Power in Unity: Forming Teams in Large-Scale Community Systems [*]

Aris Anagnostopoulos[1]
aris@dis.uniroma1.it

Luca Becchetti[1]
becchett@dis.uniroma1.it

Carlos Castillo[2]
chato@yahoo-inc.com

Aristides Gionis[2]
gionis@yahoo-inc.com

Stefano Leonardi[1]
leon@dis.uniroma1.it

[1]Sapienza University of
Rome, Italy

[2]Yahoo! Research
Barcelona, Spain

ABSTRACT

The internet has enabled the collaboration of groups at a scale that was unseen before. A key problem for large collaboration groups is to be able to allocate tasks effectively. An effective task assignment method should consider both how *fit* teams are for each job as well as how *fair* the assignment is to team members, in terms that no one should be overloaded or unfairly singled out. The assignment has to be done automatically or semi-automatically given that it is difficult and time-consuming to keep track of the skills and the workload of each person. Obviously the method to do this assignment must also be computationally efficient.

In this paper we present a general framework for task-assignment problems. We provide a formal treatment on how to represent teams and tasks. We propose alternative functions for measuring the fitness of a team performing a task and we discuss desirable properties of those functions. Then we focus on one class of task-assignment problems, we characterize the complexity of the problem, and we provide algorithms with provable approximation guarantees, as well as lower bounds. We also present experimental results that show that our methods are useful in practice in several application scenarios.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*; K.4.3 [**Computers and Society**]: Organizational Impacts—*Computer-supported collaborative work*

General Terms

Algorithms, Measurement

Keywords

Team formation, Task assignment, Scheduling

1. INTRODUCTION

While massive collaboration experiments existed before the Internet, in recent years there has been an explosion in the number and the scale of such systems. We have seen the creation of large groups that use electronic communications to coordinate, including large non-governmental bodies and large-scale collaborations such as GNU/Linux and the Wikipedia. On the enterprise, networks of peers with horizontal collaboration links are becoming increasingly common. In many cases, there is no central authority micro-managing the work of the community.

People interact online in many ways: they share information, cooperate, take collective decisions [21], and more. In this paper we aim at providing algorithmic tools for helping people collaborate efficiently. Specifically, we study how to assign tasks in a large groups of peers.

Example. A company has to deal with a continuous stream of ideas about new products or services. Evaluating a proposal requires combining the expertise of several people to create an evaluation team. Each proposal should be properly evaluated, but we must avoid spending too much effort on each one, so evaluation teams should be small. It is also desirable to spread the reviewing effort as evenly as possible, to avoid burdening someone with too many different proposals to evaluate.

This is not an easy problem, not even for a company of medium size. First, if there were no considerations of fairness and all the reviewers had the same cost, then those who are knowledgeable about many technical areas should have to work more than others, as they can provide on their own better coverage of the areas of each proposal. Second, if we want to avoid overloading these reviewers, we have to consider that in some cases we will need more than one person to replace each one of them, in order to cover many of their skills. Intuitively, there is a trade-off between individual work-load and team size.

Contributions. In this paper we study algorithms that, given an incoming stream of tasks, allocate them to teams

[*]The research leading to these results has received funding from the EU FP7 Project N. 255403 – SNAPS, and from the PRIN 2008 research projects COGENT (COmputational and GamE-theoretic aspects of uncoordinated NeTworks) and the Mad Web (Models, Algorithms and Data structures for the Web and other behavioral networks) funded by the Italian Ministry of University and Research.

CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.
Copyright 2010 ACM 978-1-4503-0099-5/10/10 ...$10.00.

of people fit for the task, in a way that is fair: no-one is overloaded or unfairly singled out. Specifically:

- we define a general framework to study task assignment problems under several variants;
- we propose various functions for measuring the fitness of a team performing a task and we discuss desirable properties of those functions;
- we focus on a broad class of task-assignment problems, provide a formal problem definition and characterize their complexity as NP-hard;
- we present a full treatment of an important and realistic scenario with binary profiles in an online setting, providing and analyzing approximate algorithms; and
- we test the performance of these algorithms on three different datasets.

The proposed framework has similarities with the problem of scheduling jobs to machines, as well as with peer-reviewing systems, such as the ones used for allocating papers to reviewers during the review phase of a conference. However, there are key differences. First, team formation entails set-cover, which differentiates our setting from the one studied in scheduling problems. Also, we do not make use of a bidding system and do not model our problem as a matching problem. We elaborate on those differences in the related-work section (Section 6).

Roadmap: Section 2 defines a general framework. Section 3 provides a problem definition and characterizes the complexity of a class of these problems. Section 4 studies in detail approximate methods for the binary case in an online setting, which are tested experimentally in Section 5. Section 6 summarizes prior works related to ours and Section 7 presents our conclusions.

2. FRAMEWORK

The general problem is to assign tasks $\mathbf{J}^j$ from a set $\mathcal{J}$ to teams Q^j, which are subsets of people $\mathcal{P}$, so that teams are *fit* for their tasks and the assignment is *fair* to people. In this section we describe in detail the general framework.

2.1 Tasks, Skills, People, and Teams

Tasks. We have a set of *tasks* (or *jobs*) $\mathcal{J} = \{\mathbf{J}^j; j = 1, 2, \ldots, k\}$, which arrive offline or online and need to be assigned to a team of experts. In the *offline setting* all the tasks are available immediately, while in the *online setting* the jth task arrives at the jth time step and is assigned to a team of experts before the arrival of the $(j+1)$th task.

Skills. Each task requires a set of particular skills to be completed. We define the notion of *skill space* $\mathcal{S}$, which is the space that contains the possible ways that skills combine to form a task, so, each task is a point in the skill space: $\mathbf{J}^j \in \mathcal{S}$. We consider two settings: *binary* and *continuous*. In the binary setting we have m different skills and each job requires a subset of these skills, thus we have $\mathcal{S} = \{0, 1\}^m$. In the continuous setting, we have again m different skills but in this case a job might need each skill up to some extent, so we define $\mathcal{S} = [0, 1]^m$. We use $\mathbf{J}^j_i$ to denote the extent of the ith skill required by the jth task, so we have (in both settings) $\mathbf{J}^j = (\mathbf{J}^j_1, \mathbf{J}^j_2, \ldots, \mathbf{J}^j_m)$.

People. There is a set of *people* (or *experts*) $\mathcal{P} = \{\mathbf{p}^j; j = 1, 2, \ldots, n\}$. People possess a set of skills and their profile is represented by a point in the skill space: $\mathbf{p}^j \in \mathcal{S}$. Similarly with the tasks, we use $\mathbf{p}^j_i$ to denote the extent to which the jth person possesses the ith skill. Thus, in both the binary and the continuous settings we have $\mathbf{p}^j = (\mathbf{p}^j_1, \mathbf{p}^j_2, \ldots, \mathbf{p}^j_m)$.

Table 1: Notation

$\mathcal{S}$	Set of skills
$\mathcal{J}$	Set of tasks
$\mathcal{P}$	Set of people
m	Number of skills
k	Number of tasks
n	Number of people
$\mathbf{J}^j$	jth task
$\mathbf{J}^j_i$	ith skill of jth task
$\mathbf{p}^j$	jth person
$\mathbf{p}^j_i$	ith skill of jth person
Q^j	Team assigned to jth task
$\mathbf{q}^j$	Profile of team j
$\mathbf{q}^j_i$	ith skill of profile of team j
$s(\mathbf{q}, \mathbf{J})$	Performance of team $\mathbf{q}$ on task $\mathbf{J}$
$L(\mathbf{p})$	Load of person $\mathbf{p}$

We assume that the vector of skills that describe each expert is a priori known. In many cases one has to *learn* the skills of the experts from data. In this paper we consider the skill-learning phase to be an independent problem, for which either one of the methods listed in Section 6 can be applied, or new methods can be developed.

Teams. To complete a task we need to assign it to a *team* of experts. We let $Q^j \subseteq \mathcal{P}$ be the team assigned to the jth task. We assume that the skills of each team can be represented by a point in the skill space, so for each team Q^j we compute its *team profile* $\mathbf{q}^j \in \mathcal{S}$. We use $\mathbf{q}^j_i$ to denote the extent of the ith skill possessed by the jth group, so we have $\mathbf{q}^j = (\mathbf{q}^j_1, \mathbf{q}^j_2, \ldots, \mathbf{q}^j_m)$. We refer to a team by using the notation $\mathbf{q}^j$ and Q^j interchangeably. Later, in Section 2.3, we describe various ways to compute the team profile.

Scoring function. Given a team Q with profile $\mathbf{q}$ and a task $\mathbf{J}$ we measure the performance of the team for the task using a *scoring function* $s(\mathbf{q}, \mathbf{J})$. We assume in general that $s(\cdot, \cdot) \in [0, 1]$, with 0 indicating complete failure and 1 complete success, but we may consider special cases of interest. For example, writing $s(\cdot, \cdot) \in \{0, 1\}$, corresponds to tasks that either fail or succeed. In the next section we consider various options for scoring, but for now it could intuitively be seen as (i) a measure of the match between the skills of the team and the requirements of the job, or as (ii) the probability of success in executing the job at a satisfactory level.

Load. Finally, an important quantity is the *load* $L(\mathbf{p})$ of a person $\mathbf{p}$, which is the number of tasks in which she participates: $L(\mathbf{p}) = \left|\{j; \mathbf{p} \in Q^j\}\right|$.

There are two important issues to address at this point: (i) how is the group profile obtained from the individual profiles; and (ii) how the scoring function for a task depends on the task profile and that of the team assigned to it. Before we enter into specifics we note that the setting is fairly general and can capture many different situations, which in turn may require qualitatively different modeling choices.

For instance, consider creating a team for a surgery that requires at least a surgeon, an anesthetist and a nurse. In this case, the surgery can not proceed if the team does not include one person having each of these profiles. On the other hand, consider assembling a team of experts to review a proposal for a new service in a company, as in the example of our introduction. In this case, the review can still be effective, even if the reviewing team does not cover each and every one of the relevant technical areas.

2.2 Properties

We begin by describing some natural properties that we might want a task assignment system to possess, and later we discuss about them.

Property 1 [Nondecreasing performance]. If $Q^j \subseteq Q^i$, then for every task $\mathbf{J}$ we have $s(\mathbf{q}^j, \mathbf{J}) \leq s(\mathbf{q}^i, \mathbf{J})$. This states that a team should perform equally or better than a subset of its members.

Property 2 [Pareto-dominant profiles]. If $\mathbf{q}^j \geq \mathbf{q}^i$ component-wise, then for each task $\mathbf{J}$, we have $s(\mathbf{q}^j, \mathbf{J}) \geq s(\mathbf{q}^i, \mathbf{J})$. In words, a team that is more competent in all skills than another team has better performance at any task.

Property 3 [Job monotonicity]. If $\mathbf{J}^j \leq \mathbf{J}^i$ component-wise, then for every team Q with profile $\mathbf{q}$ we have $s(\mathbf{q}, \mathbf{J}^j) \geq s(\mathbf{q}, \mathbf{J}^i)$. This property states that the same team has a higher performance on a simpler job.

Property 4 [Nonincreasing marginal utility]. Consider two teams Q^j and Q^i such that $Q^j \subseteq Q^i$. For a person $\mathbf{p} \notin Q^i$ let $\hat{\mathbf{q}}^j = F(Q^j \cup \{\mathbf{p}\})$ and $\hat{\mathbf{q}}^i = F(Q^i \cup \{\mathbf{p}\})$. (The function F gives the profile of the team as a function of its members; it is defined and discussed in Section 2.3.) Then, $s(\hat{\mathbf{q}}^j, \mathbf{J}) - s(\mathbf{q}^j, \mathbf{J}) \geq s(\hat{\mathbf{q}}^i, \mathbf{J}) - s(\mathbf{q}^i, \mathbf{J})$.

For each of the four properties stated, one can find situations where they would not hold. Properties 1 and 2 are nevertheless quite natural, and are required in every setting that we consider in the rest of this paper. Properties 3 and 4 are somewhat more restricting.

Property 1 (nondecreasing performance) assumes that new members do not introduce conflicts or large coordination costs – in spite for instance of Fred Brooks' observation in *The Mythical Man-Month* [8] that "adding manpower to a late software project makes it later."

Property 2 (Pareto-dominant profiles) ignores how skills are distributed among the members of a team, although to some extent this can be taken into consideration in the creation of the team profile from the profiles of its members.

Property 3 (job monotonicity), while quite natural for many settings, does not allow for the following situation: Consider a team possessing some set of skills X trying to complete a task requiring a disjoint set of skills Y. Then it should naturally fail. However, if assigned to a more demanding task that requires skills $X \cup Y$ then it might have some partial success. This is related to whether *all* skills are required to complete a task and illustrates how application-specific details can change significantly the nature of the setting and of the problems.

Property 4 (nonincreasing marginal utility) states that the scoring function satisfies a submodularity property, if seen as a function of the team members. This may be the case whenever team members can, to some extent, replace each other: each additional member adds to the team but the marginal increase of every new person might be smaller as the size of the team increases. Marginal utility does not decrease in settings where each skill is unique to a set of people (e.g., in the surgery example described above), or in settings in which a certain minimum number of people having a skill is required for a task to succeed (e.g., when assembling sports teams). A key advantage of settings where Property 4 is satisfied is that various assignment problems can become more tractable by the application of submodular function optimization techniques.

2.3 Team Profiles

As we mentioned previously, we assume that the performance of a team Q can be modeled by a team profile $\mathbf{q} \in \mathcal{S}$. In particular, the profile of a team consisting of $|Q|$ people with profiles $Q = \{\mathbf{p}^1, \mathbf{p}^2, \ldots, \mathbf{p}^{|Q|}\}$ is the profile $\mathbf{q} = F(\{\mathbf{p}^1, \mathbf{p}^2, \ldots, \mathbf{p}^{|Q|}\})$ for a suitable function F.

Some possible choices for the function F include the following:

Maximum skill. Here the expertise of the team for each skill equals to the expertise of the best team member, for that skill:

$$F(Q) = (\max_j \mathbf{p}^j_i, \max_j \mathbf{p}^j_2, \ldots, \max_j \mathbf{p}^j_m) = (\max_j \mathbf{p}^j_i)_{i=1,\ldots,m},$$

where the maxima are taken over the $|Q|$ team members. This might be appropriate in settings where possessing one expert for each skill is of paramount importance.

Additive skills. The expertise of the team is the sum of the skills of each individual. This is appropriate in the case where individuals can complement each other in particular skills, for example, when forming a team of software developers for an application having several modules.

$$F(Q) = (\min\{\sum_{j=1}^{|Q|} \mathbf{p}^j_i, 1\})_{i=1,\ldots,m}$$

Multiplicative skills. Here we set

$$F(Q) = (1 - \prod_{j=1}^{|Q|}(1 - \mathbf{p}^j_i))_{i=1,\ldots,m}.$$

One interpretation for this choice is that possessing a skill is akin to having a certain probability of success, and team members do independent attempts to apply their skills. This might be appropriate in the case of assembling a team of safety inspectors: each one has a certain chance of detecting an issue related to her particular skills, and adding more inspectors decreases the chance of a problem going unnoticed.

Binary profiles for tasks and people. Of particular interest is the special case of binary profiles, where $\mathcal{S} = \{0,1\}^m$, so $\mathbf{p}^j_i \in \{0,1\}$ and $\mathbf{J}^j_i \in \{0,1\}$. Binary profiles arise in many cases of practical interest, in which we are interested in a coarse-grained classification of people's skills (e.g., possessing a license or certification to perform a certain process). The binary-profile model is also interesting because in this case all the choices for skill combination presented above are equivalent and they can be described in terms of the binary `OR` operation:

$$F(Q) = \mathbf{q} = (\bigvee_{i=1,\ldots,k} \mathbf{p}^j_i)_{i=1,\ldots,m}.$$

2.4 Scoring Function

Having described how individuals combine to form a team, we now address the way the team is scored for its performance in accomplishing a task. Of course, depending on the setting, there are different natural scoring-function choices and here we describe some of them.

S1: All skills required. $s(\mathbf{q},\mathbf{J}) = 1$ if $\mathbf{q}_i \geq \mathbf{J}_i$ for all $i = 1, \ldots, m$ and 0 otherwise.

S2: Least-skill dominant. $\min\{1, \min_{i;\mathbf{J}_i>0}\{\mathbf{q}_i/\mathbf{J}_i\}\}$.

S3: Fraction of skills possessed. $s(\mathbf{q},\mathbf{J}) = \frac{|\{i;\mathbf{J}_i>0 \wedge \mathbf{q}_i \geq \mathbf{J}_i\}|}{|\{i;\mathbf{J}_i>0\}|}$.

S4: Microaverage of skill. $s(\mathbf{q},\mathbf{J}) = \frac{\sum_{i;\mathbf{J}_i>0} \min(\mathbf{q}_i,\mathbf{J}_i)/\mathbf{J}_i}{|\{i;\mathbf{J}_i>0\}|}$.

S5: Macroaverage of skill. $s(\mathbf{q},\mathbf{J}) = \frac{\sum_i \min(\mathbf{q}_i,\mathbf{J}_i)}{\sum_i J_i}$.

The first two scoring functions are more natural for cases where a specialist is required for each skill demanded by the task and, not surprisingly, they do not satisfy Property 4 (decreasing marginal utility). The second is a smoother version of the first one. The last three give partial credit for possession of a subset of the skills. In Lemma 1 we show that for any of the rules considered in Section 2.3 they satisfy Property 4.

In the case of a binary skill space one can consider the analogues of the aforementioned functions:

BS1: All skills required. $s(\mathbf{q},\mathbf{J}) = 1$ if $\mathbf{q}_i \geq \mathbf{J}_i$ for all $i = 1, \ldots, m$ and 0 otherwise.

BS2: Fraction of skills possessed. $s(\mathbf{q},\mathbf{J}) = \frac{|\{i;\mathbf{J}_i>0 \wedge \mathbf{q}_i \geq \mathbf{J}_i\}|}{|\{i;\mathbf{J}_i>0\}|}$.

Note that scoring functions S1 and S2 reduce to the BS1 scoring function, while the S3, S4, and S5 reduce to BS2. We now state a lemma showing the submodularity properties of the scoring functions that give partial credits. The proof will appear in the extended version of this paper.

Lemma 1. *Scoring functions S3, S4, S5, and BS2 satisfy the non-increasing-marginal-utility property (Property 4) if the profile of a team is obtained from those of its members according to any of the rules considered in Section 2.3.*

Suppose that we want to maximize the score obtained at each task, subject to constraints on team sizes and the maximum load of each person. For the team profiling methods in Section 2.3 and the scoring functions S3, S4, S5, and BS2 in Section 2.4, this problem can be formulated as the problem of maximizing a submodular function. Details of the formulation will be given in the extended version of the paper.

Lemma 2. *The hill-climbing algorithm [20] for submodular function optimization, adding one by one the person that gives to a team the largest increase in the performance function, provides a $1 - 1/e$ approximation for maximizing the sum of the scores, for scoring functions S3, S4, S5, and BS2.*

In many scenarios, maximizing the scoring may require covering all the necessary skills for each task. This motivates the problem formulation of the next section, that we follow in the remainder of this paper.

3. PROBLEM CHARACTERIZATION

We next focus on a broad class of balanced task assignment problems having *additive skills* in the construction of the team profile (extending the results to the *multiplicative skills* case is direct). We focus on scoring by *requiring all skills*, which can be trivially extended to require only a *fraction of skills*, as indeed we do in the experimental section.

3.1 Balanced Task Covering

We consider the following problem: for every task $\mathbf{J}^j$, find a team Q^j that has all the required skills for $\mathbf{J}^j$, minimizing the workload of the person with the larger number of tasks, namely, $\min \max_j L(\mathbf{p}^j)$.

Set $X_{ji} = 1$ if person $\mathbf{p}^j$ was assigned to task $\mathbf{J}^i$, $X_{ji} = 0$ otherwise. The problem can be formulated as an integer linear program as follows:

$$\min L \tag{1}$$

$$\sum_{j=1}^{n} \mathbf{p}_\ell^j X_{ji} \geq \mathbf{J}_\ell^i, \qquad \forall i = 1, \ldots, k,\ \ell = 1, \ldots, m \tag{2}$$

$$\sum_{i=1}^{k} X_{ji} \leq L, \qquad \forall j = 1, \ldots, n \tag{3}$$

$$L \geq 0, X_{ji} \in \{0, 1\} \tag{4}$$

Constraint (2) ensures that all tasks are executed. Constraint (3) limits the number of teams in which a single person participates. This combinatorial problem is a generalization of well-known load balancing and covering problems. For single-skill jobs we observe an instance of load balancing with restricted assignment [3,4]. 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 the adjacent machines in the bipartite graph. The goal is to find an assignment of jobs to machines that minimizes maximum load.

Our problem is harder than restricted assignment, as already observed in [17], since forming a team requires to solve a set-cover problem: each feasible team is formed by a set of people with profiles that cover all the skills required from a job. In the following we characterize the complexity of the balanced task covering even for relevant special cases and provide offline and online optimization algorithms.

3.2 NP-Hardness

The restricted assignment problem is an NP-hard problem when the number of tasks is large, while it is polynomial-time solvable for constant number of tasks, k. On the contrary, our balanced task covering problem remains NP-hard even when $k = 2$. The proof of the following theorem is omitted due to lack of space and will appear in the extended version of this paper.

Theorem 1. *The balanced task covering problem with 2 tasks is NP-hard.*

The next question that we ask is if some interesting special cases are poly-time solvable. The answer is positive at least for the following cases:

– the problem with constant number of people and tasks;
– the problem with constant number of tasks and skills.

The problem with constant number of people and tasks, $n, k = O(1)$, admits a polynomial-time algorithm. We can enumerate in constant time all feasible and infeasible solutions to the problem given that there exists a constant number of teams for each task and there exists only a constant number of tasks. Every solution can be checked in polynomial time.

The problem with constant number of tasks and skills, $k, m = O(1)$, also admits a polynomial-time algorithm. At most $km = O(1)$ persons are needed to accomplish the k tasks. It is therefore sufficient to check all possible teams of at most km persons, which is a polynomial number.

4. ALGORITHMS

A natural direction for solving the balanced task covering problem is to provide heuristics or approximation algorithms both in the offline and in the online case. In the case of the offline setting, we continue with the class of problems from the previous section. In the case of the online setting, we present a full treatment of the binary case. We also notice that our algorithms can also be adapted to continuous profiles and to tasks of nonuniform load. Details will be provided in the extended version of the paper.

4.1 Offline Setting

We present a simple algorithm to compute a logarithmic approximation for the balanced task covering problem from Section 3.1. Observe that this result is the best possible since the balanced task covering problem contains the set cover problem, which cannot be approximated better than for a logarithmic factor [23]. In the remainder of this subsection, we let $N = \max\{mk, n\}$. The algorithm is inspired by the well-known randomized rounding algorithm for set cover (see, for example, [23, Chapter 14]), based on solving the LP relaxation of the ILP formulation of set cover, and then interpreting the LP solutions as probabilities of including a set in the cover. We are able to prove the following result; details of the algorithm and of the proof for the binary case are given in Appendix A.

THEOREM 2. *There is a polynomial time, randomized algorithm for the balanced task covering problem such that, with probability* $1-\delta$: *(i) every task is assigned to a team possessing all required skills and (ii) the maximum number of tasks assigned to the same person is at most* $O\left(\log \frac{N}{\delta}\right)$ *times the optimum.*

4.2 Online Setting

The online setting is of special practical interest, as it is more realistic to assume that tasks arrive one by one and have to be assigned immediately to a team, without waiting for more tasks to arrive. We number tasks by order of arrival. The algorithm has to decide on a team Q^i for task i in the sequence before task $i+1$ arrives.

For evaluating the performances of the online algorithms we resort to the notion of competitive analysis [7, 15, 22]. Competitive analysis compares an online algorithm against an offline adversary that knows the entire input sequence in advance and serves it optimally. An online algorithm is c-competitive if for any input sequence it provides a solution with cost bounded by c times the optimal cost. Observe that the optimal solution may be hard to compute and that even an online algorithm with unbounded computational resources may not be able to provide an optimal solution given the lack of information about future tasks.

We propose several heuristics and analyze the competitive ratio achieved by these heuristics. All algorithms we present select for a task $\mathbf{J}^i$ the team Q^i that optimizes on a specific cost function $c(Q^i)$ of the current load of all persons $\mathbf{p}^j \in Q^i$ when task $\mathbf{J}^i$ is presented. Denote by $L(\mathbf{p}^j)$ the load of person j when task $\mathbf{J}^i$ is presented, and let Λ be an appropriately chosen value (it depends on the cost of the optimal solution and in the proof of Theorem 3 we show how to estimate it). We consider four algorithms that, whenever a task arrives, find a team Q that covers all the required skills for the current task, and minimizes:

1. `Size`: $c(Q) = |Q|$
2. `MaxLoad`: $c(Q) = \mathtt{max}_{j\in Q} L(\mathbf{p}^j)$
3. `SumLoad`: $c(Q) = \sum_{j\in Q} L(\mathbf{p}^j)$
4. `ExpLoad`: $c(Q) = \sum_{j\in Q} \mu^{\frac{L(\mathbf{p}^j)}{\Lambda}}$ with $\mu = 2n$

`Size`**:** This algorithm picks the team of minimum size among those that have all of the required skills. This corresponds to the solution of an unweighted set cover problem for which we use a standard greedy approximation algorithm that has approximation ratio $O(\log m_i)$ if we denote by m_i the number of skills required by task $\mathbf{J}^i$. Although this heuristic tries to minimize the size of the teams, it does not keep track of the work done so far, and can overload the few experts that possess most of the skills. We show indeed the following lower bound on the competitive ratio of `Size`:

FACT 1. `Size` *has competitive ratio* $\Omega(m)$, $\Omega(n)$, $\Omega(k)$.

PROOF. Consider a sequence of $k = m/2$ tasks. Task $\mathbf{J}^i$, $i = 1, \ldots, k$, requires skills $\{2i-1, 2i\}$. Person $\mathbf{p}^i$ only possesses skill i. Person $\mathbf{p}^{m+1}$ possesses all the skills. `MinSize` assigns all the jobs to person $\mathbf{p}^{m+1}$ that has load k. The optimal assignment involves each person in at most 1 team. □

This heuristic shows that optimizing only on team size may lead to poor results and that in general we should also consider individual workload.

`MaxLoad`: This algorithm requires to solve an unweighted set cover problem on the skills of $\mathbf{J}^i$ restricted to persons of current load at most $\ell = 0, 1, \ldots, \Lambda$. We stop at the minimum value of ℓ for which we obtain a team with members of current load at most ℓ that covers all the skills of $\mathbf{J}^i$. We break ties by picking the team of minimum size $|Q|$ among the ones with current load at most ℓ. This algorithm rules out the lower bound of `Size` since it also optimizes on load. But it might fall in the opposite problem of forming teams of very large size when teams of much smaller size and slightly higher load are possible. This is captured by the following lower bound.

FACT 2. `MaxLoad` *is* $\Omega(k)$, $\Omega(n)$, $\Omega(\sqrt{m})$ *competitive.*

PROOF. Consider a sequence of $k = 2n$ tasks. Tasks $\mathbf{J}^{2i-1}, \mathbf{J}^{2i}$ are identical and they can be processed either by person $\mathbf{p}^i$ alone or by a team formed by all persons with the exception of $\mathbf{p}^i$: $\{\mathbf{p}^j \in \mathcal{P}, j \neq i\}$. Observe that each job requires at least $n-1$ skills. Different jobs require different skills. The total number of skills will therefore be at least $m = k(n-1)$. Job $\mathbf{J}^{2i-1}$ is assigned from `MaxLoad` to person i whose load becomes i. Job $\mathbf{J}^{2i}$ is therefore assigned to team $\{j \in \mathcal{P}, j \neq i\}$ that has all members with load $i-1$. The final load of all persons is $n = k/2$ whereas the optimal load is 2 and it is obtained by assigning both jobs $\mathbf{J}^{2i-1}$ and $\mathbf{J}^{2i}$ to person i. □

The lower bounds presented for `Size` and `MaxLoad` show that a good algorithm for balanced task covering should jointly optimize on team size and on max load. This is captured by the two heuristics we present next, which achieve this tradeoff by weighting every person by his load when computing a team that covers all the skills of a job. SumLoad and ExpLoad require to solve a weighted set cover problem rather than an unweighted set cover problem as in Size and MaxLoad. The greedy algorithm for weighted set cover provides an $O(\log m)$ approximate solution, as in the unweighted case [23].

SumLoad: This algorithm weights each person by his load in the set cover computation performed to cover all the skills of $\mathbf{J}^i$. This strategy may appear reasonable. It is however possible to construct instances where all the jobs are assigned to a single person that possesses all the skills rather than to a larger team formed by persons with small load. This is shown in the following lower bound.

Fact 3. SumLoad *is* $\Omega(k)$, $\Omega(\sqrt{n})$, $\Omega(\sqrt{m})$ *competitive.*

Proof. The first job presented must be processed by the team formed by all people. All persons have load 1. Job i, $i > 1$ can either be processed by person $\mathbf{p}^1$ alone or by a team formed by the i persons in the range $[(i-1)i/2+1, i(i+1)/2]$. The sum of the loads of these persons is larger than the load of $\mathbf{p}^1$ that has all jobs assigned. The optimal solution will instead assign at most 2 jobs to each person. Observe also that the number of persons and the number of activities is quadratic in the number of jobs. □

We conclude from this lower bound that a good online algorithm should be more aggressive in weighting persons with high load.

ExpLoad: This algorithm is more aggressive in penalizing the inclusion of people who are already overloaded, by applying an exponential function to their current load. We study in the following the competitive ratio achieved by this algorithm.

We observe that an $\Omega(\log k)$ lower bound is unavoidable as this is a generalization of the problem of minimizing the maximum load with restricted assignment [4]. A similar lower bound is proved for any deterministic and randomized online algorithm. The idea of the lower bound applied to ExpLoad is as follows. Consider a sequence formed by $O(\log k)$ phases. In the first phase n tasks requiring one single skill owned by all persons are presented. ExpLoad assigns one task per person. In the second phase $n/2$ tasks requiring one single skill owned by $n/2$ persons are presented. In the third phase $n/4$ tasks requiring one single skill owned only by $n/4$ persons are presented. This can be iterated for $\log n$ phases resulting in a maximum load of $\Omega(\log n)$. Observe also $k = n-1$ and $m = \log n$. On the contrary the optimum can assign all tasks with a maximum load of 2.

A matching upper bound can be proved for ExpLoad. However, the additional difficulty of solving a weighted set cover problem results into an additional $O(\log m)$ overhead in the competitive factor. The proof of the following theorem is given in Appendix B.

Theorem 3. ExpLoad *is* $O(\log k \log m)$ *competitive.*

5. EXPERIMENTS

In this section we present experimental results on the performance of the online algorithms on real-world datasets.

Our data sources are IMDB (the Internet Movie Database), Bibsonomy (a social bookmarking site), and Flickr (a photo sharing site). The mapping from them to problem instances in our setting is summarized in Table 2 and explained in the next section. Summary statistics from these datasets are included Table 3.

Table 2: Mapping of data to problem instances

Dataset	Experts	Tasks
IMDB	Movie directors	Audition actors
Bibsonomy	Prolific scientists	Interview scientists
Flickr	Prolific photographers	Judge photos

Table 3: Summary statistics from datasets

Dataset	Experts	Tasks	Skills	Skills/	Skills/
				expert	task
IMDB	725	2 173	21	2.96	11.10
Bibsonomy	816	35 506	793	7.64	4.44
Flickr.art	504	59 869	12 913	49.90	15.73
Flickr.nature	2 879	112 467	26 379	31.25	15.45

5.1 Datasets

IMDB. Our first dataset is the Internet Movie Database. We focus on two types of movie personnel, *directors* and *actors*. The movie genres play the role of skills. We assume that the set of genres of the movies that a person has participated make the set of skills for that person. For example, *Alfred Hitchcock* has the skills {*comedy*, *crime*, *film-noir*, *mystery*, *romance*, *thriller*}, while *Laurence Olivier* has the skills {*biography*, *drama*, *history*, *romance*, *thriller*}. We run our algorithms so that directors represent experts and actors represent tasks.

This setting simulates a scenario in which people who have directed a movie create small committees to audition actors.

Bibsonomy. Our second dataset is a social bookmark and publication sharing system.The dataset contains a large number of computer-science related publications, where each publication is written by a set of *authors*. Publications are not only peer-reviewed articles, but also tutorials, presentations, blog posts, and others. The bibsonomy website is used by a large community of users who employ *tags* to annotate the publications. Tags are pre-processed by applying standard text normalization and frequency filtering operations to remove stopwords. The set of tags associated with the papers of one author is used to represent the set of skills for that author. We partition the set of authors into two sets: one set representing the experts and another set representing the tasks. We consider the experts to be the most prolific authors in the dataset, and this explains why in Table 3 the number of skills per expert is higher than the number of skills per task.

This setting simulates a scenario where committees of scientists interview other scientists for a position at a University or research institution.

As anecdotal aggregates from the Bibsonomy dataset, we mention that the top-10 tags (skills) for all papers are {*information*, *social*, *semantic*, *ontology*, *analysis*, *learning*, *knowledge*, *management*, *software*, *theory*}.

Flickr. We extract multiple datasets from the Flickr photo sharing portal by a kind of snowball sampling. We start with a concept c and use the flickr API to obtain a large number of photos $P(c)$ related with c. This is done by: first, obtaining a large set of photos $P_0(c)$ that contain c as tag;

then, collecting all *groups* $G(c)$ that contain a photo in $P_0(c)$. By groups here we refer to flickr groups, which are pools of photographs centered around one theme. Finally, we collect all photos $P(c)$ in the groups $G(c)$. In this way we are able to collect many relevant photos about a concept c, even if c is not contained directly in the set of tags of those photos. We have performed experiments for 10 concepts obtaining similar results, here we report the concepts `art` and `nature`.

Each photo of the set $P(c)$ has been uploaded by a *user* and is associated with a set of *tags*. We form a set of experts as the top contributing users for the photos in $P(c)$, and the tags of those experts represent their skills. Photos in $P(c)$ (other than the photos of the experts) represent tasks.

This setting simulates a scenario where a committee of photographers judges the quality of a set of photos, for instance, for a photo competition.

The 10 most frequent tags (skills) in the dataset `Flickr.art` are {*art*, *graffiti*, *streetart*, *red*, *painting*, *blue*, *green*, *street*, *sculpture*, *stencil*}, and in `Flickr.nature`: {*nature*, *water*, *flower*, *winter*, *macro*, *snow*, *sky*, *flowers*, *blue*, *green*}.

5.2 Results

We implement the online algorithms described in Section 4.2, which process one task at a time. This involved solving an instance of the set cover algorithm, which is done using the standard greedy algorithm [10]. The greedy algorithm adds experts to a team, one by one, by selecting the one who maximizes a score function.d

The results of our algorithms are shown in Figure 1 and Table 4. We report statistics on the size of the teams and the load of the experts. We note that Figure 1 and Table 4 present the results of the same experiments, the difference is that Figure 1 focuses on the center of the distributions, while the table focuses on the extreme values. In fact, for better visibility, the y-axes in Figure 1 are rescaled, so many outliers are not shown.

The first observation is that the algorithm `Size` makes smaller teams, and thus the average load is the smallest among all algorithms. However, we see that `Size` is very unfair; it uses a small number of individuals for a large number of tasks, which results in loading heavily those individuals. For example, for the `IMDB` dataset, the director whose load is 1260 is `Steven Spielberg`, who is the director with the more genres than any other director; `Steven Spielberg` has 14 while `Robert Zemeckis` and `Tim Burton` follow with 11 and 10, respectively.

In Table 4 we try to measure overload using three measures: the maximum load among all experts, the 90% quantile of the load ($\phi_{.9}$), and the average load of the top 10% of the most loaded users ($\lambda_{.1}$). We see that even though the maximum load of `Size` is much higher than the maximum load of the other algorithms, the $\phi_{.9}$ is lower, suggesting that `Size` loads only a small fraction of the users, and the load drops quickly. However, the top users are heavily overloaded, so $\lambda_{.1}$ is high, as well.

With respect to comparing the other three algorithms `MaxLoad`, `SumLoad`, and `ExpLoad`, we see that they behave very similarly. This is a surprising fact, given that the three algorithms exhibit different behavior in theory. So in our opinion, this is a case of worst case analysis that does not happen in typical data. In fact, we have also tested the algorithms on a different number of synthetic data, and the behavior of the algorithms is still similar. We omit the results on the synthetic data for space limitations.

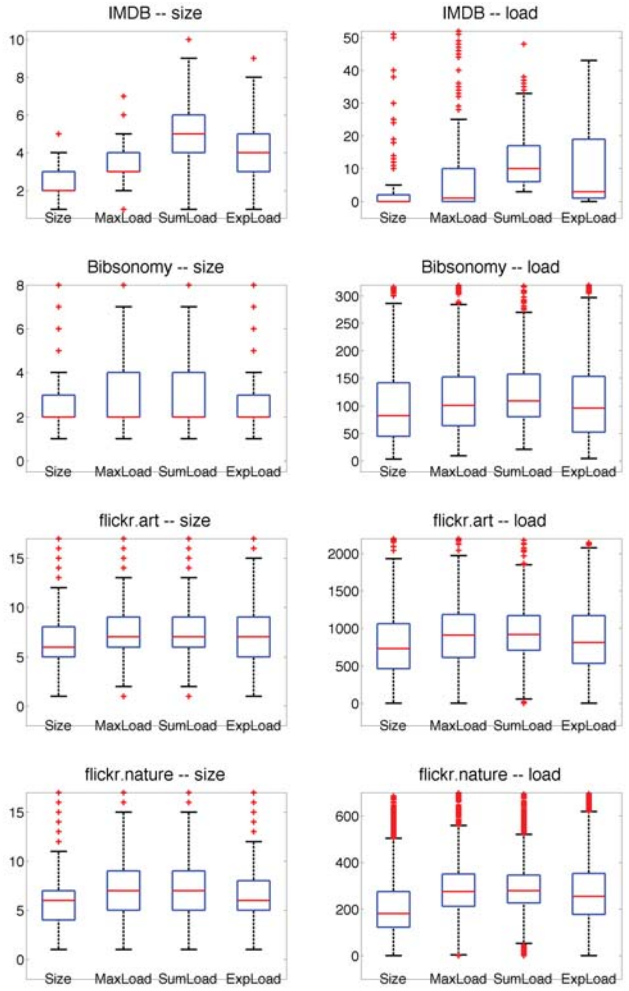

Figure 1: Distributions of team size $|Q|$ (left) and expert load L (right), as provided by our algorithms. Note that the y-axes are rescaled to emphasize the center of the distribution and not all outliers are shown.

Looking a bit closer at the results of the three algorithms `MaxLoad`, `SumLoad`, and `ExpLoad`, we see that `SumLoad` makes teams of sightly larger size, and thus slightly average load, but it performs better according to the $\lambda_{.1}$ measure.

With respect to the distribution of the size of the teams, we see that it is fairly well concentrated around its mean. The maximum team size can sometimes be high, however the 90% quantile is always less than two times the mean. We also report the measure $\sigma_{.9}$, which is the maximum team size so that all tasks are satisfied for at least 90% of their skills. We see that if we are willing to afford such a partial coverage the team size drops significantly.

6. RELATED WORK

Creating teams. Balog et al. [6] state that the profile of experts should not only consider their individual skills, but

Table 4: Statistics on team size and experts load. We report mean, maximum, and additional columns as follows: $\phi_{.9}$ denotes the 90% quantile; $\sigma_{.9}$ is the maximum team size that an algorithm allocates provided that each task is covered only up to 90% of the required skills; finally, $\lambda_{.1}$ is the mean load of the 10% more loaded experts.

	Team size statistics				Experts load statistics			
Method	`mean`	$\phi_{.9}$	$\sigma_{.9}$	`max`	`mean`	$\phi_{.9}$	$\lambda_{.1}$	`max`
				`IMDB`				
`Size`	2.31	4	3	5	6.92	11	58	1260
`MaxLoad`	3.27	4	3	7	9.80	45	53	65
`SumLoad`	4.75	7	3	10	14.23	32	46	65
`ExpLoad`	3.80	5	3	9	11.38	32	47	64
				`Bibsonomy`				
`Size`	2.70	5	5	22	117.66	251	397	1417
`MaxLoad`	2.92	5	3	22	127.13	248	353	700
`SumLoad`	3.13	6	7	25	136.05	244	343	701
`ExpLoad`	2.83	5	4	22	123.27	258	365	700
				`Flickr.art`				
`Size`	7.18	11	6	28	852.56	1473	2337	5631
`MaxLoad`	7.78	12	9	28	924.36	1456	1875	3531
`SumLoad`	8.09	13	16	31	961.18	1433	1792	3539
`ExpLoad`	7.54	12	10	31	896.24	1556	1994	3600
				`Flickr.nature`				
`Size`	6.34	10	25	29	247.85	439	823	6645
`MaxLoad`	7.38	11	27	31	288.22	468	571	941
`SumLoad`	7.53	12	30	35	294.09	438	535	937
`ExpLoad`	7.08	11	28	34	276.60	475	587	964

also a description of their collaboration network. Lappas et al. [17] adopt this view by introducing a weighted social network indicating communication costs, so that close collaborators are joined by edges of small weight. Next, they look for teams that satisfy a task (in the binary sense in our framework) and minimize the communication cost, e.g. in terms of diameter or weight of the minimum spanning tree for the team. Our framework can be extended to this case by incorporating the communication costs. Lappas et al. [17] consider the assignment of a single task, and do not consider the balance of workload when dealing with more than one task.

Scheduling jobs. Scheduling jobs on a set of machines with the goal of minimizing the maximum load on a machine has been a well-studied problem in computer science since the seminal work of Graham [16]. We are not reviewing the vast literature here, but only pointing the reader to the works that are most closely related to our contribution.

As we described in Section 3.1, the problem that is closest to the one considered here is that of restricted assignment of jobs to machines. Here, weighted jobs have to be assigned to machines, but each job can only be processed by a subset of the available machines. The goal is to find an assignment of jobs to machines of minimum maximum load. This problem is NP-hard and it was studied in [3, 4] in the online setting. Here, the authors showed a deterministic online heuristic achieving competitive ratio $O(\log n)$ (n being the number of machines) and they proved that this is asymptotically tight, even for randomized online algorithms.

Our problem is harder than restricted assignment since, as already observed in [17], forming a team entails solving a set-cover problem. This implies a logarithmic lower bound for many objective functions (e.g., `SumLoad`) even in the offline case [23].

Peer reviewing. Refereed conferences and journals require every submission to be reviewed by several members of a program committee. Many such venues receive hundreds of submissions and it is not possible to compute by hand a matching that optimizes fitness and working load. The PC chair matches papers to reviewers based on her knowledge of submissions and PC members, with the help of scores that are computed automatically from keywords provided by PC members and authors. This is a matching problem for which several systems have recently been proposed, for instance Easychair,[1] Linklings,[2] and Softconf.[3] Easychair is currently based on a preliminary bidding process where reviewers rank papers into three classes. The problem of providing efficient solutions in the bidding model has recently been addressed in [19]. In this paper, we have focused on a *covering* problem instead of a *matching* problem, and excluded explicit bids.

Determining expertise. The algorithms we have presented require knowledge about the profiles of skills of the experts. For best performance in a real system, this should be done with a state-of-the-art expert-profiling method.

Early expert-finding methods could be classified basically into methods with a strong Information Retrieval (IR) component or methods with a strong Social Networks (SN) component. The IR approach can be exemplified by Craswell et al. [12] where relevant documents are concatenated to create a "person-document" corpus, over which a standard document search system is run. Balog et al. [5] refined this to incorporate document relevance. The SN approach can be exemplified by Abrol et al. [1] where a graph of documents and people is built and connections are inferred from interaction histories. Other examples use HITS and/or PageRank in the context of online fora [24] or e-mail exchanges [9].

Most current approaches incorporate both IR and SN aspects [13]. Expert finding in general has gained considerable attention in the research community since TREC started to incorporate an expert-finding task [11].

7. CONCLUSIONS

We have described and formally characterized a task assignment problem, and proposed several efficient and effective solutions to solve it. Our results, both theoretical and experimental, show that greedy methods for an online scenario can be effective in practice, as long as they consider both team sizes and workload of members.

Experimentally, we observe that even naïve methods, such as greedily minimizing team size, behave similarly with respect to the average distribution of workload. However, incorporating considerations of workload as the algorithm performs the online assignment of people to tasks, allows to reduce significantly the workload of the most-loaded experts.

Future work. We plan to study other variants of the team-assignment problem, as well as to investigate key application scenarios. We also plan to address the problem of learning the skills of experts, as well as coping with issues such as dynamicity (experts joining and leaving the system) and coordination costs. In general, massive collaboration scenarios

[1] `http://www.easychair.org/`

[2] `http://www.linklings.com/`

[3] `http://www.softconf.com/`

present many algorithmic challenges. One of them is the task assignment problem we have described, but there are many other areas including reputation management, social-network-aware expert finding and document retrieval, and distributed decision making, among others.

Key references: [17].

8. REFERENCES

[1] M. Abrol, U. Mahadevan, K. McCracken, R. Mukherjee, and P. Raghavan. Social networks for enterprise webs. In *Proc. of WWW*, 2002.

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM*, 44(3):486–504, 1997.

[3] Y. Azar. On-line load balancing. In *Theoret. Comp. Sci.* Springer, 1992.

[4] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. In *Proc. of SODA*, 1992.

[5] K. Balog, L. Azzopardi, and M. de Rijke. Formal models for expert finding in enterprise corpora. In *Proc. of SIGIR*, New York, NY, USA, 2006. ACM Press.

[6] K. Balog and M. De Rijke. Determining expert profiles (with an application to expert finding). In *Proc. of IJCAI*, San Francisco, CA, USA, 2007. Morgan Kaufmann.

[7] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[8] F. P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Professional, 1995.

[9] C. S. Campbell, P. P. Maglio, A. Cozzi, and B. Dom. Expertise identification using email communications. In *CIKM*, 2003.

[10] V. Chvatal. A greedy heuristic for the set-covering problem. *Math. of Operations Research*, 4(3), 1979.

[11] N. Craswell, A. P. de Vries, and I. Soboroff, editors. *Proc. of 14th TREC*, volume 500-266 of *Inf. Tech. Lab. NIST Special Pubs.* NIST, 2005.

[12] N. Craswell, D. Hawking, A. marie Vercoustre, and P. Wilkins. P@noptic expert: Searching for experts not just for documents. In *Proc. of AusWeb*, 2001.

[13] H. Deng, I. King, and M. R. Lyu. Formal models for expert finding on DBLP bibliography data. In *ICDM*, 2008.

[14] D. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[15] A. Fiat and G. J. Woeginger, editors. *Online algorithms*, volume 1442 of *LNCS*. Springer, 1998.

[16] R. L. Graham. Bounds on multiprocessing anomalies and related packing algorithms. In *Proc. of AFIPS*. ACM, 1972.

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

[18] S. Leonardi. On-line network routing. In A. Fiat and G. J. Woeginger, editors, *Online algorithms*. Springer, 1998.

[19] K. Mehlhorn. Assigning papers to referees. In *ICALP*, 2009.

[20] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Math. Programming*, (14), 1978.

[21] C. Shirky. *Here Comes Everybody: The Power of Organizing Without Organizations*. Penguin Press, 2008.

[22] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2), 1985.

[23] V. V. Vazirani. *Approximation algorithms*. Springer, 2001.

[24] J. Zhang, M. S. Ackerman, and L. Adamic. Expertise networks in online communities: structure and algorithms. In *Proc. of WWW*, New York, NY, USA, 2007. ACM.

APPENDIX

A. APPROXIMATION OF BALANCED TASK COVERING

Proof of Theorem 2. We give the algorithm and the proof for the binary profiles and we let the continuous $[0,1]$ case for the extended version of the paper.

Recall the ILP formulation presented in Section 3.1. In the case of binary profiles, constraint (2) can be written as

$$\sum_{\substack{j=1\\ j:\ \mathbf{p}^j_\ell=1}}^{n} X_{ji} \geq 1, \qquad \forall i,\ell:\ \mathbf{J}^i_\ell = 1. \tag{5}$$

We solve optimally the corresponding linear programming relaxation, and let $\{\hat{X}_{ji}\}$ be the corresponding fractional solution, so that we have

$$\sum_{\substack{j=1\\ j:\ \mathbf{p}^j_\ell=1}}^{n} \hat{X}_{ji} \geq 1, \qquad \forall i,\ell:\ \mathbf{J}^i_\ell = 1. \tag{6}$$

Let $N = \max\{mk, n\}$. The algorithm performs R rounds (with R to be determined later). In each round, person $\mathbf{p}^j$ is assigned to the task $\mathbf{J}^i$ with probability $\hat{X}_{ji}$, independently of other rounds and of other assignments within the same round. Set $X_{ji}(r) = 1$ if $\mathbf{p}^j$ was assigned to $\mathbf{J}^i$ in the rth round and 0 otherwise, and let $L(\mathbf{p}^j(r))$ be the load on person j in the rth round of assignments. At the end, $X_{ji} = 1$ (i.e., person $\mathbf{p}^j$ is assigned to task $\mathbf{J}^i$) if and only if $X_{ji}(r) = 1$ for at least one round r. Let $\mathbf{L}^*$ be the optimum value. Consider the ℓth skill required by job $\mathbf{J}^i$. The probability that the skill is not covered (i.e., that constraint (5) is not satisfied) in the rth round is at most

$$\prod_{\substack{j=1\\ j:\ \mathbf{p}^j_\ell=1}}^{n} (1 - \hat{X}_{ji}) \leq \prod_j e^{-\hat{X}_{ji}} = e^{-\sum_j \hat{X}_{ji}} \leq \frac{1}{e}, \tag{7}$$

by using Equation (6). From that we obtain that

$$\mathbf{P}\Big(\exists(i,\ell) : \mathbf{J}^j_\ell \text{ not covered}\Big) \leq mke^{-R} \leq Ne^{-R},$$

and note that this quantity is at most $\delta/2$, if we set $R = \ln(2N/\delta)$.

To bound the expected cost of the solution, note first that $\mathbf{P}(X_{ji}=1) = 1-(1-\hat{X}_{ji})^R \leq R\hat{X}_{ji}$.[4] This implies that for all j we have that $\mathbf{E}\big[\sum_i X_{ji}\big] \leq R\mathbf{L}^*$. Since, for every j, the X_{ji}'s are mutually independent, a simple application of Chernoff's bound [14, Equation (1.8)], allows to conclude that $\mathbf{P}\big(L(\mathbf{p}^j) > 2eR\mathbf{L}^*\big) \leq 2^{-2eR\mathbf{L}^*} \leq 4^{-eR}$. We therefore have that

$$\mathbf{P}\Big(\exists j : L(\mathbf{p}^j) > 2eR\mathbf{L}^*\Big) \leq n4^{-eR} \leq N4^{-eR} \leq \frac{\delta}{2},$$

for our choice of R.

For the sake of brevity, we define the events

$$\mathcal{E}_1 = (\exists (i,\ell) : \mathbf{J}^i_\ell \text{ not covered})$$
$$\mathcal{E}_2 = \Big(\exists j : L(\mathbf{p}^j) > 2eR\mathbf{L}^*\Big),$$

and note that event $\mathcal{E}_2$ implies that the cost of the solution L is at most $O\left(\log \frac{N}{\delta}\right)$ the cost of the optimal solution. Then, from what we have shown we can conclude that

$$\mathbf{P}(\mathcal{E}_1 \vee \mathcal{E}_2) \leq \delta. \quad \square$$

B. PROOF OF ALGORITHM EXPLOAD

Proof of Theorem 3. The proof follows the one of Aspnes, et al. [2] for the competitive ratio of online load balancing as adapted in [18]. For the proof of the theorem it is convenient to use a slightly different notation from the rest of the text. Denote by $Q(i)$ and $Q^*(i)$ the team used for job i by the algorithm and by the optimal solution. Let $L_j(i) = |\{i' < i : j \in Q(i')\}|$ and $L^*_j(i) = |\{i' < i : j \in Q^*(i')\}|$ be the online and the optimal load of person p_j when job i is presented. Let $\Lambda = \max_{j=1,\ldots,n} L_j(k+1)$ and $\Lambda^* = \max_{j=1,\ldots,n} L^*_j(k+1)$ be the online and the optimal maximum load on a person at the end of the sequence. Our goal is to minimize the maximum load.

Let $\mu = 2n$ with n being the total number of persons. The algorithm for job i is as follows:

1. Let $c_j(i) = \mu^{\frac{L_j(i)}{\Lambda}}$ be the "exponential" cost of person j when job i is presented. We use $\mu = 2n$.

2. Find the team $Q(i)$ with minimum cost $\sum_{j\in Q(i)} c_j(i)$ that can cover the job.

3. Assign job i to team $Q(i)$

Observe that in reality we cannot find a team $Q(i)$ of minimum cost but only a team of approximately minimum cost by using an approximation algorithm for weighted set cover. Denote by c the approximation ratio of the weighted set cover algorithm used for finding a team of minimum exponential cost that covers all the skills of a job. The standard greedy algorithm achieves an $O(\log m)$ approximation with m total number of skills of a job.

Assume for the moment that the algorithm knows the optimal load Λ^*. We first show that if we assign $\Lambda = 2c\Lambda^* \log \mu$, then the load on every person is at most Λ.

Consider the time at which job i is presented. The cost of any team $Q(i)$ is at most $Z(i) = \sum_{j=1}^n c_j(i)$, the sum of the exponential costs of all the persons when job i is presented. We prove in the following that $Z(i) \leq 2n$, subject to accepting every job $i' < i$ on a team $Q(i')$ with exponential cost that is at most c times the minimum. This immediately implies, $\forall j \in Q(i)$:

$$\mu^{\frac{L_j(i)}{\Lambda}} \leq Z(i) \leq 2n,$$

and then $L_j(i) \leq \Lambda = O(c\Lambda^* \log \mu)$.

We use the following potential function to prove the claim:

$$\Phi(i) \quad = \quad \sum_{j=1}^n c_j(i)\left(1 - \frac{L^*_j(i)}{2\Lambda^*}\right).$$

We have $Z(i) \leq 2\Phi(i) = 2(\Phi(i) - \Phi(0)) + 2n$. $Z(i)$ is then bounded by $2n$ if Φ does not increase when job $i' < i$ is scheduled. The proof is as follows:

$$\begin{aligned}
\Phi(i+1) - \Phi(i) = &\sum_{j\in Q(i)} (\mu^{\frac{L_j(i+1)}{\Lambda}} - \mu^{\frac{L_j(i)}{\Lambda}}) \\
&- \frac{1}{2\Lambda^*} \sum_{j\in Q(i)\cup Q^*(i)} (\mu^{\frac{L_j(i+1)}{\Lambda}} L^*_j(i+1) - \mu^{\frac{L_j(i)}{\Lambda}} L^*_j(i)) \\
\leq &\sum_{j\in Q(i)} (\mu^{\frac{L_j(i+1)}{\Lambda}} - \mu^{\frac{L_j(i)}{\Lambda}}) \\
&- \frac{1}{2\Lambda^*} \sum_{j\in Q^*(i)} (\mu^{\frac{L_j(i+1)}{\Lambda}} L^*_j(i+1) - \mu^{\frac{L_j(i)}{\Lambda}} L^*_j(i)) \\
\leq &\sum_{j\in Q(i)} (\mu^{\frac{L_j(i)+1}{\Lambda}} - \mu^{\frac{L_j(i)}{\Lambda}}) \\
&- \frac{1}{2\Lambda^*} \sum_{j\in Q*(i)} (\mu^{\frac{L_j(i)}{\Lambda}} (L^*_j(i)+1) - \mu^{\frac{L_j(i)}{\Lambda}} L^*_j(i)) \\
= &\sum_{j\in Q(i)} (\mu^{\frac{1}{\Lambda}} - 1)\mu^{\frac{L_j(i)}{\Lambda}} - \frac{1}{2\Lambda^*} \sum_{j\in Q^*(i)} \mu^{\frac{L_j(i)}{\Lambda}} \\
= &\sum_{j\in Q(i)} (2^{\frac{\log \mu}{2c\Lambda^* \log \mu}} - 1)\mu^{\frac{L_j(i)}{\Lambda}} - \frac{1}{2\Lambda^*} \sum_{j\in Q^*(i)} \mu^{\frac{L_j(i)}{\Lambda}} \\
\leq &\frac{1}{2c\Lambda^*} \sum_{j\in Q(i)} \mu^{\frac{L_j(i)}{\Lambda}} - \frac{1}{2\Lambda^*} \sum_{j\in Q^*(i)} \mu^{\frac{L_j(i)}{\Lambda}} \\
\leq &\, 0
\end{aligned}$$

The last two inequalities hold because $2^x - 1 \leq x$ if $0 \leq x \leq 1$, and because $Q(i)$ is c-approximated minimum cost for task i.

To complete the description of the result we are left to remove the assumption that the algorithm knows the optimal load Λ^*. This is usually done [2] by using a doubling technique based on the observation that if the algorithm exceeds load Λ then we do not have a correct estimation of Λ^*, and we can double the current value of Λ^*. This results in a multiplicative factor of 4 in the competitive ratio of the algorithm. $\square$

[4] The inequality follows from the fact that the function $f(x) = (1-x)^R - 1 + Rx$ is nonnegative in $[0,1]$ and thus $(1-x)^R \geq 1 - Rx$.