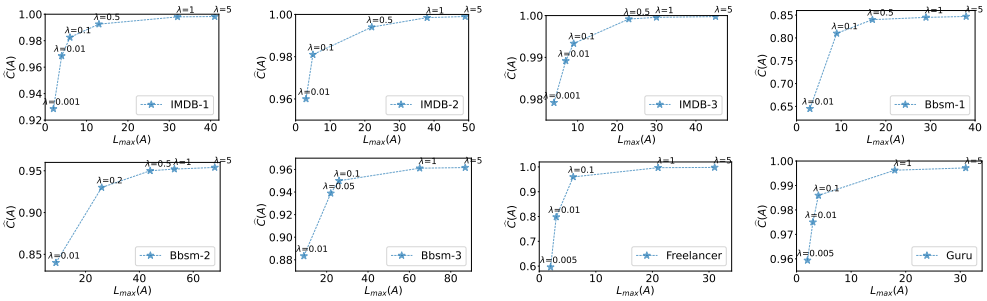


Fig. 2: The best-greedy workload $L_{\max}(A)$ value and the coverage $C(A)$ corresponding to the best-greedy objective $F^{\lambda}(A)$ computed by `ThresholdGreedy`. Each subplot shows a range of values of the balancing coefficient λ for each dataset.

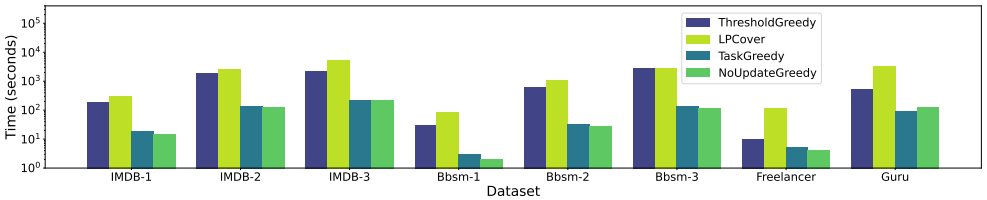


Fig. 3: Running time (in seconds) of `ThresholdGreedy` and baseline algorithms, in logarithmic scale.

initial marginal-gain values to iteratively add expert-task edges (i, j) in decreasing order of their $v(i, j)$ values, without ever updating them. 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 $\mathcal{O}(mn \log(mn))$, since there are mn total expert-task edges, and sorting these edges takes time $\mathcal{O}(\log(mn))$.

In all cases, we perform a grid search over the values of all hyperparameters and we report the best results for each algorithm and each dataset.

6.1.3 Tuning Coverage and Workload Importance

Before showing our experimental results, we discuss how we set the balancing coefficient λ, following the techniques described in Section 4.4. 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 *best-greedy* values for smaller values of λ, and plot the corresponding values of $C(A)$ and $L_{\max}(A)$ for each λ value. Fig. 2 shows these scatter plots for each dataset. In most of our datasets we experimented with relatively small values of $\lambda \in (0, 5]$. We then visually inspect these plots 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 $L_{\max}(A)$. The values of λ we picked for the different datasets are shown besides the dataset name in Table 2.

Table 2: Experimental performance of `ThresholdGreedy` and baseline algorithms in terms of the objective F^λ, the maximum load $L_{\max}$ and the average task coverage $\widehat{C} = \frac{1}{m}C$. The best values for each dataset are in bold.

Dataset (λ)	ThresholdGreedy			LPCover			TaskGreedy			NoUpdateGreedy		
	F^λ	$L_{\max}$	$\widehat{C}$	F^λ	$L_{\max}$	$\widehat{C}$	F^λ	$L_{\max}$	$\widehat{C}$	F^λ	$L_{\max}$	$\widehat{C}$
IMDB-1 (0.1)	**388**	**6**	**0.98**	295	72	0.92	318	45	0.91	191	150	0.85
IMDB-2 (0.1)	**972**	**5**	**0.98**	845	123	0.97	852	95	0.94	636	298	0.94
IMDB-3 (0.1)	**1184**	**9**	**0.99**	1099	89	0.99	922	222	0.95	957	200	0.96
Bbsm-1 (0.1)	**72**	**9**	**0.81**	65	16	0.81	23	12	0.31	24	18	0.3
Bbsm-2 (0.2)	**900**	**29**	**0.93**	848	65	0.91	350	33	0.39	330	67	0.4
Bbsm-3 (0.1)	**827**	**27**	**0.95**	723	97	0.91	330	91	0.47	323	109	0.48
Freelancer (0.1)	**88**	**6**	0.95	59	32	0.92	63	36	**0.99**	25	50	0.76
Guru (0.1)	**311**	**4**	**0.99**	287	25	0.98	225	30	0.80	17	33	0.16

6.1.4 Evaluation

We show the comparative performance of all four algorithms, in terms of the objective function (F^λ), the average coverage $\widehat{C} = \frac{1}{m}C$, and the maximum load $L_{\max}$, in Table 2. Intuitively, a *good* solution to an instance of the Balanced-Coverage problem is an assignment A that not only maximizes the overall task coverage but also minimizes the maximum load of the assignment. Our experiments for `ThresholdGreedy` show that it performs the best, compared to all our baselines, in terms of the objective across all datasets. Additionally, it finds assignments with a low maximum workload and it runs in a reasonable amount of time, even for datasets with several thousand experts and tasks. Note that for different datasets we use different values of λ; however, `ThresholdGreedy` finds the highest overall task coverage *independently* of the value of λ, and consequently would also outperform the baselines for other λ values as well.

Objective values F and workload $L_{\max}$: As we can observe in Table 2, `ThresholdGreedy` consistently finds the assignment with the best objective value. On average, across all datasets `ThresholdGreedy` performs about 15% better than `LPCover` and 55% better than `TaskGreedy` and `NoUpdateGreedy`. As the datasets get larger, the superior performance of `ThresholdGreedy` becomes more evident. This behavior may be attributed to our algorithm finding solutions with significantly lower $L_{\max}$.

`LPCover` is consistently the second-best algorithm in terms of the objective function. It also performs particularly well on the *IMDB-2*, *IMDB-3* and *Guru* datasets — it returns objective values that are comparable (but lower) to those returned by `ThresholdGreedy`. `TaskGreedy` and `NoUpdateGreedy` perform relatively well on the *IMDB* and *Freelancer* datasets — they return objective values that are within 20% of the objective value of `ThresholdGreedy`. In general, we observe that these baselines perform reasonably well on smaller datasets: one explanation is that 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. However, while the baselines often achieve an overall task coverage of 90%, `ThresholdGreedy` achieves superior task coverage in the majority of the cases.

In terms of maximum workload, `ThresholdGreedy` consistently finds the assignment with the lowest maximum workload value across all our experiments; the baselines return maximum load values that are significantly larger than those returned by `ThresholdGreedy`. On average across all datasets `ThresholdGreedy` finds a maximum load value that is 80% smaller than the maximum workload values returned

by the baselines. This is because, in an attempt to maximize the overall task coverage, the baselines make costly assignments of experts to tasks. While we do see some examples of reasonable workload values (e.g., for the *Guru* dataset), in most cases the workload values returned by the baselines would be infeasible in practice.

Running time: While `ThresholdGreedy` has a theoretical running time of $\mathcal{O}(m^2n^2)$, the speed-up techniques discussed in Section 4.3 and Section 4.4 lead to significantly lower running time in practice. Fig. 3 shows a bar plot with the running time of all algorithms for each dataset in logscale. For the smaller datasets (e.g., *Freelancer* and *Bbsm-2*), we observe that the running time of `ThresholdGreedy` is on the order of a few seconds. Even for the largest datasets (e.g., *Bbsm-3* and *IMDB-3*) the running time of our algorithm is within a few hours. We also observe that `TaskGreedy` and `NoUpdateGreedy` are faster than our algorithm, but `LPCover` is slower, due to the computational bottleneck of solving an LP with a large number of variables. Note that the running time of the baselines as we report them here do not include the grid search we performed in order to tune their hyperparameters.

6.2 Experiments for Network-Balanced-Coverage

We start by explaining our datasets, introducing a baseline algorithm and showing how we choose the balancing coefficient λ for each dataset. We then empirically evaluate the performance of `NThreshold` in terms of the objective, expert load, radius constraint and running time. We also compare its performance with `ThresholdGreedy`.

6.2.1 Datasets

We follow the method of Anagnostopoulos et al. (2012) and create social graphs with expert coordination costs for our datasets, *IMDB*, *Bbsm*, *Freelancer*, and *Guru*.

For the *IMDB* dataset, we create a social graph among the directors, who form the vertices in the graph. We connect directors using actors as intermediaries: we form and edge between two directors if they have directed at least two distinct actors in common. The cost of the edge is set to e^{-fD}, where D is the number of distinct actors directed by the two directors. The distance function e^{-fD} takes values between 0 and 1, and we note that it quickly converges to the value 0 as the number of common actors D between two directors increases. As in Anagnostopoulos et al. (2012), we set the value of the parameter $f = \frac{1}{10}$ since this value of f yields a reasonable edge-weight distribution of coordination costs in the social graph for our *IMDB* dataset.

For the *Bbsm* dataset, we create a social graph among authors using co-authorship to define the strength of social connection. Two authors are connected with an edge if they have written at least one paper together. Again the cost of the edge is set to e^{-fD}, where D is the number of distinct papers coauthored by the two authors. Similar to Anagnostopoulos et al. (2012), we set the value of the parameter $f = \frac{1}{10}$, so as to obtain a reasonable distribution of edge-weights in our *Bbsm* social graph.

For the *Freelancer* and *Guru* datasets, we use the following heuristic to create a social graph among the experts in each dataset: experts with similar, overlapping sets of skills have a lower coordination cost since they are "closer" to each other in terms of their ability to perform tasks well together. To create the expert social graphs, we

Table 3: Summary statistics of our graph datasets.

Dataset	Number of nodes	Average path length	Average degree
IMDB-1	1000	7.6	1.4
IMDB-2	3000	4.2	4.5
IMDB-3	4000	3.4	8.0
Bbsm-1	500	2.6	3.1
Bbsm-2	1500	1.4	25.8
Bbsm-3	2500	1.4	29.1
Freelancer	1212	1.2	19.2
Guru	6120	1.1	42.0

consider each pair of experts, and compute the Jaccard distance between the sets of skills of the pair of experts. The cost of the edge between each pair of experts is then represented by the Jaccard distance between their skill sets. We note that the Jaccard distance takes values between 0 and 1, and is 0 if two experts have identical skill sets, and 1 if their skills are mutually exclusive.

For all datasets, we keep the same names as before and we present the summary graph statistics of these datasets in Table 3. The average path length corresponds to the average shortest path length between all pairs of nodes in the graph, and the average degree is the average of the *unweighted* degrees of all nodes in the graph.

6.2.2 Baseline

We use the following greedy variant of the `NThreshold` algorithm as a baseline.

`GreedyIndividual:` This algorithm has a similar logic as `NThreshold` as it iterates over different workloads. However, `GreedyIndividual` does not create candidate teams. The algorithm assigns individual experts to tasks in a greedy manner: for each of the nm expert-task pairs, we consider the task coverage the expert provides for that task. We then greedily assign experts to tasks by selecting experts in order of decreasing coverage they provide for tasks. As we assign experts to tasks, we also ensure that each expert satisfies the workload constraint τ. Note this baseline has a computational overhead of checking that every new expert assigned to a task satisfies the radius constraint r with respect to all other experts already assigned to that task. `GreedyIndividual` has a running time of $\mathcal{O}(mn^2)$.

6.2.3 Tuning Coverage and Workload Importance

In this section we discuss how the value of λ is selected. We follow a similar technique as the previous section and determine the *best-network workload* and the corresponding value of the *best-network objective* for a large value of lambda, $\lambda = 5$. We then compute the *best-network* values for smaller values of $\lambda \in (0, 5]$, and plot the corresponding values of $C(A)$ and $L_{\max}(A)$ for each λ value. The scatter plots for $r = 0.3$ and $r = 0.7$ are visualized in Figures 4 and 5, respectively.

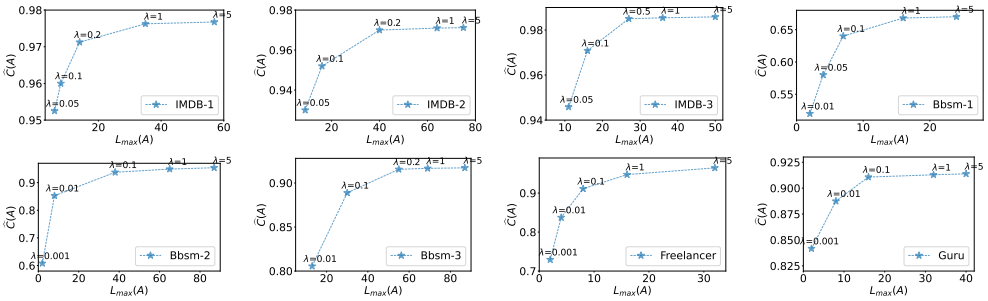


Fig. 4: The best-greedy workload $L_{\max}(A)$ value and the coverage $C(A)$ corresponding to the best-greedy objective $F^{\lambda}(A)$ computed by `NThreshold-R-LP` for $r = 0.3$. Each subplot shows a range of values of the balancing coefficient λ for each dataset.

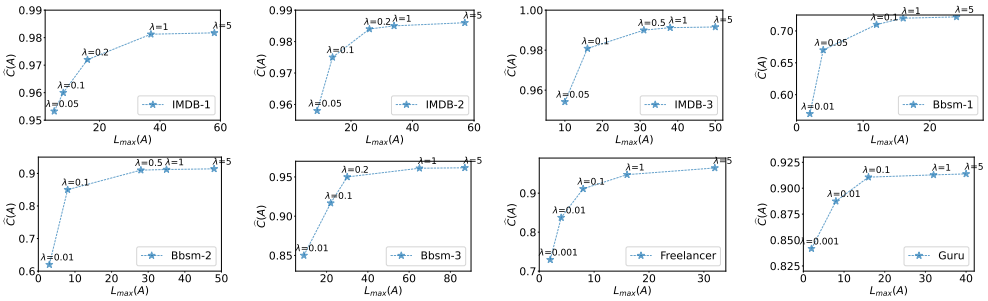


Fig. 5: The best-greedy workload $L_{\max}(A)$ value and the coverage $C(A)$ corresponding to the best-greedy objective $F^{\lambda}(A)$ computed by `NThreshold-R-LP` for $r = 0.7$. Each subplot shows a range of values of the balancing coefficient λ for each dataset.

We visually inspect these plots to identify a suitable value of λ such that the best-network workload and best-network objective values yield a high value for $C(A)$, while simultaneously giving a reasonably low value for $L_{\max}(A)$. The final λ values we selected are shown besides the different datasets in Table 4.

6.2.4 Evaluation

In this section, we evaluate the performance of the different instantiations of `NThreshold` we described in Section 5.6. Specifically, we evaluate `NThreshold-R-LP`, `NThreshold-R-Greedy` and `NThreshold-All-LP` and compare their performance with each other, and with the `GreedyIndividual` baseline. We compare the algorithms using the objective function (F^{λ}), the average coverage per skill $\widehat{C} = \frac{1}{m}C$, and the maximum load $L_{\max}$. We omit the results for `NThreshold-All-Greedy` since `NThreshold-All-LP` outperformed it in all aspects.

Since the coordination costs of our datasets have values between 0 and 1 in our datasets, we ran the algorithms for several values of the radius constraint $r \in$

$\{0.1, 0.3, 0.5, 0.7, 0.9\}$. We observed that $r \in \{0.1, 0.3\}$ yielded similar objective values, coverages, and workloads, as did $r \in \{0.5, 0.7, 0.9\}$. Consequently, we only report results in Table 4 for $r = 0.3$ and $r = 0.7$.

Objective values F and workload $L_{\max}$: From Table 4, we observe that both `NThreshold-R-LP` and `NThreshold-R-Greedy` perform very well for all datasets in terms of the average task coverage $\widehat{C}$. For *IMDB* (for both $r = 0.3$ and $r = 0.7$) we observe high coverage values greater than or equal to 0.95. Additionally, these algorithms find reasonably low expert workloads of $L_{\max} \in [13, 17]$.

We observe that `NThreshold-All-LP` and `GreedyIndividual` also perform well on *IMDB* in terms of average coverage, with coverage values greater than 0.92 for $r = 0.3$, and coverage values greater than 0.95 for $r = 0.7$. However we observe that `Greedy-Individual` returns significantly higher workload values of $L_{\max} \geq 31$. Similarly, for *IMDB-2* and *IMDB-3*, we observe that `NThreshold-All-LP` also returns higher workload values of $L_{\max} \geq 23$.

For *Bbsm*, we observe that the `NThreshold` algorithms have the highest F^λ and $\widehat{C}$ values and also the lowest $L_{\max}$ values (for both $r = 0.3$ and $r = 0.7$). `Greedy-Individual` yields a significantly lower coverage with a much higher expert workload. We note that for *Bbsm-2*, `NThreshold-All-LP` gives the best results in terms of the objective F^λ and coverage values. However the `NThreshold` algorithms only perform marginally worse in terms of the objective and have similar workload values.

For the *Freelancer* and *Guru* datasets, we observe that `NThreshold-R-LP` yields the best F^λ and $\widehat{C}$, with workload values $L_{\max} \leq 15$.

Effect of radius constraint: We observe that $\widehat{C}$ decreases slightly for all algorithms, across our datasets as the radius constraint decreases from $r = 0.7$ to $r = 0.3$. This is expected since a smaller radius implies that the potential teams of experts available is also smaller. We observe, however, that the difference is marginal, with a decrease in coverage of less than 3%. We observe that the maximum workload values returned by the `NThreshold` algorithms are also comparable for the different radius constraints. These observations lead us to conclude that the increase in coverage due to increasing team radius could be attributed to the availability of new experts that are within the new, larger team radius.

Mean expert workloads: We examine the teams formed by our algorithm in terms of the *mean* of the expert workloads. For *IMDB*, we have that $L_{\max} \in [13, 17]$, yet the mean mean expert load of the `NThreshold` solutions is in the range $[3.9, 4]$. This indicates that while there are a few experts who are heavily loaded, on average the `NThreshold` algorithms find good load-balancing solutions. In contrast, we observe that the baseline `GreedyIndividual` has a higher mean expert load for the *IMDB* datasets, in the range $[5.5, 6]$.

Similarly, we observe that for *Bbsm-1* and *Bbsm-2* the mean expert load of the `NThreshold` algorithms is in the range $[1.8, 2]$ for both datasets (and for both radius constraints). On the other hand, the mean expert load of `GreedyIndividual` is higher for these datasets in the range $[2.7, 3]$. A similar pattern was observed for the *Freelancer* and *Guru* datasets as well.

Comparison with `ThresholdGreedy`: We compare the performance of `NThreshold` with `ThresholdGreedy` by comparing values in Tables 2 and 4. While $\widehat{C}$ returned by

Table 4: Experimental performance of `NThreshold` and `GreedyIndividual` in terms of the objective F^{λ}, the maximum load $L_{\max}$ and the average task coverage $\widehat{C} = \frac{1}{m}C$. The best values for each dataset are in bold.

	Dataset (λ)	NThreshold-R-Greedy			NThreshold-R-LP			NThreshold-All-LP			GreedyIndividual		
		F^{λ}	$L_{\max}$	$\widehat{C}$	F^{λ}	$L_{\max}$	$\widehat{C}$	F^{λ}	L	$\widehat{C}$	F^{λ}	L	$\widehat{C}$
$r = 0.3$	*IMDB-1* (0.2)	752	**14**	**0.97**	**752**	**14**	0.96	756	8	0.92	742	34	0.93
	IMDB-2 (0.1)	960	15	0.95	**961**	**13**	**0.96**	942	26	0.94	938	33	0.95
	IMDB-3 (0.1)	1149	17	**0.97**	**1153**	**15**	0.95	1143	23	0.95	1138	22	0.94
	Bbsm-1 (0.1)	**56**	**7**	**0.64**	**56**	8	**0.64**	54	9	0.63	46	16	0.62
	Bbsm-2 (0.01)	26	9	0.86	27	**7**	0.86	**29**	8	**0.89**	25	9	0.84
	Bbsm-3 (0.1)	767	34	**0.89**	**785**	**32**	**0.89**	748	34	0.87	738	37	0.86
	Freelancer (0.1)	80	9	0.89	**82**	**8**	**0.9**	78	11	0.87	74	14	0.80
	Guru (0.1)	268	**11**	0.86	**272**	15	**0.9**	256	28	0.88	241	15	0.76
$r = 0.7$	*IMDB-1* (0.2)	761	16	**0.97**	762	15	0.96	**766**	**8**	**0.97**	748	31	0.97
	IMDB-2 (0.1)	960	15	**0.97**	**963**	**14**	**0.97**	943	30	0.96	941	31	0.97
	IMDB-3 (0.1)	1159	17	**0.98**	**1161**	**15**	0.97	1147	28	0.97	1143	35	0.96
	Bbsm-1 (0.05)	**30**	**4**	0.67	**30**	6	**0.68**	29	4	0.65	15	16	0.63
	Bbsm-2 (0.1)	426	8	0.87	428	8	0.87	**438**	**8**	**0.90**	332	32	0.72
	Bbsm-3 (0.2)	1677	32	**0.95**	**1685**	**31**	**0.95**	1638	37	0.93	1360	61	0.79
	Freelancer (0.1)	82	9	**0.91**	**83**	**8**	**0.91**	81	8	0.90	77	17	0.84
	Guru (0.1)	271	**12**	0.89	**275**	15	**0.91**	260	30	0.90	242	14	0.78

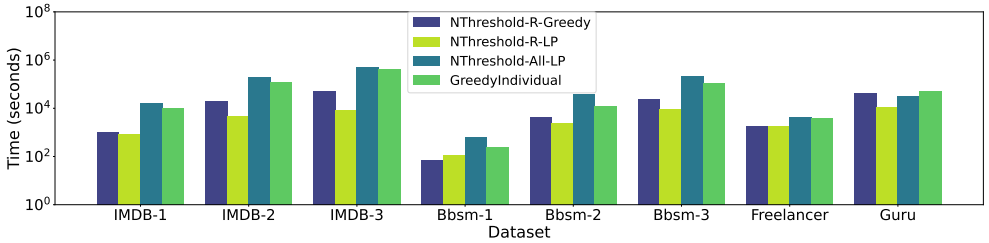


Fig. 6: Running time (in seconds) of `NThreshold` and `GreedyIndividual`, in logarithmic scale for radius constraint $r = 0.7$.

both algorithms is comparable, we see that `ThresholdGreedy` finds a slightly higher coverage across all the datasets. Additionally, the maximum workload values achieved by `NThreshold` is higher than those achieved by `ThresholdGreedy`. This is because the problem solved by the former algorithms is harder than the one solved by the latter; there are more constraints in terms of how experts can be combined into teams.

Running time: We record the total running time of all algorithms for $r = 0.7$ (we observed similar patterns for the other radii values) and illustrate them in Fig. 6. We observe that `NThreshold-R-LP` has the best running time of all algorithms, and this is closely followed by `NThreshold-R-Greedy`. While `NThreshold-All-LP` does perform well on some of the datasets, we observe that it has the maximum running time of all algorithms, across all datasets. This is an expected result since `NThreshold-All-LP` considers many more candidate teams than the other algorithms.

6.3 Team Characteristics

In this section, we investigate the characteristics of the teams formed by `ThresholdGreedy`, `NThreshold` and `GreedyIndividual`. We examine four characteristics: team size, team radii, within-team degree distributions and average pairwise distance

Table 5: Team characteristics of `ThresholdGreedy`, `NThreshold` and `GreedyIndividual` algorithms in terms of the average team size $\overline{S}$, maximum team size $S_{\max}$ and average team radius $\overline{R}$.

Dataset	ThresholdGreedy			NThreshold-R-Greedy			NThreshold-R-LP			GreedyIndividual		
	$\overline{S}$	$S_{\max}$	$\overline{R}$	$\overline{S}$	$S_{\max}$	$\overline{R}$	$\overline{S}$	$S_{\max}$	$\overline{R}$	$\overline{S}$	$S_{\max}$	$\overline{R}$
IMDB-1	1.09	4	0.99	1.04	3	0.68	1.04	3	0.66	1.02	2	0.67
IMDB-2	1.06	3	0.99	1.03	3	0.65	1.05	4	0.64	1.02	2	0.62
IMDB-3	1.06	4	0.99	1.04	4	0.64	1.04	3	0.64	1.03	2	0.62
Bbsm-1	2.05	6	0.99	1.71	11	0.64	2.15	13	0.66	1.42	12	0.59
Bbsm-2	1.94	8	0.99	2.89	48	0.69	3.05	53	0.68	1.77	13	0.63
Bbsm-3	1.79	8	0.99	8.68	109	0.69	9.70	167	0.69	2.07	19	0.68
Freelancer	1.87	5	0.98	8.57	88	0.69	8.59	88	0.69	2.21	6	0.63
Guru	2.06	12	0.98	14.06	85	0.69	13.21	84	0.69	1.56	9	0.65

Table 6: Team characteristics of `ThresholdGreedy`, `NThreshold` and `GreedyIndividual` algorithms in terms of the average team density $\overline{\delta}$, and average team pairwise distance $\overline{\rho}$.

Dataset	ThresholdGreedy		NThreshold-R-Greedy		NThreshold-R-LP		GreedyIndividual	
	$\overline{\delta}$	$\overline{\rho}$	$\overline{\delta}$	$\overline{\rho}$	$\overline{\delta}$	$\overline{\rho}$	$\overline{\delta}$	$\overline{\rho}$
IMDB-1	1.02	0.52	1.07	0.41	1.07	0.38	1.05	0.34
IMDB-2	1.01	0.51	1.06	0.34	1.08	0.35	1.03	0.31
IMDB-3	1.01	0.51	1.08	0.34	1.07	0.35	1.06	0.31
Bbsm-1	1.02	0.59	1.40	0.48	1.51	0.52	1.37	0.38
Bbsm-2	1.04	0.58	1.13	0.65	1.35	0.61	1.52	0.45
Bbsm-3	1.04	0.57	1.52	0.74	1.38	0.75	1.29	0.56
Freelancer	1.15	0.56	2.53	0.76	2.49	0.76	2.89	0.40
Guru	2.80	0.58	9.76	0.85	10.92	0.78	2.82	0.34

of the experts in the formed teams In the remainder of the section, we discuss the team characteristics in detail. For this analysis, we consider $r = 0.7$, but similar characteristics were observed for other radii as well. We report the average values of the different characteristics for each algorithm and dataset in Tables 5 and 6.

Team size: We characterize the size of a team (for a task) by the total number of experts assigned to that task. In Table 5, we report the average team size formed by the different algorithms for the different datasets.

Overall, we observe that across all datasets, `ThresholdGreedy` consistently finds teams with the smallest sizes and highest task coverage. This is intuitive, since this algorithm doesn't have any graph constraints to satisfy. We also observe that `ThresholdGreedy` has a smaller variance in team sizes, since even the largest teams formed are significantly smaller than those formed by the `NThreshold` algorithms.

For the *IMDB* datasets, all algorithms yield relatively small teams, with an average team size of a little over 1 expert. The smaller team sizes in the *IMDB* datasets could be attributed to the fact that there are relatively few skills in this dataset; often a single director is able to cover the skills of the tasks – which typically have fewer skills.

For the *Bbsm* datasets, particularly *Bbsm-3*, `ThresholdGreedy` and `GreedyIndividual` have smaller average team sizes than the `NThreshold` algorithms. While most teams formed by the `NThreshold` algorithms are relatively small with under 10 experts, we observe that there are some teams that are much larger. We observe similar patterns for *Freelancer* and *Guru*, where `ThresholdGreedy` finds teams that are smaller on average, with lower variance in the size than the `NThreshold` algorithms.

Team radii: We observe that `ThresholdGreedy` has teams with a much larger radius, of almost 1, since there is no radius constraint for `ThresholdGreedy`. For *IMDB* and *Bbsm*, we observe that the `NThreshold` algorithms form most of their teams with radii that are just below the $r = 0.7$ constraint. We observe that for *Bbsm*, the `NThreshold` and `GreedyIndividual` algorithms have mean team radii of about 0.6, and several teams with radii less than 0.5; this is not the case for `ThresholdGreedy`.

For *Freelancer* and *Guru*, we see that the `NThreshold` algorithms form more teams of varying radii, but still have average radii of about 0.6. The teams formed by `ThresholdGreedy` for these datasets still have the largest team radii, with means of 0.92 and 0.96, respectively (See Table 5).

Team densities: We define the density $\overline{\delta}$ of a team to be the sum of degrees of all experts in that team divided by the total number of experts on that team. This measure quantifies how well-connected the output teams are. In Table 6, we report the average team density $\overline{\delta}$ achieved by the different algorithms. While we observed that the `NThreshold` algorithms formed slightly larger teams than `ThresholdGreedy`, we now see that the former also outputs denser teams on average.

Team pairwise distances: We define the mean pairwise distance $\overline{\rho}$ of a team to be the mean of all pairwise shortest paths of experts on that team. The average pairwise distance of a team gives us an indication of how well connected experts on a team are. In Table 6, we report the mean pairwise distance $\overline{\rho}$ of teams formed by the different algorithms. We observe that for all three *IMDB* datasets and *Bbsm-1*, `ThresholdGreedy` has the highest team mean pairwise distance of all the algorithms. However, we observe that for *Bbsm-2*, *Bbsm-3*, *Freelancer* and *Guru*, the `NThreshold` algorithms have a higher team mean pairwise distance.

Overall, we observe that `ThresholdGreedy` forms teams with fewer experts and smaller variance in team size compared to `NThreshold`. On the other hand, the `NThreshold` algorithms form more compact teams than `ThresholdGreedy` in terms of the radii of teams. Additionally, the teams formed by the `NThreshold` algorithms are significantly denser in terms of their connections between team members.

7 Conclusions

In this paper, we two new team-formation problems: Balanced-Coverage and the more general Network-Balanced-Coverage problem; we also designed algorithms for solving them.

In Balanced-Coverage the objective 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. We proved that Balanced-Coverage is NP-hard. We adopted a weaker notion of approximation (Harshaw et al., 2019; Mitra et al., 2021), tailored for our objective, and – within this setting – we designed a polynomial-time approximation algorithm, `ThresholdGreedy` for Balanced-Coverage.

In the Network-Balanced-Coverage problem, we expand our Balanced-Coverage formulation to include communication costs in a social graph. We have the same objective with the added constraint that every team in the expert-task

assignment, A must also satisfy a radius constraint $R_{\max}(A) \leq r$. This problem is a generalization of Balanced-Coverage, and thus also NP-hard. For Network-Balanced-Coverage, we designed `NThreshold` a practical algorithm for solving it. This algorithm follows the same high-level algorithmic ideas we used for `ThresholdGreedy`, yet it does not come with approximation guarantees.

For both problems, we showed that 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. Finally, we demonstrated the practical utility of the algorithmic framework we proposed in a variety of real datasets and also compared the characteristics of teams formed by the `ThresholdGreedy` and `NThreshold` algorithms. Our experiments with a variety of datasets from various domains demonstrated the utility of our framework and the efficacy of our algorithms.

Acknowledgments: Evimaria Terzi and Karan Vombatkere are supported by NSF grants III 1908510 and III 1813406 as well as a gift from Microsoft. Aristides Gionis is supported by ERC Advanced Grant REBOUND (834862), EC H2020 RIA project SoBigData++ (871042), and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., Leonardi, S.: Power in unity: forming teams in large-scale community systems. In: ACM Conference on Information and Knowledge Management, CIKM, pp. 599–608. ACM, ??? (2010)

Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., Leonardi, S.: Online team formation in social networks. In: WWW (2012)

Anagnostopoulos, A., Castillo, C., Fazzone, A., Leonardi, S., Terzi, E.: Algorithms for hiring and outsourcing in the online labor market. In: Guo, Y., Farooq, F. (eds.) ACM SIGKDD, pp. 1109–1118. ACM, ??? (2018)

Bhowmik, A., Borkar, V., Garg, D., Pallan, M.: Submodularity in team formation problem. In: SDM (2014)

Kargar, M., Zihayat, M., An, A.: Finding affordable and collaborative teams from a network of experts. In: SDM (2013)

Kargar, M., An, A.: Discovering top-k teams of experts with/without a leader in social networks. In: CIKM (2011)

Kargar, M., An, A., Zihayat, M.: Efficient bi-objective team formation in social networks. In: ECML PKDD (2012)

Lappas, T., Liu, K., Terzi, E.: Finding a team of experts in social networks. In: KDD

(2009)

Majumder, A., Datta, S., Naidu, K.: Capacitated team formation problem on social networks. In: KDD (2012)

Li, C.-T., Shan, M.-K., Lin, S.-D.: On team formation with expertise query in collaborative social networks. KAIS (2015)

Li, L., Tong, H., Cao, N., Ehrlich, K., Lin, Y.-R., Buchler, N.: Replacing the irreplaceable: Fast algorithms for team member recommendation. In: WWW (2015)

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

Rangapuram, S.S., Bühler, T., Hein, M.: Towards realistic team formation in social networks based on densest subgraphs. In: WWW (2013)

Yin, X., Qu, C., Wang, Q., Wu, F., Liu, B., Chen, F., Chen, X., Fang, D.: Social connection aware team formation for participatory tasks. IEEE Access (2018)

Harshaw, C., Feldman, M., Ward, J., Karbasi, A.: Submodular maximization beyond non-negativity: Guarantees, fast algorithms, and applications. In: International Conference on Machine Learning, ICML, pp. 2634–2643 (2019)

Mitra, S., Feldman, M., Karbasi, A.: Submodular + concave. CoRR **abs/2106.04769** (2021)

Hamidi Rad, R., Fani, H., Bagheri, E., Kargar, M., Srivastava, D., Szlichta, J.: A variational neural architecture for skill-based team formation. ACM Transactions on Information Systems **42**(1), 1–28 (2023)

Kou, Y., Shen, D., Snell, Q., Li, D., Nie, T., Yu, G., Ma, S.: Efficient team formation in social networks based on constrained pattern graph. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE), pp. 889–900 (2020). IEEE

Berktaş, N., Yaman, H.: A branch-and-bound algorithm for team formation on social networks. INFORMS Journal on Computing **33**(3), 1162–1176 (2021)

Nikolakaki, S.M., Ene, A., Terzi, E.: An efficient framework for balancing submodularity and cost. In: Zhu, F., Ooi, B.C., Miao, C. (eds.) ACM SIGKDD, pp. 1256–1266. ACM, ??? (2021)

Dorn, C., Dustdar, S.: Composing near-optimal expert teams: a trade-off between skills and connectivity. In: CoopIS (2010)

Nikolakaki, S.M., Cai, M., Terzi, E.: Finding teams that balance expert load and task coverage. CoRR **abs/2011.04428** (2020)

Selvarajah, K., Zadeh, P.M., Kobti, Z., Palanichamy, Y., Kargar, M.: A unified framework for effective team formation in social networks. Expert Systems with Applications **177**, 114886 (2021)

Krause, A., Golovin, D.: Submodular function maximization. Tractability **3**(71-104), 3 (2014)

Lewis, H.R.: Michael r. πgarey and david s. johnson. computers and intractability. a guide to the theory of np-completeness. wh freeman and company, san francisco1979, x+ 338 pp. The Journal of Symbolic Logic **48**(2), 498–500 (1983)

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

Vazirani, V.V.: Approximation Algorithms. Springer, ??? (2013)

Minoux, M.: Accelerated greedy algorithms for maximizing submodular set functions. In: Optimization Techniques, pp. 234–243. Springer, ??? (1978)

Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Courier Corporation, ??? (1998)

Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023). https://www.gurobi.com

Khan, A., Pothen, A., Mostofa Ali Patwary, M., Satish, N.R., Sundaram, N., Manne, F., Halappanavar, M., Dubey, P.: Efficient approximation algorithms for weighted b-matching. SIAM Journal on Scientific Computing **38**(5), 593–619 (2016)

Benz, D., Hotho, A., Jäschke, R., Krause, B., Mitzlaff, F., Schmitz, C., Stumme, G.: The social bookmark and publication management system BibSonomy. The VLDB Journal **19**(6), 849–875 (2010) https://doi.org/10.1007/s00778-010-0208-4

Håstad, J.: Clique is hard to approximate within n 1- ε (1999)

A The Teams-Matching Problem

Consider the following problem, which is solved for every threshold τ in each iteration of the `NThreshold` algorithm: given a set of preformed teams $\mathcal{T}$ find an assignment of teams to tasks such that the sum of task coverages is maximized while every expert's load is below a pre-specified threshold τ. We call this problem Teams-Matching and we formally define it as follows:

Problem 3 (Teams-Matching)**.** *Consider a set of m tasks $\mathcal{J} = \{J_1, \ldots, J_m\}$, a set of n experts $\mathcal{X} = \{X_1, \ldots X_n\}$, and a set of t teams $\mathcal{T} = \{T_1, \ldots, T_t\}$, such that $T_i \subseteq \mathcal{X}$. Also assume a load threshold τ. Let $x_{kj} = 1$ if team T_k is assigned to task J_j and let C_{kj} denote the fraction of skills required by J_j covered by team T_k. Finally, let $M(i,k) = 1$ if expert X_i is in team T_k, and $M(i,k) = 0$ if X_i is not in T_k. Our problem can be written as the following integer program:*

$$
\begin{aligned}
\textit{maximize} \quad & \sum_{k=1}^{t}\sum_{j=1}^{m} C_{kj}x_{kj} \\
\textit{such that} \quad & \sum_{k=1}^{t} x_{kj} \leq 1, && \textit{for all } 1 \leq j \leq m, \\
& \sum_{k=1}^{t} M(i,k) \sum_{j=1}^{m} x_{kj} \leq \tau && \textit{for all } 1 \leq i \leq n, \textit{ and} \\
& x_{kj} \in \{0,1\}.
\end{aligned}
$$

Theorem 9. *The* Teams-Matching *problem is* **NP***-hard.*

Proof. We provide a proof of **NP**-hardness via a reduction from Independent Set. Given an undirected graph $G = (V, E)$ and an integer $\sigma \geq 0$, the decision version of the Independent Set problem is to determine if there exists a subset of nodes $V' \subseteq V$ such that no two nodes in V' share an edge, and $|V'| \geq \sigma$. For the purposes of this reduction, we consider the decision version of Teams-Matching with an integer $\sigma \geq 0$, where the objective is to determine if their exists a solution to the above program such that $\sum_{k=1}^{t}\sum_{j=1}^{m} C_{kj}x_{kj} \geq \sigma$, i.e., the sum of task coverages is at least σ.

Given an arbitrary undirected graph $G = (V, E)$ with $|V| = t$ vertices, we create an instance `TM` of Teams-Matching in polynomial time. The idea is to map vertices in G to teams in `TM`, and edges between pairs of vertices in G correspond to *distinct* experts shared by the corresponding teams in `TM`. We give the details below.

- For each vertex $v \in V$ in G, create a team with a *single* expert with 1 *unique* skill that is only associated with this expert in `TM`. Thus, create t teams $\{T_1, \ldots, T_t\}$.
- For every edge $(v, v') \in E$ in G, add *one* expert with *zero* skills i.e $\{\emptyset\}$, to both the teams in `TM` that correspond to the vertices v and v' in G.
- Create t tasks $J_1, \ldots, J_t$ to have exactly the set of skills corresponding to the t teams $T_1, \ldots, T_t$ in `TM`. This way, there is an 1-1 correspondence between team T_i and task J_i and we call J_i the *corresponding* task for team T_i.

Observe that TM has $|E| + t$ experts, t tasks, and t teams, such that each task has *exactly* one team that covers its skills *completely*. Additionally, each team (and each task) has exactly one unique skill. Consider the k-th team and j-th task: if $k = j$, the coverage $C_{kj} = 1$, and if $k \neq j$, then $C_{kj} = 0$. Then it immediately follows that, given an assignment A that has a total task coverage $\sigma \geq 0$, there exists a set of σ teams such that these σ teams were assigned to their *corresponding* tasks.

We now show that the Teams-Matching instance (TM) we have created has a team-task assignment A with total task coverage of σ and load $\tau = 1$ if and only if the corresponding instance of Independent Set has an independent set of size σ in G.

Independent Set → Teams-Matching. Assume that G has an independent set of size σ, i.e., it is possible to select σ vertices in G such that no two vertices share a common edge. This corresponds to picking σ teams in the Teams-Matching instance, such that no two teams share a expert. Since by our construction of the problem instance each team has exactly one corresponding task that can be fully covered, we can assign each of the σ teams to its corresponding task and achieve a total task coverage of σ, within the load constraint $\tau = 1$.

Teams-Matching → Independent Set. Assume we have an assignment A for our Teams-Matching instance with total task coverage of $\sigma \geq 0$, and every expert is assigned to at most $\tau = 1$ tasks. Now we can find an independent set of size σ in G as follows. We know that there exists a set of σ teams that have been assigned to their corresponding tasks, to achieve a coverage of σ. Since A satisfies the load constraint $\tau = 1$, each expert is assigned to at most one task, and consequently no teams share an expert. Thus, if we select the σ vertices in the Independent Set instance that correspond to the σ tasks in the Teams-Matching instance, we have an independent set of size σ, such that no two vertices share a common edge. □

Corollary 1. Teams-Matching *cannot be approximated to within a factor better than* $\mathcal{O}(n^{1-\epsilon})$.

Proof. For any $\epsilon > 0$, Independent Set cannot be approximated to within a factor better than $\mathcal{O}(n^{1-\epsilon})$ in polynomial time unless $\mathbf{P} = \mathbf{NP}$ (Håstad, 1999).

We consider the optimization versions of Independent Set and Teams-Matching; the goal is to find the maximum independent set in an undirected graph $G = (V, E)$ in the former, and to find a feasible assignment such that the sum of coverages is maximized with load constraint $\tau = 1$ in the latter. We use the subscripts IS and TM to refer to instances x, solutions y, and the objective functions m of these problems respectively. OPT denotes the optimal value of the solution to either problem.

The reduction in Theorem 9 describes a function that maps an instance $x_{\texttt{IS}}$ of Independent Set to an instance $x_{\texttt{TM}}$ of Teams-Matching. Given a solution $y_{\texttt{TM}}$, Theorem 9 describes a function that maps $y_{\texttt{TM}}$ back into a solution $y_{\texttt{IS}}$ of Independent Set. Note that $OPT(x_{\texttt{IS}}) = OPT(x_{\texttt{TM}})$ and $m_{\texttt{TM}}(x_{\texttt{TM}}, y_{\texttt{TM}}) = m_{\texttt{IS}}(x_{\texttt{IS}}, y_{\texttt{IS}})$, since for any Teams-Matching instance with (optimal) coverage of σ the corresponding instance of Independent Set has a (maximal) independent set of size σ.

Towards a contradiction, assume Teams-Matching can be approximated to within a factor γ in polynomial time, such that $\gamma > \mathcal{O}(n^{1-\epsilon})$. Thus, there exists an instance $x_{\texttt{TM}}$, and a solution $y_{\texttt{TM}}$, such that $\gamma OPT(x_{\texttt{TM}}) \leq m_{\texttt{TM}}(x_{\texttt{TM}}, y_{\texttt{TM}})$. Then, we have:

$$\gamma \leq \frac{m_{\texttt{TM}}(x_{\texttt{TM}}, y_{\texttt{TM}})}{OPT(x_{\texttt{TM}})} = \frac{m_{\texttt{IS}}(x_{\texttt{IS}}, y_{\texttt{IS}})}{OPT(x_{\texttt{TM}})} \leq \frac{\mathcal{O}(n^{1-\epsilon})OPT(x_{\texttt{IS}})}{OPT(x_{\texttt{TM}})} = \mathcal{O}(n^{1-\epsilon}),$$

which is a contradiction. Thus, we conclude that Teams-Matching cannot be approximated to within a factor better than $\mathcal{O}(n^{1-\epsilon})$. □