Profit-maximizing Cluster Hires

Behzad Golshan
Boston University
behzad@cs.bu.edu

Theodoros Lappas
Stevens Institute of Technology
tlappas@stevens.edu

Evimaria Terzi
Boston University
evimaria@cs.bu.edu

ABSTRACT

Team formation has been long recognized as a natural way to acquire a diverse pool of useful skills, by combining experts with complementary talents. This allows organizations to effectively complete beneficial projects from different domains, while also helping individual experts position themselves and succeed in highly competitive job markets. Here, we assume a collection of projects $\mathcal{P}$, where each project requires a certain set of skills, and yields a different benefit upon completion. We are further presented with a pool of experts $\mathcal{X}$, where each expert has his own skillset and compensation demands. Then, we study the problem of hiring a cluster of experts $\mathcal{T} \subseteq \mathcal{X}$, so that the overall compensation (cost) does not exceed a given budget B, and the total benefit of the projects that this team can collectively cover is maximized. We refer to this as the ClusterHire problem. Our work presents a detailed analysis of the computational complexity and hardness of approximation of the problem, as well as heuristic, yet effective, algorithms for solving it in practice. We demonstrate the efficacy of our approaches through experiments on real datasets of experts, and demonstrate their advantage over intuitive baselines. We also explore additional variants of the fundamental problem formulation, in order to account for constraints and considerations that emerge in realistic cluster-hiring scenarios. All variants considered in this paper have immediate applications in the cluster hiring process, as it emerges in the context of different organizational settings.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications-Data Mining

Keywords

Team Formation; Online Marketplaces

KDD'14, August 24–27, 2014, New York, NY, USA.

http://dx.doi.org/10.1145/2623330.2623690.

1. INTRODUCTION

When searching for a group of experts to hire, organizations and decision-makers aim to find the most cost-effective team that can accomplish their goals. The hiring process includes allocating an available budget toward the recruitment of a set of experts from a collection of candidates, in order to form a team that has all the required expertise to perform a large number of profitable projects.

As an example, consider the recruitment of a cluster of college professors by a university. In this setting, the university has to hire individuals that allow it to maximize its academic benefit, given the current opportunities and trends in terms of research and funding options. Given the available budget, the goal is to recruit a cluster of professors that can collectively provide the expertise required to capitalize on such opportunities and maximize the university's benefit. The benefit can be measured in terms of the number of publications or citations, the amount of dollars at grants, or any other function that the university wants to optimize.

In an industry setting, cluster hiring scenarios are even more prevalent. Perhaps the most characteristic example is a typical startup company, where the founders need to select a team of experts with the combined expertise required to capitalize on the different opportunities that have been identified within the market that the company targets.

Another relevant setting comes from the domain of online labor markets, such as oDesk (`www.odesk.com`), Freelancer (`www.freelancer.com`), and Guru (`www.guru.com`). In these online portals, employers hire freelancers with various skills to work remotely on different types of projects. In the early stages of this model, freelancers registered and worked independently. However, as the competition grew, experts realized that it is in their best interest to form "hives", known as agencies, with people of complementary skillsets [11]. This allows them to diversify their talent pool, and go after a larger number of more profitable projects. This trend, has also been recognized by major companies in this area, such as `oDesk.com`, which are already offering team-hiring services to their enterprise customers[1].

In this paper, we formalize the cluster hiring problem as follows: we assume a pool of n experts $\mathcal{X}$, where each expert i is associated with the set of skills X_i that he masters. Additionally, we assume a set of m projects $\mathcal{P}$, where each project $P \in \mathcal{P}$ is also associated with a set of skills; these are the skills that are required for the project to be completed. Finally, every project is associated with a profit $F(P)$ and every expert is associated with a cost $C(X_i)$,

[1] `https://www.odesk.com/info/enterprise/`

which corresponds to X's compensation. Given a budget B, our goal is to form a team of experts $\mathcal{T} \subseteq \mathcal{X}$ such that $\sum_{X \in \mathcal{T}} C(X) \leq B$ and the sum of the profits of the projects that $\mathcal{T}$ can perform is maximized. In the basic version of the problem, we assume that $\mathcal{T}$ can perform a project P only if for every required skill in P, there exists at least one worker in $\mathcal{T}$ that has this skill. We call this problem the ClusterHire problem.

In addition to this fundamental definition, we consider variants of the ClusterHire problem that take into consideration (i) a probabilistic version of project profits, based on how likely it is for the team to actually acquire and complete the project, and (ii) constraints on the usage of the experts with respect to how many projects they are willing and able o participate in simultaneously.

Contributions: To the best of our knowledge we are the first to define and study the ClusterHire problem and its variants. More specifically, we show that this problem is NP-hard to solve and even approximate. We then proceed to design effective heuristics and demonstrate their efficacy in practical settings. For our experiments we use data from Freelancer and Guru. These are two of the largest players in the rapidly growing area of *online labor*, which has been identified as a clear candidate for the cluster hire setting [11]. In addition to experimentally evaluating our algorithms and investigating the characteristics of their solutions, we also provide an extensive data-analytic and visual study of these datasets that provides a much deeper understanding of the nature and dynamics of expert communities.

Roadmap: The rest of the paper is organized as follows: we review the related work in Section 2. In Section 3, we define the ClusterHire problem and study its computational complexity. Our algorithms for the problem are described in Section 4, and in Section 5 we provide a thorough analysis of our datasets. An extensive experimental evaluation of our methods is given in Section 6. We conclude the paper in Section 7.

2. RELATED WORK

To the best of our knowledge we are the first to introduce and study the ClusterHire problem and its variants. However, our work is clearly related to other recently-studied team formation problems as well as literature on studying team dynamics and predicting the performance of teams, as well as work on inferring the abilities of the team members based on their team performance. We give an overview of this body of related work below.

Our previous work [14] was the first to introduce the problem of team formation in the context of social networks. Given a pool of experts and a set of skills that needed to be covered, the goal was to select a team of experts that can collectively cover all required skills, while ensuring efficient communication between the team's members. Over the last years, this work has been extended in multiple directions. For example, there exists recent work that focuses on incorporating different definitions of communication costs between experts [1, 6, 8, 13, 15, 19]. Others take into consideration different levels of users' abilities and capacity to participate in different projects [18]. Finally, more recently, the online version of the problem was introduced where the goal is to create multiple teams that can address multiple projects that arrive in an online fashion [3]. The common characteristic of all the above works is that they assume a network of experts and therefore, all these formulations have a graph-theoretic component. Our work does not assume the existence of a graph and therefore our setting raises different computational questions than the ones addressed in the above papers.

Probably the most related to ours is the recent work by Anagnostopoulos et al. [2]. This paper considers a pool of experts, with each expert associated with a set of skills, and a collection of projects arriving one at a time in an online fashion. Each project is characterized by the set of skills required for its completion. In the version of their problem that is most related to ours, Anagnostopoulos et al. aim to create one team for each project, such that over time, the maximum number of teams that each expert participates in is minimized. There are two significant differences between their setting and ours. First, our goal is to create a single team that can address many projects, while the goal of Anagnostopoulos et. al. is to choose a single team per project. Second, we do not assume that projects arrive online: in our setting the set of projects is known apriori. Therefore, again, the computational issues we face and address here are distinct.

In the area of online gaming and robot team formation, the problem of inferring the performance of a teams of players has been investigated at length [17, 16]. Most recently Liemhetcharat and Velose [17] explore a version of the problem in which each expert has a non-deterministic capability with respect to each project. The expected performance of the team is then evaluated based on the synergy of the participating experts, rather than their coverage of skills. Other relevant works [5, 7] evaluate team performance from an anthropological/sociological perspective. We consider these complementary to our computational approach.

Recently Gionis et al. [10] have proposed a combinatorial method for computing the skillset of individual experts, based on their participation in successful teams. Their problem setting can be thought as inverse to ours, since our goal is to compute teams based on the skillsets of experts.

3. PROBLEM DEFINITIONS

In this section, we provide the notation we will use throughout the paper, and present the formal definitions and complexity results for different versions of our problem.

3.1 Preliminaries

Throughout the discussion we will assume that there is a set of ℓ skills S, a set of n experts $\mathcal{X}$ and a set of m projects $\mathcal{P}$. Each expert $X \in \mathcal{X}$, is represented by a subset of skills, i.e., $X \subseteq S$; these are the skills that the expert possesses. Similarly, every project $P \in \mathcal{P}$ is characterized by the set of skills that are required in order for the project to be completed (i.e., $P \subseteq S$).

In addition to the above, we have a *profit function* (F), such that for every $P \in \mathcal{P}$, $F(P)$ gives the (expected) profit that completing P with incur. Similarly, for every expert X, function $C(X)$ gives the cost of hiring a particular expert.

For a team of experts $\mathcal{T} \subseteq \mathcal{X}$, we say that team $\mathcal{T}$ has a skill s if there exist at least one expert $X \in \mathcal{T}$, such that X has skill s, i.e., $s \in X$. For a project $P \in \mathcal{P}$, we say that team $\mathcal{T}$ *covers* P if $\mathcal{T}$ (as a team) has all the skills required for P. Clearly, a team of experts may cover more than one projects; in fact the more such projects being covered by

a single team the better the team. That is, we define the *coverage* of team $\mathcal{T}$ to be the set of distinct projects that the team can cover. That is,

$$\text{Cov}(\mathcal{T}) = \{P \in \mathcal{P} \mid \mathcal{T} \text{ covers } P\}. \quad (1)$$

Given the projects that a team $\mathcal{T}$ can cover, we define the profit of the team to be the summation of the profits of the projects that $\mathcal{T}$ covers. That is,

$$F(\text{Cov}(\mathcal{T})) = \sum_{P \in \text{Cov}(\mathcal{T})} F(P). \quad (2)$$

In addition, every team incurs a certain cost, computed as the sum of the costs of its members. That is,

$$C(\mathcal{T}) = \sum_{X \in \mathcal{T}} C(X). \quad (3)$$

Dollar profit: In real applications, there is typically a prespecified gain (in terms of dollars) that the completion of a project will yield for the team (or for the organization that has hired the team). This dollar amount can thus naturally serve as the profit of that project, which we refer to as F_d.

Competition-based profit: The above dollar-based assignment of profits to projects does not consider the uncertainty that is involved in the process of trying to acquire and complete a project. Consider the following example: we are presented with two projects P_1 and P_2 that are worth the same dollar amount. However, assume that P_1 requires a set of "mundane" skills, i.e., skills that are very popular among the pool experts, while P_2 requires a rare skill s^*. Clearly, there is larger competition for project P_1, since there are many possible teams that can contribute the required skills. On the other hand, the competition for P_2 is smaller due to the rare skill that it requires. Therefore, a team that has s^* in its combined skillset has a higher probability of being assigned project P_2, if it chooses to pursue it.

This competitive setting is simply one of the alternative instantiations of uncertainty, which is present in all the applications we consider. Alternatively, one could consider the probability of completing a project even after it has been secured, given the deadlines and other constraints placed by the employer. Our framework is compatible with any method that computes the probability of success for each project. For our own experiments, we compute the *dollar-based* profit of a project P as follows:

$$F_c(P) = \frac{1}{\mathit{freq}(s^*(P))} F_d(P),$$

where $s^*(P)$ is the rarest skill among the skills required for P and $\mathit{freq}(s^*(P))$ is the number of experts that actually possess this skill. While we experimented with other alternatives, including the median and average frequency of the skills required by a project, we found that using the minimum frequency yielded the most intuitive results.

Throughout the paper, we use the generic notation (F) to refer to projects' profits and only use F_d and F_c when we need to explicitly compare them.

3.2 The ClusterHire problem

Given the above notation, we can now define the main problem addressed in this paper as follows:

Problem 1 (ClusterHire). *Given a set of projects $\mathcal{P}$, a pool of experts $\mathcal{X}$, cost and profit functions $C()$ and $F()$, respectively, and a budget $B \in \mathbb{R}^+$ find $\mathcal{T} \subseteq \mathcal{X}$ such that $F(\text{Cov}(\mathcal{T}))$ is maximized and $C(\mathcal{T}) \leq B$.*

From the computational point of view, we have the following results for the ClusterHire problem.

Lemma 1. *The decision version of the* ClusterHire *problem is NP-complete.*

Proof. For our proof, we will consider a simplified version of the problem where $\mathcal{P}$ consists of a single project P and $F(P) = 1$. Moreover, $C(X) = 1$ for every expert $X \in \mathcal{X}$. In the decision version of this simplified instance of the ClusterHire problem the question is if there exist a team of K experts that covers project P.

Now, we will reduce the decision version of the SetCover problem [9] to this simplified version of the ClusterHire problem. In the classical SetCover problem there is a universe of items U and a set of sets $\mathcal{Q}$ such that for every $Q \in \mathcal{Q}$, $Q \subseteq U$. The question in the decision version of the problem is whether there exist L sets from $\mathcal{Q}$, forming $\mathcal{Q}'$ such that $\cup_{Q \in \mathcal{Q}'} Q_i = U$.

Clearly, if we map every set $Q_i \in \mathcal{Q}$ from SetCover into an expert $X_i \in \mathcal{X}$ of ClusterHire, the two problems become identical. That is, there exists a solution of size L in the SetCover problem if and only if there exists a solution of cost L in the ClusterHire problem. □

Lemma 2. *The* ClusterHire *problem is NP-hard to approximate.*

Proof. The proof of the above lemma uses the same simplified decision version of the ClusterHire problem used in the proof of Lemma 1, as well as some intuition we gained from that proof.

More specifically, we will prove the lemma by contradiction. That is, assume that there exists an α-factor approximation algorithm for this simplified version of the ClusterHire problem such that if $\mathcal{T}^A$ is the solution of this algorithm and $\mathcal{T}^*$ is the optimal solution, we have that: $F(\mathcal{T}^A) \geq \alpha F(\mathcal{T}^*)$.

Now observe that such an algorithm can be used to decide whether there exists a solution consisting of K experts that could perform the project P of the ClusterHire problem. However, as we showed above, the decision version of this problem is NP-complete. Therefore, such an α-approximation algorithm cannot exist. □

The t-ClusterHire problem: Observe that the ClusterHire problem as defined in Problem 1 allows for solutions where a single expert $X \in \mathcal{T}$ can use a particular skill in multiple projects in $\text{Cov}(\mathcal{T})$. In practice, however, this setting can lead to an expert being overworked, especially if he is one of the few members (or even the sole member) in the team who possesses a frequently required skill.

In order to avoid such shortcomings, we propose a variant of ClusterHire that places an upper bound $t(X, s)$ on the number of projects for which an expert X can utilize a skill s. By allowing for a different threshold for each expert-skill combination, this formulation is a natural fit for scenarios where certain skills are much easier to apply than others, especially for specific experts. For example, while it is very difficult for a software engineer to be the "development leader" for more than one project, the same person can use his "software consulting" expertise to assist in numerous ongoing efforts. In the same consulting setting, it

is easier for a more experienced engineer to participate in more projects.

Computing $\text{Cov}_t(\mathcal{T})$: In practice, such a threshold corresponds to a different definition of the coverage of projects by teams. We refer to this alternative definition as t-coverage and we define it below. If use $\text{Cov}_t(\mathcal{T})$ to denote the set projects that are t-covered by team $\mathcal{T}$, then this set has the following characteristic:

$$F(\text{Cov}_t(\mathcal{T})) \text{ is maximized} \tag{4}$$

such that for all $s \in S$,

$$|\{P \in \text{Cov}_t(\mathcal{T}) | s \in \mathcal{P}\}| \leq \sum_{X \in \mathcal{T}} \sum_{s \in X} t(X, s). \tag{5}$$

The constraints encoded by the inequalities in Equation (5) are essentially the threshold constraints imposed by the experts.

We observe that given a team $\mathcal{T}$, computing $\text{Cov}_t(\mathcal{T})$ as described by Equations 4 and 5 is also an NP-hard problem. In fact, in the special case where $t(X, s) = 1$ for all skills s and experts X, then the problem described by the above equation is identical to the SetPacking problem [9].

4. ALGORITHMS

In this section, we describe our algorithms for the basic version of the ClusterHire problem. We then demonstrate how to adapt these algorithms to solve t-ClusterHire.

The `ExpertGreedy` algorithm: The `ExpertGreedy` algorithm is a greedy algorithm on the space of experts. That is, it greedily picks experts — one at a time — so that the budget constraint is satisfied while at the same time the F objective is maximized. Specifically, the algorithm starts with an empty team and at iteration i, it forms $\mathcal{T}^i$. The expert X, picked at iteration $(i+1)$ needs to be among the qualified candidates $\mathcal{Q}$, where:

$$\mathcal{Q} = \{X \mid X \notin \mathcal{T}^i \text{ and } C(\mathcal{T}^i \cup \{X\}) \leq B\},$$

while X maximizes:

$$\frac{F(\text{Cov}(\mathcal{T}^i \cup \{X\})) - F(\text{Cov}(\mathcal{T}^i))}{C(X)}.$$

Note that that the set of qualified candidates $\mathcal{Q}$ is different at every iteration. The set consists of all the remaining experts that can be added to the current solution without violating the budget constraint B. In case of a tie between candidate experts, the algorithm picks an expert at random.

Algorithm 1 The `ExpertGreedy` algorithm.

Input: Experts $\mathcal{X}$, projects $\mathcal{P}$, budget B.
Output: $\mathcal{T} \subseteq \mathcal{X}$

1: $\mathcal{T} = \emptyset$, $b = 0$, $\mathcal{Q} = \mathcal{X}$
2: **while** $b < B$ and $\mathcal{Q} \neq \emptyset$ **do**
3: $\quad \mathcal{Q} = \{X \mid \notin \mathcal{T} \text{ and } C(\mathcal{T} \cup \{X\}) \leq B\}$
4: $\quad X = \arg\max_{X' \in \mathcal{Q}} \frac{F(\text{Cov}(\mathcal{T} \cup \{X'\})) - F(\text{Cov}(\mathcal{T}))}{C(X')}$
5: $\quad \mathcal{T} = \mathcal{T} \cup \{X\}$
6: $\quad b = b + C(X)$
7: return $\mathcal{T}$

The pseudocode of `ExpertGreedy` is shown in Algorithm 1. The worst-case running time of each iteration of `ExpertGreedy` is $O(nm\ell)$. However, careful implementation and bookkeeping that takes into consideration the sparsity of the data allow for much better running times in practice.

The `ProjectGreedy` algorithm: In contrast to `ExpertGreedy`, which picks experts greedily, `ProjectGreedy` operates by greedily selecting the projects to be covered – one a a time – and then finding the best set of experts that can cover the selected project.

Algorithm 2 The `ProjectGreedy` algorithm.

Input: Experts $\mathcal{X}$, projects $\mathcal{P}$, budget B.
Output: $\mathcal{T} \subseteq \mathcal{X}$

1: $\mathcal{T} = \emptyset$, $b = 0$
2: **while** $b < B$ and $\mathcal{P} \neq \emptyset$ **do**
3: $\quad \mathcal{P}$: set of projects *not covered* by $\mathcal{T}$
4: $\quad$ **for** $P \in \mathcal{P}$ **do**
5: $\quad\quad \mathcal{X}_P$: experts from $\mathcal{X}$ required to cover P
6: $\quad\quad$ **if** $C(\mathcal{X}_P) + b \geq B$ **then**
7: $\quad\quad\quad \mathcal{P} = \mathcal{P} \setminus \{P\}$
8: $\quad P = \arg\max_{P \in \mathcal{P}} \frac{F(\text{Cov}(\mathcal{T}^i \cup \mathcal{X}_P)) - F(\text{Cov}(\mathcal{T}^i))}{C(\mathcal{X}_P)}$
9: $\quad \mathcal{T} = \mathcal{T} \cup \mathcal{X}_P$
10: $\quad b = b + C(\mathcal{X}_P)$
11: return $\mathcal{T}$

Assume that, at iteration i, `ProjectGreedy` has formed a team $\mathcal{T}^i$. Then, at iteration $(i+1)$, the algorithm picks a project P that is not covered by the skills in team $\mathcal{T}^i$. The selection of P is done so that, if $\mathcal{X}_P$ is the set of additional experts required to cover P, the ratio

$$\frac{F(\text{Cov}(\mathcal{T}^i \cup \mathcal{X}_P)) - F(\text{Cov}(\mathcal{T}^i))}{C(\mathcal{X}_P)}$$

is maximized and the budget constraint, i.e., $C(\mathcal{T}^i \cup \mathcal{X}_P) \leq B$ is satisfied. Similar to `ExpertGreedy`, any ties are broken arbitrarily.

The pseudocode of `ProjectGreedy` is shown in Algorithm 2. We draw attention to line 5 of this pseudocode. This step finds a subset of experts, $\mathcal{X}_P$, who together with the current members of $\mathcal{T}$ can collectively cover the skills of project P. Clearly, the formation of $\mathcal{X}_P$ needs to be budget-efficient. Thus, this step involves solving an instance of the *weighted set-cover* problem for the skills of P which are not currently covered by $\mathcal{T}$. For this, we use the standard greedy approximation algorithm for weighted set cover. Thus, a different set-cover problem needs to be solved for each one of the candidate projects.

If I is the number of iterations of the outer `while` loop and T_G the running time required for finding $\mathcal{X}_P$ (i.e., the running time of the greedy algorithm for weighted-set cover on the space of experts), the worst-case running time of `ProjectGreedy` is $O(ImT_G)$. Next, we introduce a method that reduces the number of candidate projects m and consequently leads, in practice, to smaller running times.

The `CliqueGreedy` algorithm: By greedily selecting a single project at every iteration, the `ProjectGreedy` algorithm is forced to repeatedly solve an instance of the weighted set cover problem. At the same time, by evaluating the profit-cost ratio of each project independently, the algorithm fails to identify sets of projects that require similar or even near-identical skillsets, which could lead to even higher ratios if selected together.

Motivated by these observations, we design the `CliqueGreedy` algorithm. `CliqueGreedy` can be thought of as an

extension of `ProjectGreedy`, which operates on *groups* of projects, rather than individual projects. Essentially, a group can be viewed as a larger project that requires the union of the skills required by the projects in the group. Next, we describe a 2-step method for grouping projects.

In the first step, we consider the grouping benefit for each pair of projects. Given two project P_1 and P_2, we consider them *compatible* if the profit-to-cost ratio for both projects increases by at least a factor of $(1+\alpha)$ if they are merged. Formally, if

$$R_i = \frac{F(P_i)}{C(P_i)},$$

for $i = 1, 2$, and

$$R = \frac{F(P_1 \cup P_2)}{C(P_1 \cup P_2)} \tag{6}$$

then P_1 and P_2 are in the same group if the following *compatibility condition* holds:

$$R > (1+\alpha)R_i \text{ for both } i = 1, 2. \tag{7}$$

In the second step, we compute the maximal cliques in the graph that has a node for every project and edges between every two compatible projects. The computed cliques then serve as the groups that are considered by `CliqueGreedy`.

Note that a clique is maximal if it is not included in any other possible clique. This allows to avoid trivial candidates and limit the number of cliques that need to be considered. In practice, even the enumeration of all maximal cliques is possible. In our experiments, we use the efficient implementation of the algorithm proposed by Bron and Kerbosch [4] that is included in the NetworkX library [12].

As we saw in our experiments, the computational time of the grouping phase is dominated by the first step, since checking the compatibility condition for a pair of projects requires solving yet another set-cover problem. Therefore, checking the compatibility between all $\binom{m}{2}$ pairs of projects requires solving as many set-cover problems. In order to address this, we effectively eliminate a large number of such pair-wise checks by making the following observations:

OBSERVATION 1. *Any two projects P_1 and P_2 are not compatible if $P_1 \cap P_2 = \emptyset$.*

OBSERVATION 2. *Any two projects P_1 and P_2 are not compatible if*

$$\frac{F(P_i)}{C(P_i)} \leq (1+\alpha)\frac{F(P_1)+F(P_2)}{\max\{C(P_1), C(P_2)\}}.$$

Both these observations are direct consequences of the compatibility condition and can be evaluated in constant time given that $C(P_i)$ is computed for every project – which is required anyway even in the simple version of `ProjectGreedy`. In practice, we have observed that these two pruning mechanisms are extremely effective, as they promptly dismiss many project pairs as incompatible.

Another factor that affects the running time of the grouping phase is the value of the parameter α. This has a direct effect on the applicability of the second pruning criterion, as well as on the density (and thus the clique computation) of the resulting project graph. Larger values of α lead to sparser graphs and reduce the number of cliques.

In our experiments, we found that the value of α is dataset dependent, but tuning this parameter is easy if the dataset characteristics are studied appropriately.

Algorithms for T-CLUSTERHIRE**:** All three algorithms we designed for T-CLUSTERHIRE, i.e., `ExpertGreedy`, `ProjectGreedy` and `CliqueGreedy` can be modified to take into consideration the threshold on the number of times every expert can use each skill, as stipulated by the definition of the T-CLUSTERHIRE problem. In all cases, the greedy heuristic is modified to maximize the marginal gain at the profit level, while satisfying the threshold constraints. Considering the pseudocode for `ExpertGreedy` given by Algorithm 1, the only required modification is to alter line 4 so that it computes the t-coverage of the teams, and thus use COV_t instead of simple COV. Finally, the `ProjectGreedy` for this version is similarly obtained by modifying line 8 of Algorithm 2 to compute the t-coverages instead of the simple coverages.

5. EXPERTISE DATASETS

In our experiments, we used data collected from two large online labor markets: `guru.com` and `freelancer.com`. We refer to these datasets as *Guru* and *Freelancer* respectively.

The business model for labor markets: Both websites follow the same business model: employer post a description of a project that needs to be completed, including the required skills and the monetary reward that they are willing to pay. Experts with various skillsets and salary demands apply for each project, and are evaluated by the employer.

Data analytics: From each website, we collect the following artifacts: (i) the set of skills and the salary demands (in dollars per hour) for each expert, and (ii) the set of required skills and the monetary reward (in dollars) for each posted project. For the *Guru* dataset, we collected data on 6 473 experts and 1 764 projects. For the *Freelancer* dataset, we collected data on 1 763 experts and 721 projects.

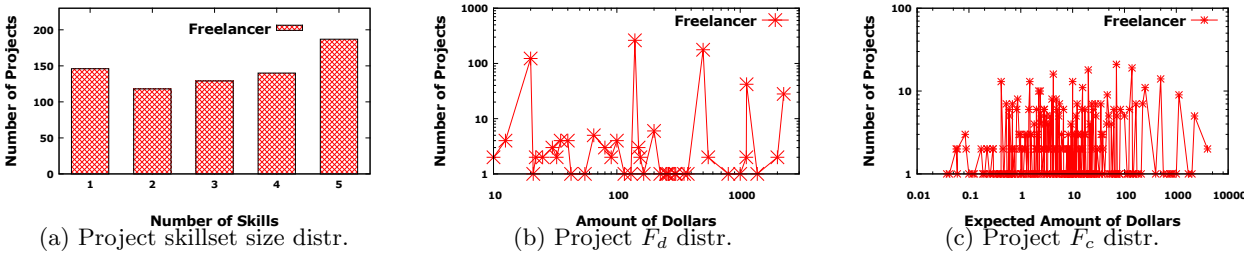
Project analytics: Figures 1 and 2 provide some descriptive analytics on the projects from *Freelancer* and *Guru* datasets, respectively.

Figures 1(a) and 2(a) display the distribution of the size of the skillset required for the projects in *Freelancer* and *Guru* respectively. From Figure 2(a), we observe that while the majority of projects in *Guru* require up to 10 skills, larger projects of 30 skills or more are also posted. For *Freelancer* the distribution is different. This is due to the fact in `freelancer.com` employers are only allowed to specify at most 5 skills per project; this fact clearly manifests itself in Figure 1(a).

Figures 1(b) and 2(b) show the distribution of project profits for *Freelancer* and *Guru*, respectively, under the F_d profit function (amount of dollars). Similarly, figures 1(c) and 2(c) show the distribution of project profits under the F_c (expected amount of dollars, based on competition). The x-axis holds the profits, sorted from lowest to highest, and the y-axis the number of projects associated with each profit value. For both figures, both axes are in a logarithmic scale.

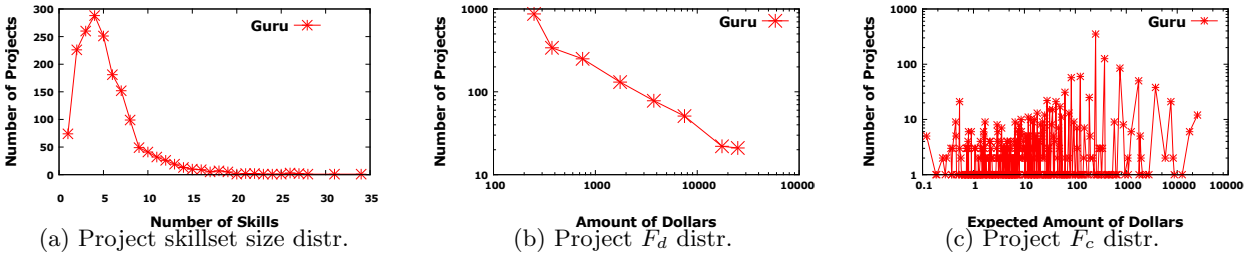
For the F_d scheme, we observe only 9 distinct profit values for *Guru* (due to the quantization of profits made by the website), which also follow a distribution that resembles a power law (given the almost straight line and the log-log scale). On the other hand, for *Freelancer*, we observe a much higher variance in the distribution of profits.

As anticipated, the F_c scheme introduces much larger variance in the profit distribution for both datasets, with no clearly identifiable distribution shape. This is reasonable,



(a) Project skillset size distr. (b) Project F_d distr. (c) Project F_c distr.

Figure 1: Project analytics for the *Freelancer* dataset.



(a) Project skillset size distr. (b) Project F_d distr. (c) Project F_c distr.

Figure 2: Project analytics for the *Guru* dataset.

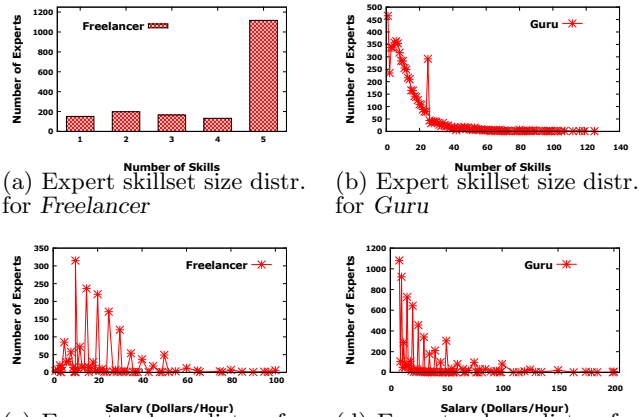
since this function is based on the rarity of the required skills. which can vary a lot more across projects.

Expert analytics: Figure 3 shows different analytics for the experts from *Freelancer* and *Guru*. Figures 3(a) and 3(b) show the distribution of the skillset size (i.e., number of skills) of the experts in each dataset: the x-axis corresponds to the size of skillset and the y-axis to the number of experts that have skillsets of that size. For *Guru*, Figure 3(b), this distribution is a power-law distribution, with most users having less than 20 skills. On the other hand, the majority of experts on *Freelancer* have five skills, while the remaining skillsets are almost uniformly distributed over 1,2,3, and 4 skills. This difference is explained by the skill verification mechanism that is in place by `freelancer.com` where an expert can declare any number of skills, however on each expert's only at most 6 most verified skills are displayed; a skill of an expert gets a verification every time the expert participates in a project that utilizes this skill. Such a mechanism is absent from `guru.com` (at least at the time the data was collected), and thus the distribution of skillset sizes is different.

Finally, Figures 3(c) and 3(d) show the distribution of salaries for *Freelancer* and *Guru* experts respectively. The x-axis holds the salaries, sorted from lowest to highest, and the y-axis the number of experts with a given salary. Both figures display a "power law"-like distribution, with the majority of experts asking for an hourly salary of at most 50.

Visualization: In an attempt to gain a deeper understanding of the two datasets in the context of our problem, we display the *similarity graphs* for experts in Figures 4 and 6.

Expert graph: In the graphs in Figure 4, nodes correspond to experts. There is an edge between two experts if their Jaccard similarity, computed on their skillsets, is higher than 0.7. For the graph we removed all nodes with



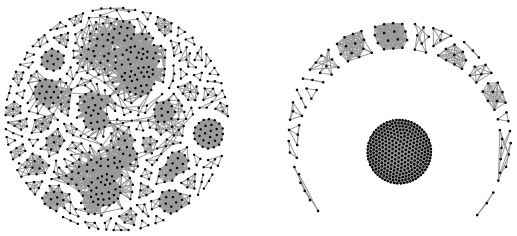
(a) Expert skillset size distr. for *Freelancer* (b) Expert skillset size distr. for *Guru*

(c) Expert salary distr. for *Freelancer* (d) Expert salary distr. for *Guru*

Figure 3: Expert analytics for *Freelancer* and *Guru* datasets.

degree less than 2.[2] The figure demonstrates that the latent similarity structure among experts differs dramatically in the two datasets For *Freelancer* (Figure 4(a)) we observe a number of dense neighborhoods of different sizes, representing clusters of similar experts. At the same time, the graph is well-connected, with several edges often bridging the observed neighborhoods. On the other hand, for *Guru*, (Figure 4(b)), we observe a single large dense component, as well as several smaller components that are not connected to each other. This difference stems from the different na-

[2]Experimenting with lower and higher values of the thresholds had the expected results of producing larger and smaller cliques, respectively, for both datasets.



(a) *Freelancer* (b) *Guru*

Figure 4: Expert graphs: Similarity graphs for *Guru* and *Freelancer* experts.

ture of the two websites; `guru.com` hosts a diverse spectrum of experts with skillsets in different domains ranging from IT to legal services, financial consulting etc. On the other hand, the majority of the experts in `freelancer.com` are technology-oriented professionals who are more likely to have overlapping skillsets.

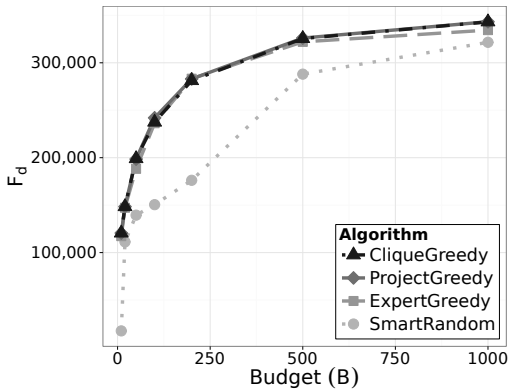
Project graphs: In the graphs in Figure 6, nodes correspond to projects. The graphs were constructed as follows: there is an edge between two projects if the compatibility condition described in Equation (7) is satisfied. We visualize the compatibility graphs for both the F functions: F_d (the dollar amount attached to each project) and F_c (the expected dollar amount, given the competition a team has to face in the process of acquiring the project). Figures 6(a) and 6(b) display the graphs for both functions for the *Freelancer* dataset for $\alpha = 0.2$. Figures 6(c) and 6(d) display the same graphs for *Guru* and $\alpha = 0.7$.

Comparing the graphs across datasets, we observe the same trends found in the expert graphs: *Freelancer* includes a number of distinct (but still connected) neighborhoods. On the other hand, the *Guru* graphs are dominated by 1-2 large components which correspond to experts from disciplines with more dominant representation on the website.

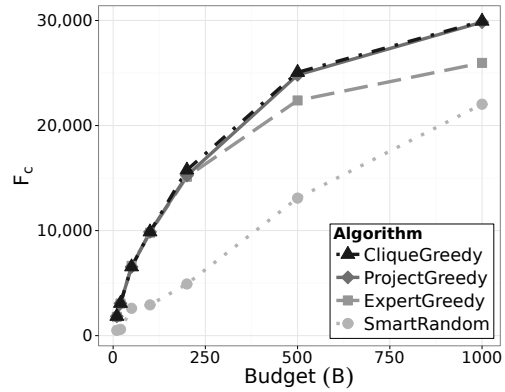
The figures also reveal some interesting facts when considered in the contexts of the two different profit functions. For *Guru*, introducing the competition factor in the profitability of a project leads to a sparser graph. As we saw in Figures 2(a) and 1(a), the average project on *Guru* requires a significantly higher number of skills than the average *Freelancer* project. This makes the occurrence of rare skills more likely, which has a direct effect on the competitive-driven F_c function: as these competitive projects emerge, they are less likely to improve their profit-to-cost ratio by being grouped with other projects, leading to less edges in the graph. On the other hand, the effects of competition on the *Freelancer* project-graph are less prevalent, which is likely to be due to the lower diversity of skillsets required by each project (limited to at most 5).

6. EXPERIMENTS

In our experiments we study the relative performance of our algorithms for the *Freelancer* and the *Guru* datasets, which we extensively analyzed in the previous section.



(a) Dollar profit (F_d)



(b) Competition-based profit (F_c)

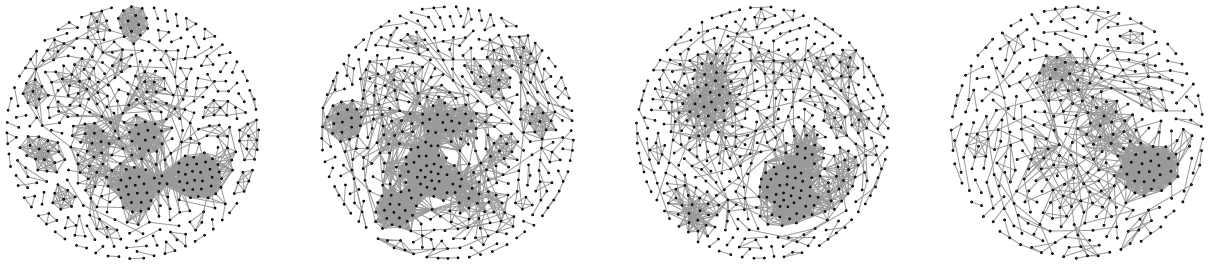
Figure 5: Performance of different algorithms for the ClusterHire problem (*Freelancer* dataset).

6.1 Evaluating algorithms for ClusterHire

First, we focus on the evaluation of our algorithms for the ClusterHire problem, i.e., `ExpertGreedy`, `ProjectGreedy` and `CliqueGreedy`. To do so, we report the overall profit achieved by each algorithm, for increasing values of the available budget, i.e., $B \in \{10, 20, 50, 100, 200, 500, 1000\}$.

The `SmartRandom` baseline: As an additional baseline, we also evaluate an iterative randomized algorithm, which we refer to as `SmartRandom`. `SmartRandom` is essentially a random version of `ProjectGreedy` and at each iteration it selects a random project, and then proceeds to hire the cheapest set of experts who can fully cover this project. This set of experts is again identified by the greedy algorithm for set cover. In order to ensure that `SmartRandom` exhausts the available budget, only projects that can be covered using the current budget are considered on every iteration. The algorithm then terminates when no such projects exist. Although we use `SmartRandom` as our baseline, it really makes much more educated guesses than a naive random algorithm that forms random teams of experts.

Results for the *Freelancer* dataset: Figures 5(a) and 5(b) show the performance of different algorithms on the *Freelancer* dataset, in terms of the dollar (F_d) and competition-based (F_c) profit functions respectively. The y-axis shows



(a) (F_d); *Freelancer* (b) (F_c); *Freelancer* (c) (F_d); *Guru* (d) (F_c); *Guru*

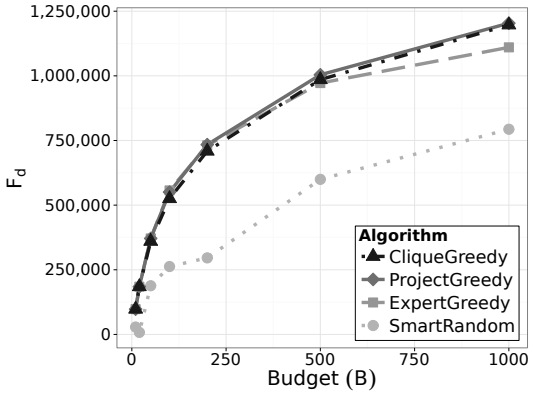
Figure 6: Project graphs: Compatibility graphs for *Guru* and *Freelancer* projects.

the profit achieved by each algorithm, and the x-axis holds the budget that was used to hire experts.

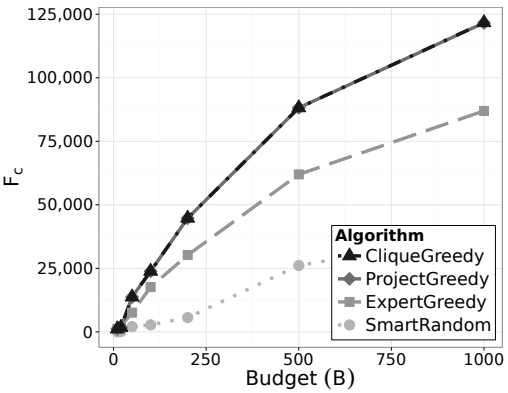
Starting with the results for the dollar profit model, shown in Figure 5(a), we observe that, given a budget of 500$, every algorithm except `SmartRandom` achieves a profit around $32K$$, while `SmartRandom` reaches a value of $29K$$. The fact that all algorithms perform well implies that the dataset contains many profitable projects and many low-cost experts who can accomplish these projects. We also observe that `SmartRandom` performs similar to other algorithms under a limited budget (i.e., $0 \leq B \leq 200$), which is reasonable since all algorithms are restricted to a limited set of projects that can actually be covered given the budget.

Under the competition-based profit model the results are different, as illustrated by Figure 5(b). What we observe in this case is that both `ProjectGreedy` and `CliqueGreedy` start to diverge and outperform other methods for budgets above 200$. To understand why `ExpertGreedy` is not as efficient as before, one has to remember that many projects with high dollar profit values will no longer be profitable once the competition of the market is taken into consideration in the evaluation of the profit. The effect of this function is that it changes the space of projects so that there are less projects that are both profitable and cost-effective. This gives `ProjectGreedy` and `CliqueGreedy` an advantage over `ExpertGreedy`. This is because the former evaluate and search for *profitable* projects (or groups of projects) as opposed to `ExpertGreedy` that searches for individual workers who alone can only do projects with small profits.

Results for the *Guru* dataset: To check the consistency of our results, we repeated the same experiment for both profit functions for the *Guru* dataset. The results are shown in Figure 7. Similar to our observations for *Freelancer* dataset, we observe that, while all algorithms perform well in the dollar-based setting, `ExpertGreedy` falls short once the notion of competition is introduced. In fact, the gap between `ExpertGreedy` and the two algorithms based on project selection, i.e., `ProjectGreedy` and `CliqueGreedy`, is even greater than the one observed for *Freelancer*. A similar trend can be observed for `SmartRandom`: the algorithm is again consistently outperformed for both profit functions, with the difference in profit being significantly greater for this dataset, especially under the competition-based profit scheme. As re-



(a) Dollar profit (F_d)



(b) Competition-based profit (F_c)

Figure 7: Performance of different algorithms for ClusterHire problem (*Guru* dataset).

vealed in the project analytics presented in Figures 1 and 2, the competitive-based profit function results in a much higher variance in the profits of the available projects. This has a negative effect on `SmartRandom` and `ExpertGreedy`, which do not take into consideration the profitability of the projects when they make their selections. This negative effect is even stronger for the *Guru* dataset, since the competitive profit function dramatically increases the number of less profitable projects. In fact, as can be seen by Figures 2(b) and 2(c), the F_c function introduces very large number of projects with a profit less than 100$, while no such projects existed for the simple dollar F_d function.

The characteristics of the solutions: In order to gain a deeper understanding of the results, we compute the average profit of the projects covered by the teams reported by the different algorithms for fixed budget $B = 500$\$. The results are summarized in Table 1. The main message of this table is that the average profit of the projects that can be covered by the solutions of `ProjectGreedy` and `CliqueGreedy` is higher than the corresponding average profit achieved by `ExpertGreedy` only in the case where profit is computed by the competition-based profit function. This observation is true for both our datasets. As we have already explained, the reason for that is that `ProjectGreedy` and `CliqueGreedy` are able to identify profitable projects and pick them even if these projects cannot be performed by a single worker. On the other hand, even in the presence of competition, `ExpertGreedy` cannot ignore projects that can be performed by single workers, which are also projects that typically require common skills.

Table 1: Average profit of covered projects for $B = 500$\$

	Freelancer		*Guru*	
	F_d	F_c	F_d	F_c
`ExpertGreedy`	569.0	40.3	1528.3	110.4
`ProjectGreedy`	594.9	47.7	1701.9	186.0
`CliqueGreedy`	594.0	47.5	1667.3	186.0

Performance of `CliqueGreedy`: Figures 5 and 7 show that the `CliqueGreedy` algorithm performs only slightly better in terms of profit, when compared to `ProjectGreedy`. Therefore, a natural question to ask is whether there are any benefits that this algorithm has to offer. Our answer to this question is the following: although `CliqueGreedy` offers smalls gains in terms of profit, there are datasets for which it offers significant computational speedups. We provide evidence for this statement in Table 2.

The table reports the number of candidates that need to be evaluated by `CliqueGreedy` and `ProjectGreedy` for both datasets. The number of candidates per iteration that an algorithm has to evaluate is an important measure of its running time. This is because for every candidate project (or group of projects) the algorithm picks, it needs to solve a set cover problem for this candidate.

The results in Table 2 show that `CliqueGreedy` consistently evaluates less candidates during its computation, especially for the *Freelancer* dataset. This essentially means that for this dataset about 1/3 of the candidates are eliminated and thus 1/3 less set cover computations need to be made by `CliqueGreedy`. Therefore, from the computational point of view for the *Freelancer* `CliqueGreedy` is beneficial

Table 2: Number of candidate (groups) of projects.

	Freelancer		*Guru*	
	F_d	F_c	F_d	F_c
`ProjectGreedy`	721	721	1764	1764
`CliqueGreedy`	520	570	1732	1660

since it offers a significant speedup while giving solutions with (approximately) the same profit. On the other hand, for the *Guru* dataset `CliqueGreedy` does not appear to offer significant computational savings. All these results were computed for $\alpha = 0.2$ for *Freelancer* and $\alpha = 0.7$ for *Guru*.

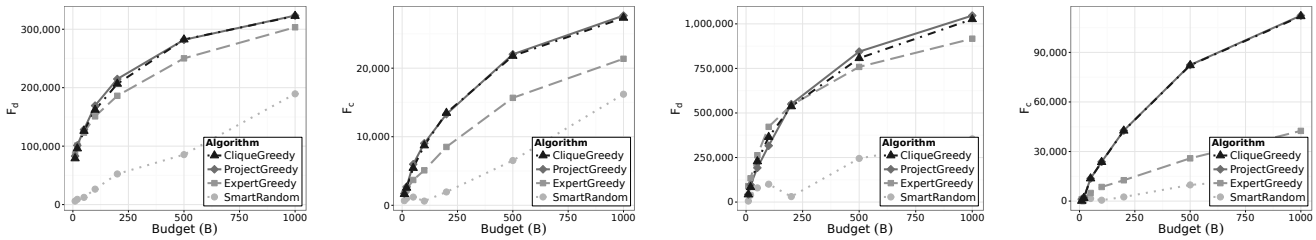
A visual analysis of the available projects in a dataset, such as the one we presented in Figure 6, can be used prior to running the algorithms, to evaluate if the underlying graph structures justifies the use of `CliqueGreedy`. For example, if multiple dense neighborhoods (which are likely to include cliques) can be identified, then the algorithm can indeed lead to significant computational savings. This is indeed the case for the *Freelancer* dataset and thus the savings.

6.2 Evaluating algorithms for T-CLUSTERHIRE

In this section, we evaluate the performance of for our methods for the T-CLUSTERHIRE problem. For this we use the same experimental setup and evaluation methodology as in the previous section. For T-CLUSTERHIRE, we need to set the value of the threshold on the number of times that a user is willing to utilize each one of his skills. We set this value to $t = 3$ for all users and all skills. However, our experiments suggest that despite the fact that the actual profits change for different thresholds, the relative performance of algorithms is independent of this threshold.

Figure 8 shows the profit achieved by the different algorithms for both datasets and for both our profit models, i.e., dollar profit and competition-based profit. Most of the observations we made in the previous section are true here as well. More specifically, Figures 8(a) and 8(c) show how different algorithms perform under the dollar-based profit model for *Freelancer* and *Guru* datasets respectively. Similar to our previous results for CLUSTERHIRE , we can see that all algorithms perform significantly better than `SmartRandom`. However, unlike our previous results, we can see that the performance of `ExpertGreedy` is significantly lower than the performance of both `ProjectGreedy` and `CliqueGreedy` algorithms. This is due to the utilization constraint which limits the number of cost-effective projects. That is, a set of projects with many overlapping skills are profitable in the CLUSTERHIRE problem since few cheap experts can cover all of the at once, while in the CLUSTERHIRE the profit one can get from this overlapping projects is bounded due to the utility constraint. Therefore, `ExpertGreedy`, that essentially prefers cheap experts looses its power because these experts cannot be used over and over again for multiple projects.

Figures 8(b) and 8(d) show how different algorithms perform under the competition-based profit model. As discussed in the previous section, adjusting the profits based on the competition in the market leads to significantly smaller number of cost-effective jobs. This gives both `ProjectGreedy` and `CliqueGreedy` an advantage as they effectively search for good projects. On the other hand, `ExpertGreedy` does not perform well because due to its myopic nature selects cheap experts that can only do "trivial" projects.



(a) F_d (*Freelancer*) (b) F_c (*Freelancer*) (c) F_d (*Guru*) (d) F_c (*Guru*)

Figure 8: Performance of different algorithms for the T-CLUSTERHIRE problem (*Guru* and *Freelancer* datasets)

Overall, for the T-CLUSTERHIRE problem `ProjectGreedy` and `CliqueGreedy` are consistently and significantly better than `ExpertGreedy`. Despite the computational speedups achieved by `CliqueGreedy` for this problem, the profit it obtained in this case is slightly less than the profit achieved by `ProjectGreedy`. The reason for the slight degradation of the profit achieved by `CliqueGreedy` can be explained as follows: once projects are grouped into clusters, they need to be picked together. However, given the utilization constraints, the set of workers that can satisfy a group of projects may end up being expensive and hence less profitable. In this case, one has to find the balance between computational efficiency and profit that one wants to achieve.

7. CONCLUSIONS

In this paper, we proposed formalizations and algorithmic solutions for the CLUSTERHIRE problem, where the goal is to hire a profit-maximizing team of experts with the ability to complete multiple projects, given a fixed budget. This problem repeatedly emerges in organizational settings, and it has become prevalent due to the establishment of the collaboration paradigm in online labor markets. We provided a detailed analysis of the computational complexity and hardness of approximation of the problem, and presented algorithms that take into consideration the unique nature of expertise data. Our methodology was evaluated on data from two of large players in the domain of online labor.

8. ACKNOWLEDGMENTS

This research was supported by: NSF CAREER #1253393, NSF grants: CNS #1017529, III #1218437, IIS #1320542 and gifts from Microsoft and Hariri Institute of Computing.

9. REFERENCES

[1] A. An, M. Kargar, and M. ZiHayat. Finding affordable and collaborative teams from a network of experts. In *SDM*, pages 587–595, 2013.

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

[3] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. Online team formation in social networks. In *WWW*, pages 839–848, 2012.

[4] C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, 1973.

[5] S.-J. Chen and L. Lin. Modeling team member characteristics for the formation of a multifunctional team in concurrent engineering. *IEEE Transactions on Engineering Management*, 2004.

[6] C. Dorn and S. Dustdar. Composing near-optimal expert teams: A trade-off between skills and connectivity. In *OTM Conferences (1)*, pages 472–489, 2010.

[7] E. L. Fitzpatrick and R. G. Askin. Forming effective worker teams with multi-functional skill requirements. *Ind. Eng.*, 2005.

[8] A. Gajewar and A. D. Sarma. Multi-skill collaborative teams based on densest subgraphs. In *SDM*, pages 165–176, 2012.

[9] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[10] A. Gionis, T. Lappas, and E. Terzi. Estimating entity importance via counting set covers. In *KDD*, pages 687–695, 2012.

[11] R. Greenwald. Freelancers find it pays to team up. *The Wall Street Journal*, 2014.

[12] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, 2008.

[13] M. Kargar and A. An. Discovering top-k teams of experts with/without a leader in social networks. In *CIKM*, pages 985–994, 2011.

[14] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *ACM SIGKDD*, pages 467–476, 2009.

[15] C.-T. Li and M.-K. Shan. Team formation for generalized tasks in expertise social networks. In *SocialCom/PASSAT*, pages 9–16, 2010.

[16] S. Liemhetcharat and M. Veloso. Forming an effective multi-robot team robust to failures. In *IROS*, 2013.

[17] S. Liemhetcharat and M. Veloso. Weighted synergy graphs for effective team formation with heterogeneous ad hoc agents. *Artif. Intell.*, 208:41–65, 2014.

[18] A. Majumder, S. Datta, and K. V. M. Naidu. Capacitated team formation problem on social networks. In *KDD*, pages 1005–1013, 2012.

[19] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *ACM SIGKDD*, pages 939–948, 2010.