Balancing Task Coverage and Expert Workload in Team Formation

Karan Vombatkere[*] Evimaria Terzi[†]

Abstract

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

1 Introduction

The abundance of online and offline labor markets (e.g., Amazon mechanical turks, Guru, Freelancer, online scientific collaborations etc.) has motivated a lot of work on *team formation*.

Most of the existing work in team formation takes as input a task (or a collection of tasks) that requires a set of skills and a set of experts, where each expert also has a set of skills. Then the goal is to identify one team for every task, such that for every skill of the task there is at least one member of the team that has it. Thus, the majority of the work on team formation requires *complete* coverage of the skills of the input tasks [1, 2, 3, 7, 12, 10, 11, 13, 17, 14, 16, 15, 23, 25]. Their differences lie in the way they define the "goodness" of a team. For example, in some cases they optimize for the communication cost, while in other cases they optimize for the load of the experts or their compensation cost.

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

$$F(A) = C(A) - L_{\max}(A), \tag{1.1}$$

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

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

Although we show that the two terms of the objective (Eq. (1.1)) are comparable, we believe that there may be applications where coverage is more important than maximum load or vice versa. Therefore, we expand our framework with a *balancing coefficient* which enables an effective tuning of the importance of the two parts of the objective.

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

$$C(A) - L_{\max}(A) \geq \alpha C(\text{OPT}) - L_{\max}(\text{OPT}), \tag{1.2}$$

[*]Boston University, Boston, MA, {kvombat@bu.edu}

[†]Boston University, Boston, MA, {evimaria@bu.edu}

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

Our experimental results demonstrate that our algorithm is practical in terms of its running time and outputs assignments with high task coverage and very low maximum load. Comparisons with a number of baselines inspired by existing work show that our algorithm consistently outperforms them.

There is some other recent work that also formalizes the team formation problem with partial coverages [8, 21, 22]. We discuss in detail the relationship between these works and ours in Section 2. The take-away message of this discussion is that although these works consider partial coverage, their setting and their objectives are significantly different from ours and thus we cannot use their algorithms to solve our problem.

Main contributions:

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

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

2 Related Work

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

Team formation with complete task coverage: The majority of the work in team formation assumes that there is a single task, or a collection of tasks, which require a set of skills. Additionally there are experts who possess a subset of skills. The goal is to identify a "good" subset of the experts that collectively cover *all* the skills required by the task – or each task of the collection. In the majority of the research[1, 2, 3, 7, 12, 10, 11, 13, 17, 14, 16, 15, 23, 25], 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, small maximum workload etc.).

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

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

Also related is the work of Nikolakaki et al. [21]. Similar to the problem we study here, that work also considers partial task coverage and aims to optimize for coverage and maximum workload of the experts simultaneously. However, that problem is different from ours because it is is stated as a minimization problem: the goal there is to minimize the maximum workload and the fraction of tasks that remain uncovered. In contrast, we express the Balanced Coverage problem as a maximization problem. While we find our objective function more intuitive, it is also challenging to optimize – because it can potentially take negative values. Consequently, we adopt an approximation framework that is more appropriate for such functions and then design approximation algorithms tailored to this maximization objective. On the other hand, Nikolakaki et al. provide a set of intuitive heuristics, which have no formal approximation guarantees.

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

3 Problem Definition

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

3.1 Preliminaries Throughout, we assume a set of m tasks $\mathcal{J} = \{J_1, \ldots, J_m\}$ and a set of n experts $\mathcal{E} = \{E_1, \ldots, E_n\}$. We also assume 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 $E_i \subseteq S$.

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

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

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

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

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

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

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

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

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

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

3.2 Problem definition 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 $\mathcal{E} = \{E_1, \ldots E_n\}$ find an assignment A of experts to tasks such that*

$$F(A) = C(A) - L_{max}(A) \tag{3.3}$$

is maximized.

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

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

Observation 2: The objective function (see Eq. 3.3) 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 the Balanced Coverage problem itself.

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

Proposition 3.1. *The overall coverage function: $C(A) = \sum_{j=1}^{m} C(J_j \mid A)$ is a monotone and submodular function.*

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

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

Theorem 3.1. *The* Balanced Coverage *problem is NP-hard even for $m = 2$.*

4 The ThresholdGreedy Algorithm

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

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

(4.4)
$$C(A) - L_{\max}(A) \geq \left(1 - \frac{1}{e}\right) C(\text{OPT}) - L_{\max}(\text{OPT}),$$

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

The approximation guarantee described in Eq. (4.4) is a weaker form of approximation than standard multiplicative approximation guarantees. However, this is standard in cases, like ours, where the objective function is not guaranteed to be positive [9, 19, 22].

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 $F()$. The pseudocode is given in Algorithm 1.

Algorithm 1 The `ThresholdGreedy` algorithm.

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

In more detail, given a threshold τ on the value of $L_{\max}$, any expert can be used at most τ times. Conceptually, this means that there are τ copies of every expert and we find A_τ to be the `Greedy` assignment corresponding to τ; A_τ is found by invoking the standard `Greedy` algorithm [24] – for optimizing a monotone submodular function – in order to optimize the overall coverage i.e. $C()$. 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 $\mathcal{E}_\tau$ and tasks $\mathcal{J}$ (Line 4 of Algorithm 1) greedily assigns experts in $\mathcal{E}_\tau$ 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*

$$\tilde{C}((i,j) \mid A^{\ell}) = C\left(A_\tau^{\ell} \cup (i,j)\right) - C\left(A_\tau^{\ell}\right) \tag{4.5}$$

is maximized. In this process, 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.

4.1 Approximation properties Here, we prove our approximation result for `ThresholdGreedy`, as captured already in Eq. (4.4). Before proving the main theorem we need the following lemma:

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

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

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

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

THEOREM 4.1. *Let A be the assignment returned by `ThresholdGreedy` and let OPT be the optimal assignment for the* BALANCED COVERAGE *problem. Then we have the following approximation:*

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

The proof of this theorem is provided in the supplementary material.

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

Lazy greedy instead of greedy: Instead of using the naive implementation of `Greedy`, we deploy the lazy-evaluation technique introduced by Minoux [18]. The details of how we implement it are given in the supplementary material. In our experiments, we only use this lazy-evaluation version of `Greedy`.

Early termination of `ThresholdGreedy`: Another computational bottleneck for `ThresholdGreedy` is its outer loop (line 2 in Alg. 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 4.2. *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 $F_{\tau^*} \geq F_{\tau^*+1} \geq \ldots \geq F_m$.*

The proof of this theorem is given in the supplementary material.

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

5 Tuning Coverage and Workload Importance

Up to this point, we have assumed that the objective function is of the form $F(A) = C(A) - L_{\max}(A)$. However, sometimes one might need to weight one of the two terms more heavily than the other. We can do so by adding a *balancing coefficient* $\lambda > 0$ so that the new enhanced objective becomes:

$$F^{\lambda}(A) = \lambda C(A) - L_{\max}(A). \tag{5.6}$$

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

Finding the assignment that optimizes the enhanced objective as given in Eq. (5.6) can still be done using ThresholdGreedy; for a given λ the only modification one has to make to the algorithm is to change line 5 to take the value of λ into account.

However, in each application one must choose an appropriate value of λ such that it balances 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 speedups we proposed in Section 4.2) for the different values of λ. The second is to run ThresholdGreedy *without* the early termination technique we discussed in Section 4.2 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.

An efficient search on the values of λ: We show that 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 4.2. This is based on the following observation:

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

The proof of this proposition is given in the supplementary material.

6 Experiments

In this section, we evaluate the performance of ThresholdGreedy using real-world datasets. We compare its performance with three baseline algorithms in terms of the objective function $F()$ and the running time. We observe that ThresholdGreedy performs the best 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.

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

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

IMDB : The data is obtained from the International Movie Database [1]. The original dataset contains information about movies, TV shows and documentaries along with the associated actors, directors, movie year and movie genre(s). We simulate a team-formation setting where movie directors conduct auditions for movie actors. Thus, 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 an actor is the union of movie genres of the movies they have participated in. In order to experiment with datasets of different sizes, we created three instances from the original data by selecting all movies created since 2015, 2018 and 2020. We refer to these datasets as *IMDB*-15, *IMDB*-18 and *IMDB*-20 respectively.

Bibsonomy: Our second dataset comes from bibsonomy, a social bookmark and publication sharing system [6]. The original dataset consists of a large number of publications, each of which is written by a set of authors. Each publication is associated with a set of *tags*. We filter tags for stopwords and use the 1000 most common tags as skills in our experiments. We use this data to simulate a setting where certain expert authors conduct interviews for other less prolific authors. Thus, the prolific authors become the experts and the rest are the tasks. An author's skills are the union of the tags associated with their publications. Upon inspection of the distribution of skills among all authors we determine prolific authors (experts) to be those with at least 15 skills. As before, we created three datasets by selecting all publications since 2010, 2015 and 2020 and extracting their authors. We refer to these datasets as *Bbsm*-10, *Bbsm*-15, *Bbsm*-20 respectively.

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

Guru: Similar to *Freelancer*, this dataset also

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

Table 1: Summary statistics of our datasets.

Dataset	Experts	Tasks	Skills	skills/ expert	skills/ task
IMDB-15	5551	18109	26	2.4	3.1
IMDB-18	3871	13183	26	2.1	2.7
IMDB-20	2176	7858	25	1.9	2.4
Bbsm-10	3044	21981	1000	13.7	4.8
Bbsm-15	1904	9061	1000	10.9	4.3
Bbsm-20	177	834	858	11.5	3.6
Freelancer	1212	993	175	1.5	2.9
Guru	6120	3195	1639	13.1	5.2

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

6.2 Baselines 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. [1]. 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 such a fractional assignment of experts to tasks is obtained (let X_{ij} be the fractional assignment of expert i to task j), they have a rounding scheme that proceeds in logarithmic number of rounds and in each round it independently assigns expert i to task j with probability X_{ij}. They show that at the end of this rounding phase every 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. Also, for some of our datasets not all tasks can be fully covered (i.e., they require some skills that do not exist in the pool of experts). For those tasks we require that they achieve the maximum possible coverage. Our LP has mn variables and $O(mn)$ constraints. If T is the running time for the LP then the overall running time of `LPCover` is $O(T + mn)$. For our experiments we use Gurobi [2] 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. [21]. `TaskGreedy` goes over all tasks sequentially and for each task it greedily assigns experts to maximize the task's coverage. To balance the maximum workload with the total task coverage successfully, we implement two heuristics. First we randomize the order in which experts are greedily assigned to tasks in each iteration. This ensures an even distribution of experts in a setting in which several experts might be equivalently good for a task. Second, we only assign experts if they yield a significant increase in the task coverage. We quantify this coverage amount by a hyperparameter, β, which we specifically grid search and optimize for each dataset. Excluding the grid search, the `TaskGreedy` algorithm has a running time of $O(mn)$ since there are n experts available for each of the m tasks.

`NoUpdateGreedy`**:** This algorithm is a simple modification of the `ThresholdGreedy` algorithm: For each expert-task pair (i, j), we initialize the keys in the priority queue to $v(i, j) = \widetilde{C}((i, j) \mid A^0)$, where A^0 is the assignment with all entries equal to 0. We then use these initial marginal gain values to iteratively add expert-task edges (i, j) from the top of the priority queue to our solution. In order to improve the performance of `NoUpdateGreedy`, we only use an expert if $v(i, j) > \beta$, where β is a hyperparameter. `NoUpdateGreedy` has a running time of $O(mn \log(mn))$, since there are mn total expert-task edges, and sorting these edges for the priority queue takes $O(\log(mn))$ time.

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

6.3 Evaluation In Table 2, we show the comparative performance, in terms of the objective function (F^λ), the average coverage $\widehat{C} = \frac{1}{m}C$ and the maximum load $L_{\max}$, of all four algorithms. 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. Note that for different datasets we use different values of λ; we set these values by following the techniques outlined in Section 5. By searching over the different values of λ for each dataset we pick one 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 details of the experiments related to picking the value of λ for each dataset are discussed in the supplementary material.

Objective values F and workload $L_{\max}$: From Table 2, we observe that `ThresholdGreedy` consistently finds the assignment with the best objective value across all our experiments. On average across all datasets `ThresholdGreedy` performs about 15% better

[2]https://www.gurobi.com/products/gurobi-optimizer/

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	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-15 ($\lambda = 0.05$)	**885**	**7**	**0.99**	777	100	0.97	475	362	0.92	720	150	0.96
IMDB-18 ($\lambda = 0.05$)	**643**	**8**	**0.99**	474	122	0.90	339	264	0.91	448	200	0.98
IMDB-20 ($\lambda = 0.1$)	**771**	**7**	**0.99**	676	87	0.97	644	118	0.97	650	100	0.95
Bbsm-10 ($\lambda = 0.1$)	**2039**	**70**	**0.96**	1691	282	0.90	1319	243	0.71	1097	200	0.59
Bbsm-15 ($\lambda = 0.05$)	**389**	**27**	**0.92**	336	55	0.86	96	126	0.49	129	250	0.84
Bbsm-20 ($\lambda = 1$)	**438**	**41**	0.57	418	84	**0.60**	402	93	0.59	408	94	**0.60**
Freelancer ($\lambda = 0.1$)	**88**	**6**	0.95	59	32	0.92	63	36	**0.99**	25	50	0.76
Guru ($\lambda = 0.1$)	**311**	**4**	**0.99**	287	25	0.98	225	30	0.80	17	33	0.16

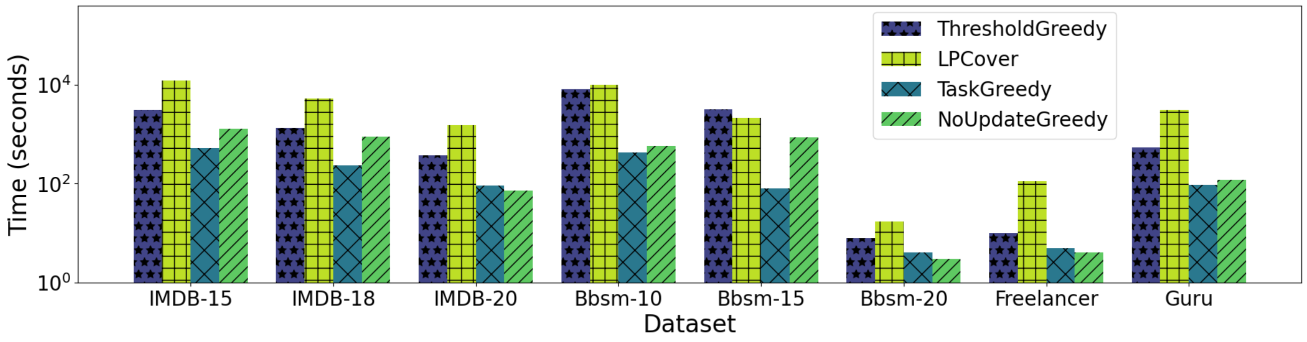

Figure 1: Running times (in seconds) of ThresholdGreedy and baseline algorithms, in logarithmic scale.

than LPCover and 55% better than TaskGreedy and NoUpdateGreedy. As the datasets get larger, the superior performance of ThresholdGreedy becomes more evident. This 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 *Bbsm*-15, *Bbsm*-20 and *Guru* datasets: it returns objective values that are comparable (yet lower) to those returned by ThresholdGreedy. TaskGreedy and NoUpdateGreedy perform relatively well on the *IMDB*-20 and *Bbsm*-20 datasets: they return objective values that are within 20% of the objective value of ThresholdGreedy. This is because the smaller datasets lend themselves better to heuristic choices made by these baselines: the pool of suitable experts available to TaskGreedy is small and the initial marginal gain values used by NoUpdateGreedy are good estimators of the true marginal gain values in subsequent iterations. While the baseline algorithms often achieve a overall task coverage percentage 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. Additionally, 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 baseline algorithms end up making 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 work load values returned by the baselines would be infeasible in practice.

Running times: While ThresholdGreedy has a theoretical running time of $O(m^2n^2)$, the speed-up techniques discussed in Section 4.2 and Section 5 lead to significantly lower running times in practice. Figure 1 shows a bar plot with the running times of all algorithms for each dataset in logscale. For the smaller datasets (e.g. *Freelancer* and *Bbsm*-20), we observe that the running time of ThresholdGreedy is in the order of a few seconds. Even for the largest datasets (e.g., *Bbsm*-2010 and *IMDB*-15) the running time of our algorithm is within a few hours. We observe that TaskGreedy and `NoUpdateGreedy` run faster than our algorithm, but `LPCover` runs slower. This is due to the computational bottleneck of solving an LP for a large number of variables. Note that the running times of the baselines as we report them here do not include the grid search we performed in order to tune their hyperparameters.

7 Conclusions

In this paper, we introduced the BALANCED COVERAGE problem, a new problem in team formation, where the goal is to assign experts to tasks such that the total coverage of the tasks (in terms of their skills) is maximized and the maximum workload of any expert in the assignment is minimized. To the best of our knowledge we are the first to study this problem and study its approximability. We also demonstrated the practical utility of the algorithmic framework we proposed in a variety of real datasets. In the future, we plan to investigate whether other team-formation problems that also take into consideration the social relationship between experts can be studied within the framework we proposed here.

References

[1] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. Power in unity: forming teams in large-scale community systems. In *ACM Conference on Information and Knowledge Management, CIKM*, pages 599–608. ACM, 2010.

[2] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. Online team formation in social networks. In *WWW*, 2012.

[3] A. Anagnostopoulos, C. Castillo, A. Fazzone, S. Leonardi, and E. Terzi. Algorithms for hiring and outsourcing in the online labor market. In Y. Guo and F. Farooq, editors, *ACM SIGKDD*, pages 1109–1118. ACM, 2018.

[4] Y. Azar, A. Z. Broder, and A. R. Karlin. On-line load balancing. *Theor. Comput. Sci.*, 130(1):73–84, 1994.

[5] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. *J. Algorithms*, 18(2):221–237, 1995.

[6] D. Benz, A. Hotho, R. Jäschke, B. Krause, F. Mitzlaff, C. Schmitz, and G. Stumme. The social bookmark and publication management system BibSonomy. *The VLDB Journal*, 19(6):849–875, Dec. 2010.

[7] A. Bhowmik, V. Borkar, D. Garg, and M. Pallan. Submodularity in team formation problem. In *SDM*, 2014.

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

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

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

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

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

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

[14] C.-T. Li, M.-K. Shan, and S.-D. Lin. On team formation with expertise query in collaborative social networks. *KAIS*, 2015.

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

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

[17] A. Majumder, S. Datta, and K. Naidu. Capacitated team formation problem on social networks. In *KDD*, 2012.

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

[19] S. Mitra, M. Feldman, and A. Karbasi. Submodular + concave. *CoRR*, abs/2106.04769, 2021.

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

[21] S. M. Nikolakaki, M. Cai, and E. Terzi. Finding teams that balance expert load and task coverage. *CoRR*, abs/2011.04428, 2020.

[22] S. M. Nikolakaki, A. Ene, and E. Terzi. An efficient framework for balancing submodularity and cost. In F. Zhu, B. C. Ooi, and C. Miao, editors, *ACM SIGKDD*, pages 1256–1266. ACM, 2021.

[23] S. S. Rangapuram, T. Bühler, and M. Hein. Towards realistic team formation in social networks based on densest subgraphs. In *WWW*, 2013.

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

[25] X. Yin, C. Qu, Q. Wang, F. Wu, B. Liu, F. Chen, X. Chen, and D. Fang. Social connection aware team formation for participatory tasks. *IEEE Access*, 2018.

Supplementary material

Proof of Theorem 4.1

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

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

□

Proof of Theorem 4.2 In order to prove Theorem 4.2, we rely on the properties of `ThresholdGreedy` as well as on the fact that the coverage function, $C()$, is monotone and submodular (Proposition 3.1). Recall that A_τ is the assignment produced at the τ-th iteration of `ThresholdGreedy` and $C_\tau = C(A_\tau)$; by definition: $F_\tau = C_\tau - \tau$. Moreover, the monotonicity and submodularity of the coverage function imply the following observations[3]:

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

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

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

Now we are ready to prove Theorem 4.2.

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

$$C_{\tau^*} - \tau^* \geq C_{\tau^*-1} - (\tau^* - 1)$$

$$(C_{\tau^*} - C_{\tau^*-1}) \geq 1. \tag{7.7}$$

Using (7.7) and Observation 2, we have that

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

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

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

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

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

$$C_{\tau^*} - \tau^* \geq C_{\tau^*+1} - (\tau^* + 1)$$

$$(C_{\tau^*+1} - C_{\tau^*}) \leq 1. \tag{7.8}$$

Using (7.8) and Observation 2, we have that

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

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

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

□

Proof of Proposition 5.1

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

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

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

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

Combining these two results we get

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

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

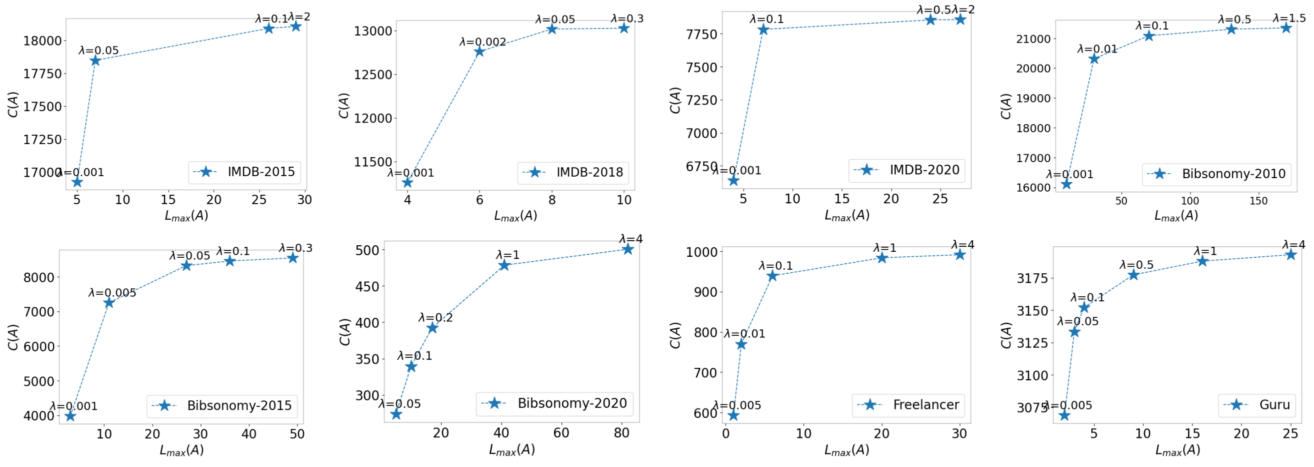

Figure 2: The best-greedy workload $L_{\max}(A)$ value and the coverage $C(A)$ corresponding to the best-greedy objective $F^{\lambda}(A)$ computed by `ThresholdGreedy`. Each subplot shows a range of values of the balancing coefficient λ for each dataset.

Lazy-evaluation Technique The computational bottleneck in `Greedy` is that at every iteration ℓ, the algorithm needs to compute for every expert-task pair (i, j) it's marginal gain $\tilde{C}((i, j) \mid A^{\ell})$ (see Eq. (4.5)). To avoid unnecessary evaluations, we store each pair (i, j) in a maximum priority queue with key $v(i, j)$. We initialize the keys to $v(i, j) = \tilde{C}((i, j) \mid A^0)$, where A^0 is the assignment with all entries equal to 0. As the algorithm runs, the keys in the queue could store potentially outdated marginal gains, but the algorithm only updates them in a lazy fashion. Since $C()$ is a submodular function, marginal gains can only decrease as the number of 1-entries in the formed assignment increases. Therefore, the keys are always an upper bound on the corresponding marginal gains. In each iteration, `Greedy` finds the element in the queue that has the maximum marginal gain as follows: first it removes from the queue the pair (i, j) with the maximum key value and computes its updated marginal gain: $\tilde{C}((i, j) \mid A^{\ell})$. Then, it compares $\tilde{C}((i, j) \mid A^{\ell})$ to the key $v(i', j')$ of pair (i', j') that became the new top of the queue (after removing (i, j)). If $\tilde{C}((i, j) \mid A^{\ell}) > v(i', j')$, then (i, j) is the element with the largest marginal gain. Otherwise, we reinsert (i, j) in the queue with its updated key value being $v(i, j) = \tilde{C}((i, j) \mid A^{\ell})$, and repeat this process with the new maximum key value.

Tuning coverage and workload importance Before showing our experimental results, we discuss how we set the balancing coefficient λ. For this, we use the techniques we described in Section 5. That is, we first run `ThresholdGreedy` with a large value of λ, and determine the *best-greedy workload* and the corresponding value of the *best-greedy objective*. We then compute the corresponding *best-greedy* values for smaller values of λ, and plot the corresponding values of $C(A)$ and $L_{\max}(A)$ for each λ value. Figure 2 shows these scatterplots for each dataset. In most of our datasets we experimented with relatively small values of $\lambda \in (0, 5]$.

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