PLOS ONE

RESEARCH ARTICLE

Cost-minimizing team hires with participation constraint

Heli Sun[1], Jianbin Huang[2]*, Ke Liu[2], Mengjie Wan[2], Yu Zhou[2], Chen Cao[1], Xiaolin Jia[1], Liang He[1]

1 Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, Shaanxi, China, **2** School of Software, Xidian University, Xi'an, Shaanxi, China

* jbhuang@xidian.edu.cn

Citation: Sun H, Huang J, Liu K, Wan M, Zhou Y, Cao C, et al. (2018) Cost-minimizing team hires with participation constraint. PLoS ONE 13(8): e0201596. https://doi.org/10.1371/journal.pone.0201596

Editor: Kim-Kwang Raymond Choo, University of Texas at San Antonio, UNITED STATES

Received: November 15, 2016

Accepted: July 18, 2018

Published: August 28, 2018

Data Availability Statement: All relevant data are within the paper and its Supporting Information files.

Funding: The author(s) received no specific funding for this work.

Competing interests: The authors have declared that no competing interests exist.

Abstract

Team formation, which aims to form a team to complete a given task by covering its required skills, furnishes a natural way to help organizers complete projects effectively. In this work, we propose a new team hiring problem. Given a set of projects $\mathcal{P}$ with required skills, and a pool of experts $\mathcal{X}$, each of which has his own skillset, compensation demand and participation constraint (i.e., the maximum number of projects the expert can participate in simultaneously), we seek to hire a team of participation-constrained experts $\mathcal{T} \subseteq \mathcal{X}$ to complete all the projects so that the overall compensation is minimized. We refer to this as the participation constrained team hire problem. To the best of our knowledge, this is the first work to investigate the problem. We also study a special case of the problem, where the number of projects is within the participation constraint of each expert and design an exact algorithm for it. Since participation constrained team hire problem is proven to be NP-hard, we design three novel efficient approximate algorithms as its solution, each of which focuses on a particular perspective of the problem. We perform extensive experimental studies, on both synthetic and real datasets, to evaluate the performance of our algorithms. Experimental results show that our exact algorithm far surpasses the brute-force solutions and works well in practice. Besides, the three algorithms behave differently when distinct facets of the problem are involved.

Introduction

A successful recruitment process or project bidding process devotes to hiring a set of experts from a batch of candidates that satisfy the requirements of specific projects. From the perspective of managers how to form a cost-efficient team to accomplish specific projects is one of the most essential issue. In most cases, the specific properties of experts, such as professional expertise [1], work time, the maximum workload [2] and leader evaluation and team cohesiveness [3] vary among different individuals. Such factors may affect the amount of projects they want to engage in simultaneously. Therefore, an efficient hiring process should take both the cost and the ability of experts into consideration.

Assume such a scenario, a software company wants to build a team of engineers to develop a number of mobile applications, supplying programmers, system architects, product managers, UI designers and technical advisers separately for each application. In this setting, creating a skilled and cost-effective team for the projects is desired for managers. We can easily come up with a solution that each position just needs to find one appropriate expert to guarantee the completion of it. However, it seems unreasonable from the perspective of experts that no distinction is made between the numbers of projects each member can enter. Due to the heavy workload inherent in coding, programmers are more inclined to engage in only one project, while technical advisers may take an active part in multiple projects at the same time depending on entailed by work. Thus, the number of projects each expert participates in, i.e., the participation constraint [4], varies among different individuals. Effective team hiring process should take this trait into consideration whose significance has been proven in many real-world scenarios.

However, team hiring is not limited to the domain of software development only. In the setting of film industry, social-bookmarking, academic cooperation and crowdsourcing, teams are fundamental to these collaborative scenarios, where both the cost of a team and the ability of experts are essential and should be considered [2, 5, 6]. In addition, there are some other works related to team hiring problem such as top-k team formation [7], analytical team formation [8] and social event organization [9–12] and so on. Overall, the problem of hiring a team of experts for collaborative projects has extensive real-world applications and is an important problem to study.

We illustrate aforementioned characteristics more concretely through the following simple example. We assume there is a manager who wants to build a team of experts to perform the following projects: $\mathcal{P} = \{1, 2, 3, 4\}$, with required skills shown in Table 1. Also assume there are eight experts, $\mathcal{X} = \{A, B, C, D, E, F, G, H\}$, equipped with the skills, cost and participation constraint denoted by p-constraint listed in Table 2.

Without considering the participation constraint of each expert, the manager can select either $X = \{A, E, F, G\}$, $X' = \{A, C, D, E, F, H\}$ or $X'' = \{C, D, F, G, H\}$, since all these teams can collectively cover the required skills of the projects. Fig 1 depicts the assignment scheme without participation constraint, where apparently expert F joins four projects in parallel while expert G enters merely one project. Then, after imposing the participation constraint on project assignment, the resulting assignment schemes are shown in Figs 2 and 3. Furthermore, the comparison of the total cost that 21 incurred by X'' is less than 27 by X' suggests that $X'' = \{C, D, F, G, H\}$ is a superior solution.

Motivated by the above observation, in this paper, we formalize the problem inspired by [13] in the following. Assume a pool of n experts $\mathcal{X}$, where each expert $x_i \in \mathcal{X}$ possesses a set

Table 1. Skills of projects.

projectid	1	2	3	4
skills	{a,b,c,d,e}	{a,c,d,e,f}	{b,c,f,g}	{a,d,e,g}

https://doi.org/10.1371/journal.pone.0201596.t001

Table 2. Skills, cost and participation constraint of experts.

expertid	A	B	C	D	E	F	G	H
skills	{a,b}	{d,e}	{b,c,d}	{a,f,g}	{c,f}	{d,e,g}	{b,c,f}	{a,b,e}
cost	5	10	3	6	4	5	3	4
p-constraint	2	4	2	3	2	3	2	2

https://doi.org/10.1371/journal.pone.0201596.t002

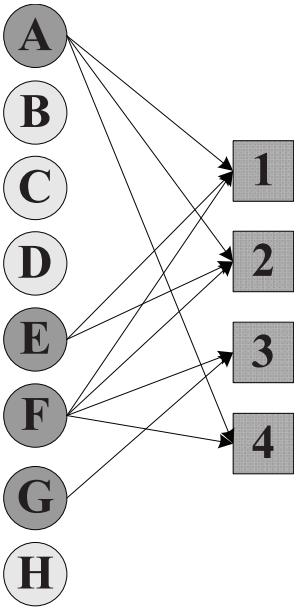

Fig 1. An assignment scheme without participation constraint.

https://doi.org/10.1371/journal.pone.0201596.g001

of skills $s(x_i)$. Additionally, we assume a set of m projects $\mathcal{P}$, for each project $p_j \in \mathcal{P}$, $s(p_j)$ is composed of the skills required to complete the project. Finally, every expert is associated with a cost function $c(x_i)$ which corresponds to x_i's compensation and a participation constraint function $w(x_i)$ which represents the number of projects expert x_i can engage in at most during the same period of time. Our goal is to form a team of participation-constrained experts $\mathcal{T} \subseteq \mathcal{X}$ to complete all given projects such that the total cost is minimized. We assume that $\mathcal{T}$ can complete all the projects $\mathcal{P}$ only if for each skill required by project $p_j \in \mathcal{P}$, there exists at

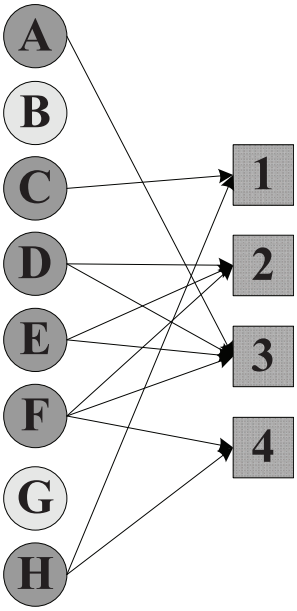

Fig 2. The cost of an assignment scheme with participation constraint is 27.

https://doi.org/10.1371/journal.pone.0201596.g002

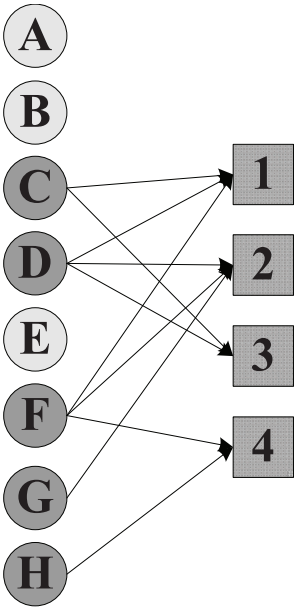

Fig 3. The cost of an assignment scheme with participation constraint is 21.

https://doi.org/10.1371/journal.pone.0201596.g003

least one member in $\mathcal{T}$ who can cover it. To be clear, for each expert we only consider his skills which are required by projects. We call this problem the participation constrained team hire problem. In addition to the fundamental problem, we also tackle a special case of participation constrained team hire problem where the number of projects is within the participation constraint of each expert, such that we can ignore the participation constraint of experts. We dub the preceding special case the participation free team hire problem. Although both the participation constrained team hire problem and the participation constrained team formation problem proposed in [4] consider the participation constraint, they have different objectives.

In this paper we proposed the participation constrained team hire problem and a special case of it, and our major contributions are summarized as follows:

1. To the best of our knowledge, we are the first to define and study the participation constrained team hire problem (PCTH). We impose the participation constraint of experts so that no expert is overworked under the resulting assignment scheme. We define a special case of PCTH called participation free team hire problem (PFTH) where the number of projects stays within the participation constraint of each expert.
2. Though we prove that PFTH is NP-hard, two exact algorithms, integer programming and Linking-Pruning Algorithm (LPA), are proposed, which can discover the best team based on well-designed pruning strategies.
3. We show that PCTH is NP-hard to solve, and design three effective algorithms, each of which focuses on a particular dimension of the problem. Then, we conduct a careful and detailed set of experiments to evaluate the performance of the proposed algorithms.

The rest of the paper is organized as follows: in Section we review the related work. Section formally defines the PCTH and PFTH respectively, and analyzes the complexity of the two problems. Our exact algorithm for PFTH is described in Section, and in Section we present three algorithms for PCTH. In Section we perform extensive experimental studies for the evaluation of our methods. We conclude the paper in Section.

Related work

To the best of our knowledge, we are the first to introduce and study PCTH. However, our problem is also related to some well-studied ones. We give an overview of their treatment on this subject below.

(Team Formation Problem) Lappas et al. [14] first introduced team formation in the context of social networks. In recent years, many researchers [15–22] extend this work. All these extended variations except [17] assume the context of a social network and therefore, their formulations and solutions are graph-theoretically based. Our work does not make this assumption and differ markedly from theirs. Anagnostopoulos et al. [17] do not assume a network of experts. In their paper, a collection of projects with different skill requirements arrive one at a time in an online fashion, and for each project coming, they create a team for it. Their goal is to minimize the maximum number of teams that each expert participates in. Obviously, our work diverges considerably from theirs in three aspects. First, our projects are known apriori. Second, we only create a single team for all the projects and do not create teams for each project individually. Third, their optimization aim is to minimize the maximum number of teams that each expert participates in while our goal is to minimize the overall compensation of the team.

(Set Cover Problem) Our work is also related to the Set Cover (SC) problem [23–26], especially the weighted Set Cover (WSC) problem [27–30] and the Set Multicover (SMC) problem [31–34]. In the set cover problem, given a universal set E and a set of subsets of it which are called S, the goal is to find a minimized collection of sets from S such that it covers all the elements in E. Weighted Set Cover problem defines a nonnegative weight for each set in S, and attempts to minimize the total cost of the found sets. Set Multicover problem is predicated on a multiset N instead of the universe E. Here a multiset N contains a specified number of duplicates of each element $n_i \in N$, which is denoted by b_i. The objective is to find a minimum cardinality subset such that each element $n_i \in N$ is covered by b_i times. All the Set Cover problems involve one universal set, However, because of the participation constraint, we are unable to merge the projects together, which implies that there is more than one universal set. This is where the primary distinction between these two problems lies.

(Cluster Hire Problem) Perhaps the closest work to ours is the Cluster Hire problem [13]. Given a set of projects, each project is characterized by the skills that are necessitated for its completion. Additionally, each project is associated with a profit gained upon its completion, and every expert incurs a cost corresponding to his compensation. The goal of ClusterHire is to form a team of experts such that the total cost does not exceed the specified budget and the total profit stemming from the projects accomplished by the team peaks. Differences between Cluster Hire problem and our problem are two-fold. First, our problem takes the participation constraint into consideration which implies our experts are not inexhaustible. But experts in Cluster Hire problem are inexhaustible and an expert can be assigned to an arbitrary number of projects. Second, our problem intends to create a team of experts to handle all the projects while minimizing the overall compensation. However, Cluster Hire aims to form a team of experts whose total salaries stays below the budget while maximizing the overall financial gain. In their work [13], they also consider a variant of ClusterHire. The variant places an upper bound on the number of projects for which an expert can utilize a skill a_k. This setting is different from ours since our participation constraint limits the individual rather than their skills. [6] proposed an more effective algorithm for the Cluster Hire problem. [4] imposed a participation constraint on the Cluster Hire Problem, and proposed an effective algorithm for the problem.

(Reviewer Assignment Problem (RAP)) Reviewer Assignment Problem [35–38], which coordinates the assignment of reviewers to papers, also behaves like our problem. However,

differences between the two problems are also evident. First, each paper must be reviewed by a fixed number of reviewers in the setting of RAP while in our problem, the size of expert set attached to each project is not rigidly constrained. Second, the skills required by each project in our problem must be totally covered while the topics of each paper are not purported to be completely satisfied in RAP, and in most cases, this is expected. Third, the ultimate objective of our problem is to minimize the overall compensation, however, RAP devotes itself to maximizing the covered topics.

Problem definition

In this section, we first introduce some concepts we will use to define the problems. Then, we formulate the participation constrained team hire problem and participation free team hire problem respectively, and analyze their corresponding complexity.

Concepts

We assume there is a set of k skills $\mathcal{A} = \{a_1, \ldots, a_k\}$, a set of m projects $\mathcal{P} = \{p_1, \ldots, p_m\}$ and a set of n experts $\mathcal{X} = \{x_1, \ldots, x_n\}$. Projects $\mathcal{P}$ need to be all accomplished and we use a skill function (s), such that for each project $p_j \in \mathcal{P}$, $s(p_j)$ denotes the set of skills required by p_j for its completion, $s(p_j) \subseteq \mathcal{A}$. Similarly, each expert $x_i \in \mathcal{X}$ is associated with a set of skills which we also designate it by $s(x_i)$, $s(x_i) \subseteq \mathcal{A}$. In addition, we have a cost function (c) and a participation constraint function (w), such that for every $x_i \in \mathcal{X}$, $c(x_i)$ gives the cost of hiring x_i and $w(x_i)$ specifies the maximum number of projects x_i can engage in simultaneously, $w(x_i) \geq 1$.

To complete all the obligatory projects we need to hire a team of experts. Let $\mathcal{T} \subseteq \mathcal{X}$ be a team established to cover the requirements of all the projects. $\mathcal{T}$ also constitutes a certain skill set, which is computed as the union of the skills of its members. That is, $s(\mathcal{T}) = \cup_{x_i \in \mathcal{T}} s(x_i)$. After a team of experts $\mathcal{T}$ is formed, each project $p_j \in \mathcal{P}$ can be completed by one of $\mathcal{T}$'s subsets. We define a complete function (com), such that for each $p_j \in \mathcal{P}$, $com(p_j)$ stands for a subset comprising experts of the formed team which are allocated to p_j. Taking Fig 3 as an example, $com(2) = \{D, F, G\}$, which represents that experts D, F and G are in charge of project 2. Table 3 summarizes the terse notations we described above.

For a team of experts $\mathcal{T} \subseteq \mathcal{X}$ and a project $p_j \in \mathcal{P}$, we say that $\mathcal{T}$ can cover p_j if $\mathcal{T}$ encompasses all the required skills for p_j, i.e., $s(p_j) \subseteq s(\mathcal{T})$. Obviously, the formed team is capable of covering more than one project. Thus, we introduce the coverage of a team $\mathcal{T}$ in Definition 1,

Table 3. Notations.

$\mathcal{A}$	Set of skills
$\mathcal{P}$	Set of projects
$\mathcal{X}$	Set of experts
$\mathcal{T}$	A formed team
k	Number of skills
m	Number of projects
n	Number of experts
$s(x_i)$	Skills of expert x_i
$s(p_j)$	Skills of project p_j
$c(x_i)$	Compensation cost of expert x_i
$w(x_i)$	Participation constraint of expert x_i
$com(p_j)$	Subset experts assigned to project p_j

https://doi.org/10.1371/journal.pone.0201596.t003

and a similar notion can be found in ClusterHire [13]. To ensure all the team members are not overworked, we present the feasibility of a team in Definition 2. Additionally, every team incurs certain expenses, hence Definition 3 gives the computed total cost of a team.

Definition 1 (Coverage). Given a set of projects $\mathcal{P}$ and a team $\mathcal{T}$, we define the coverage of $\mathcal{T}$ to be the set of projects that $\mathcal{T}$ can cover. That is,

$$Cov(\mathcal{T}) = \{p_j | p_j \in \mathcal{P} \wedge s(p_j) \subseteq s(\mathcal{T})\}. \quad (1)$$

As illustrated in Fig 3, the coverage of team X'' is $Cov(\mathcal{X}'') = \{1, 2, 3, 4\}$.

Definition 2 (Feasible Team). Given a team of experts $\mathcal{T} \subseteq \mathcal{X}$ which is formed to handle a set of projects $\mathcal{P}$, we say that $\mathcal{T}$ is a feasible team if for each $x_i \in \mathcal{T}$, the number of projects he participates in is within his participation constraint, i.e., for each $x_i \in \mathcal{T}$, $|\cup_{p_j \in \mathcal{P}: x_i \in com(p_j)} p_j| \leq w(x_i)$.

In our running example, team X shown in Fig 1 can not be a feasible team because expert F shares the workload with others in 4 projects in parallel while his given participation constraint $w(F)$ in Table 2 falls short, i.e., $\left|\cup_{p_j \in \mathcal{P}: F \in com(p_j)} p_j\right| = |\{1, 2, 3, 4\}| = 4 \geqslant w(F) = 3$. Apart from F, expert A whose participation constraint has been violated too also renders team X infeasible.

Definition 3 (Team Cost). Given a team of experts $\mathcal{T} \subseteq \mathcal{X}$, we define the cost of the team as $c(\mathcal{T})$, computed by the sum of the costs of its members. That is,

$$c(\mathcal{T}) = \sum_{x_i \in \mathcal{T}} c(x_i). \quad (2)$$

As we can see in Table 2, expert E is associated with the compensation cost 4, notated $c(E) = 4$. Also, we can easily calculate the total cost of the team X'' in Fig 3 as $c(X'') = c(\{C, D, F, G, H\}) = 3 + 6 + 5 + 3 + 4 = 21$.

The participation constrained team hire

Having introduced the foregoing preliminaries, we can now formulate the participation constrained team hire problem addressed in this paper as follows:

Problem 4. Given a set of projects $\mathcal{P}$, a set of experts $\mathcal{X}$, we seek to find a team $\mathcal{T} \subseteq \mathcal{X}$, such that

1. $Cov(\mathcal{T}) = \mathcal{P}$;
2. $\mathcal{T}$ is a feasible team;
3. $c(\mathcal{T})$ is minimized.

We abbreviate the name of the problem to PCTH. By definition, PCTH is a constrained optimization problem. From the computational point of view, we have following results for this problem.

Theorem 1. *The decision version of participation constrained team hire problem is NP-complete.*

Proof. We prove the theorem by a reduction from the SetCover problem. In the classical SetCover problem there is a universe of items U and a set of sets $S = \{S_1, S_2, \ldots, S_k\}$ such that for every $S_i \in S$, $S_i \subseteq U$. Given a constant K, the decision version of SetCover problem is whether there exists $S' \subseteq S$ such that $\cup_{S_i \in S'} S_i = U$ and $|S'| \leq K$.

Now, we concentrate on a simplified version of the problem which stipulates that experts can participate in all projects without any constraints, i.e., $\forall x_i \in \mathcal{X}$, $w(x_i) = \infty$. Moreover, we specifically consider a special case that $\mathcal{P}$ consists solely of a single project and $c(x_i) = 1$. In this case, we are only concerned with the amount of experts. Thus, the problem now transforms into finding a feasible assignment X' that minimizes the cost to complete all projects.

Clearly, if we map every set $S_i \in S$ from SetCover problem onto $s(x_i)$ of PCTH, the two problems become identical. That is, there exists a solution of cost K in the SetCover problem if and only if there exists a solution of cost K in PCTH.

Theorem 2. *The participation constrained team hire problem is NP-hard to approximate.*

Proof. The proof of the above theorem leverages the same simplified decision version of PCTH employed in the proof of theorem 1. We create an instance Γ of SetCover and a PCTH instance T based on the simplified decision version. Through our construction, $OPT_\Gamma = OPT_T$, i.e., a feasible solution for instance Γ is identical to the one for instance T.

We now prove this theorem by contradiction. That is, assume that there exists an approximation algorithm Λ with approximation guarantee [39] α (also called approximation factor, i.e. the supremum of the fraction of the approximate value to the optimum value for all the problem instances) for this simplified version of our problem. Then, running Λ on T can decide whether a solution comprising K experts who manage to perform all the projects of our problem can be discovered. Apparently, algorithm Λ is suitable for instance Γ. However, this deduction flatly contradicts to the previous findings by Lund and Yannakakis who showed that SetCover problem cannot be approximated in polynomial time unless NP has quasi-polynomial time algorithms [23]. Therefore, such an approximation algorithm with approximation guarantee α does not exist.

In the definition of PCTH, we focused on minimizing the compensation cost with participation constraint. If the participation constraint was not a concern, our goal would change to find a team $\mathcal{T}$ such that $Cov(\mathcal{T}) = \mathcal{P}$ and $c(\mathcal{T})$ is minimized. Such a problem definition is actually an instance of the classic Weighted Set Cover problem since all the projects can be merged into one whose required skills is the union of its members'. If the merged project asks for a particular skill a_k, so will at least one of the projects constituting its combined counterpart. Besides, that a_k is demanded by most projects further aggravates this issue. To put it simply, if an expert who owns the skill a_k is selected to cover the projects, he is very likely to join too many projects and overwork. Our work precisely attacks such a problem. Taking the participation constraint into account, each expert can only be assigned to a limited (and usually very small) number of projects, suggesting that the projects ought not to be merged into one which covers each skill only once.

Alternatively, we may attempt to aggregate the projects into one and convert the skill set of this combined version into a multi-set, so the crux of the issue can be now viewed as a Set Multicover problem. Clearly, the number of duplicates of each skill indicates how many times the skill should be utilized. However, having merged all of them, we are not able to discern which project each skill initially belongs to. Furthermore, even if a particular team may seem fit for the merged project, where each skill required can be covered multiple times, we can hardly add up the statistics of projects that an expert in the team enters in parallel. Consequently, whether the participation constraint is satisfied can not be interpreted from this team formation scheme, once again reminding us that the projects ought not to be integrated into one.

Therefore, the essence of PCTH is the participation constraint and the critical factor of the solution resides in the fact that the projects must be kept separate from one another and can not be merged. Here lies the core difference between previous pertinent work and our problem which is more common in practical applications and more difficult to tackle.

The participation free team hire

In this section, we present a special case of the participation constrained team hire problem, called participation free team hire, where the participation constraint of each expert exceeds the total quantity of all the projects. Therefore, we can ignore the participation constraint of

experts, since even if an expert engages in all the projects, the participation constraint will not be violated. Therefore, given a set of m projects $\mathcal{P} = \{p_1, \ldots, p_m\}$, our goal is to hire a team of experts to manage these projects and minimize the overall compensation cost. Formally, the participation free team hire problem (PFTH) can be condensed into the following definition:

Problem 5. Given a set of projects $\mathcal{P}$, a set of experts $\mathcal{X}$, we seek to find a team $\mathcal{T} \subseteq \mathcal{X}$, such that

1. $Cov(\mathcal{T}) = \mathcal{P}$;
2. $c(\mathcal{T})$ is minimized.

As has been discussed above, with the participation constraint having no decisive effect on our problem, the candidate projects can be merged into a larger one whose required skills is the union of skills of its members. The following theorem proves the NP-hardness of PFTH.

Theorem 3. *The participation free team hire problem is NP-hard.*

Proof. We will show that the NP-hard Weighted Set Cover problem can be reduced to an instance of PFTH. Given a universe of items U and a set of sets $S = \{S_1, S_2, \ldots, S_k\}$ such that for every $S_i \in S$, $S_i \subseteq U$, where each set S_i is assigned a cost. The Weighted Set Cover problem attempts to find a subset $S' \subseteq S$ such that $\bigcup_{S_i \in S'} S_i = U$ and the total cost of S' is minimized. Our problem considers a set of projects $\mathcal{P}$, and a set of experts $\mathcal{X}$ where each expert $x_i \in \mathcal{X}$ features a skill set $s(x_i)$ and a cost $c(x_i)$. The goal is to work out a combination of experts which can collectively cover the projects and the total cost is minimized. Since projects in PFTH can be merged into one and the skills required by the project is the union of the skills of its members, we can first aggregate the projects into one project P', and then discover a team of experts to handle P' while minimizing the total cost. PFTH is equivalent to the Weighted Set Cover problem if we map every set S_i from the Weighted Set Cover problem onto an expert skill set $s(x_i)$ of PFTH and similarly map U onto the skill set of project P'. Thus, the PFTH problem is NP-hard.

The most valuable feature of PFTH rests on a special case where only a single project needs to be completed (i.e., $\mathcal{P} = \{p\}$). This case frequently emerges in practical applications. For instance, a software company is looking for programmers for one cellphone application or the medical personnel want to closely cooperate with their peers and perform an emergency surgery for their patients. These scenarios merely require one project.

Two exact algorithms for pfth

Here, we introduce two exact algorithms for problem PFTH. In Section, we introduce the linking-pruning algorithm based on the Aprior algorithm. In Section, we introduce the integer programming based algorithm.

Linking-pruning algorithm

Below, we introduce an exact algorithm for PFTH as a baseline. This algorithm is based on Apriori algorithm [40]. Its time efficiency is comparable to integer programming when the number of skills is relatively small. In real world, experts usually have relative small skill. Obviously, Brute Force Search which enumerates every possible team can be employed to address our problem exactly. However, this solution is very sensitive to the size of expert set and does not scale well since it examines every possible permutation. In this section, we delineate our exact algorithm Linking-Pruning algorithm (LPA) for PFTH. Given $\mathcal{P} = \{p_1, \ldots, p_m\}$, from the problem definition we can merge $\mathcal{P}$ into a large project P', and the skills required by P' is the union of skills of its members, i.e., $s(P') = \cup_{p_j \in \mathcal{P}} s(p_j)$. To be clear, the algorithm described in this section proceeds to the completion of P'. Evidently, if P' is done, so will be all the projects in $\mathcal{P}$.

We adopt the thought of Apriori Algorithm [40] for mining association rules to reduce the search space of our problem. That is, LPA employs an iterative method searching layer by layer to examine all the promising permutations which eventually facilitate identifying the optimal solution. First, given *Eset* to represent an expert set, and *k-Eset* for an *Eset* with k experts. In each layer, we start with a seed set of *k-Eset*s, notated as L_k ($k \geq 1$), and try to use L_k to generate L_{k+1}. However, the scale of L_k might be large, so the computational cost can be prohibitively high. To compress L_k, we scan the whole L_k and determine which of those *k-Eset*s in L_k has the potential to be a component of the best team. We then exclude those *k-Eset*s which would never contribute to our solution, and obtain the k-candidate expert set I_k^E. Finally, we link I_k^E with I_k^E to generate all $(k+1)$-*Eset*, i.e., L_{k+1}, and then L_{k+1} becomes the seed set for the next layer. The main process consists of two steps: Linking and Pruning.

Linking. To reduce the search space, I_k^E instead of L_k is used to generate L_{k+1}, since all *k-Eset*s in I_k^E are potential candidates. We achieve this by linking (notated as $\bowtie$) I_k^E with I_k^E. What should be clear is that, $\forall i, j \in I_k^E$, i and j can be linked only if the newly formed expert set is a $(k+1)$-*Eset*, i.e., $L_{k+1} = I_k^E \bowtie I_k^E = \{i \cup j | i, j \in I_k^E \wedge |i \cup j| = k+1\}$. The constraint $|i \cup j| = k+1$ stipulates that every generated expert set is a $(k+1)$-*Eset*.

We now illustrate the linking step with our running example. Let I_2^E be {{*A*, *G*}, {*C*, *G*}, {*G*, *H*}, {*E*, *F*}}. After the linking step, L_3 will be {{*A*, *C*, *G*}, {*A*, *G*, *H*}, {*C*, *G*, *H*}}. The expert sets {*A*, *C*} and {*E*, *F*} can not be linked because their union {*A*, *C*, *E*, *F*} is not a 3-*Eset*.

Pruning. L_k is a superset of I_k^E and may contain some expert sets which can be excluded. We now introduce a definition that underlies the exclusion of such *k-Eset*. The minimum cost threshold is designated by *min_cost* which always records the cost of the current optimal team. The definition is as follows:

Definition 6 (Minimum Cost Threshold). Given a project P' and a set of expert sets L_k, the minimum cost threshold, if exists, is notated as *min_cost* representing the lowest cost of the expert set which can cover the project P', i.e.,

$$min\_cost = \min\{c(Eset) | Eset \in L_k \wedge s(P') \subseteq s(Eset)\}.$$

Based on Definition 6, we present the following property.

Property 4. *Expert sets in L_k whose costs exceed the minimum cost threshold can be excluded from L_k.*

In addition, assume there exist two expert sets *Eset*1, *Eset*2 $\in L_k$, if *Eset*1 contains all the skills that *Eset*2 possesses and the cost of hiring *Eset*1 is less than hiring *Eset*2, we can replace *Eset*2 with *Eset*1 and remove *Eset*2 from L_k. So we have the following property.

Property 5. *Given a set of expert sets L_k, $\forall$Eset*1, *Eset*2 $\in L_k$, *if Eset*1 *can cover Eset*2 *(i.e., s(Eset*2*)* $\subseteq$ *s(Eset*1*)) and c(Eset*1*)* $<$ *c(Eset*2*), we can exclude Eset*2 *from L_k.*

We draw on both properties for pruning. If a *k-Eset* in L_k can be pruned, it must comply with the requirement of either of the two properties. In our running example, we assume L_1 is {{*A*}, {*B*}, {*C*}, {*D*}, {*E*}, {*F*}, {*G*}, {*H*}} and *min_cost* is 8. We take the pruning step to discard 1-*Eset* {*B*} because the cost of {*B*} surpasses 8, i.e., $c(B) = 10 > min\_cost$. On the other hand, we can delete 1-*Eset* {*A*} and {*E*} in the pruning step, owing to the fact that their skills can be covered by other 1-*Eset* whose costs are less than theirs. Here, they can entirely be substituted by another two 1-*Eset*s {*H*} and {*G*} respectively.

Algorithm 1 shows the pseudo-code of LPA. In Algorithm 1, first the projects $\mathcal{P}$ coalesce into one project P' and we take a greedy strategy to work out a near-optimal solution (lines 2-8). The strategy greedily selects experts, one at a time, and assign it to the project P'', a duplicate of P' (lines 4-5). Experts who can cover more skills of the project and incur lower cost are preferred. This greedy approach yields a current optimal team $\mathcal{T}$ and its corresponding

min_cost. In lines 9-27, we search L_k for the *k-Eset*, which is returned as the output, that can handle P' and carries the least cost. Firstly, L_1 is initialized to a set of 1-*Eset*s, each of which is constituted by an expert in $\mathcal{X}$ (line 9). After that, we search L_1 and update $\mathcal{T}$ and *min_cost* (lines 10-14), and by pruning, we get I_1^E (line 15). For each k ($k \geq 2$), the algorithm first generates L_k by linking I_{k-1}^E and I_{k-1}^E (line 17), and if L_k is empty, we will return the optimal team $\mathcal{T}$ and its corresponding *min_cost* (lines 18-20). Then, we try to identify the most optimal solution in current L_k, if exists (lines 21-25). After that, it prunes the elements in L_k according to Property 4 and Property 5 (line 26). Consequently, I_k^E which is used to be linked and generate L_{k+1} is formed.

Algorithm 1 Linking-Pruning Algorithm.

Require: project set $\mathcal{P}$, expert set $\mathcal{X}$, cost function c, skill function s
Ensure: a team $\mathcal{T}$ and the corresponding cost *min_cost*

1: $\mathcal{T} \leftarrow \emptyset$, *min_cost* $\leftarrow 0$
2: $P' \leftarrow merge(\mathcal{P})$, $P'' \leftarrow P'$
3: **while** $s(P'') \neq \emptyset$ **do**
4: $x_i \leftarrow \arg\max_{x_i \in \mathcal{X} \setminus \mathcal{T}} \frac{|s(P'') \cap s(x_i)|}{c(x_i)}$
5: $s(P'') \leftarrow s(P'') \setminus s(x_i)$
6: $\mathcal{T} \leftarrow \mathcal{T} \cup \{x_i\}$
7: *min_cost* $\leftarrow$ *min_cost* + $c(x_i)$
8: **end while**
9: $L_1 \leftarrow \{\{x_i\} | x_i \in \mathcal{X}\}$
10: **for** *Eset* $\in L_1$ **do**
11: **if** c(*Eset*) < *min_cost* and s(*Eset*) $\supseteq s(P')$ **then**
12: *min_cost* $\leftarrow c$(*Eset*), $\mathcal{T} \leftarrow$ *Eset*
13: **end if**
14: **end for**
15: $I_1^E \leftarrow L_1$ //Pruning
16: **for** ($k \leftarrow 2$;; k++) **do**
17: $L_k \leftarrow I_{k-1}^E \bowtie I_{k-1}^E$ //Linking
18: **if** $L_k = \emptyset$ **then**
19: **return** $\mathcal{T}$, *min_cost*
20: **end if**
21: **for** *Eset* $\in L_k$ **do**
22: **if** c(*Eset*) < *min_cost* and s(*Eset*) $\supseteq s(P')$ **then**
23: *min_cost* $\leftarrow c$(*Eset*), $\mathcal{T} \leftarrow$ *Eset*
24: **end if**
25: **end for**
26: $I_k^E \leftarrow L_k$ //Pruning
27: **end for**

The procedure of LPA is well exemplified in Fig 4. In this example, experts and projects are apparently identical to those in Table 1 and we assume a project P', which is merged by a set of projects $\mathcal{P} = \{1, 2\}$, requires a set of skills $S(P') = \{a, b, c, d, e, f\}$. Obviously, the participation constraint of each expert is beyond the total number of projects (i.e., 2). Our illustration focuses on the linking and pruning steps, and we assume a current optimal team $\mathcal{T} = \{A, F, G\}$ and the corresponding *min_cost* 13 being reported by the greedy strategy of LPA (Note that for now, $\mathcal{T}$ is not the final outcome of the greedy strategy, and we choose this value simply for the sake of discussion).

From Fig 4 we observe that L_1 is a set of 1-*Eset*s, each of which is composed of a single expert. By pruning, we can exclude $\{A\}$ and $\{E\}$ from L_1 since they can entirely be replaced by $\{H\}$ and $\{G\}$ respectively (according to Property 5). After pruning, the 1-candidate expert set I_1^E is formed, which can be seen from Fig 4. I_1^E is employed for linking and generating L_2. Similarly, by pruning and linking, we can generate L_3 from L_2. In L_3, we obtain the current optimal team

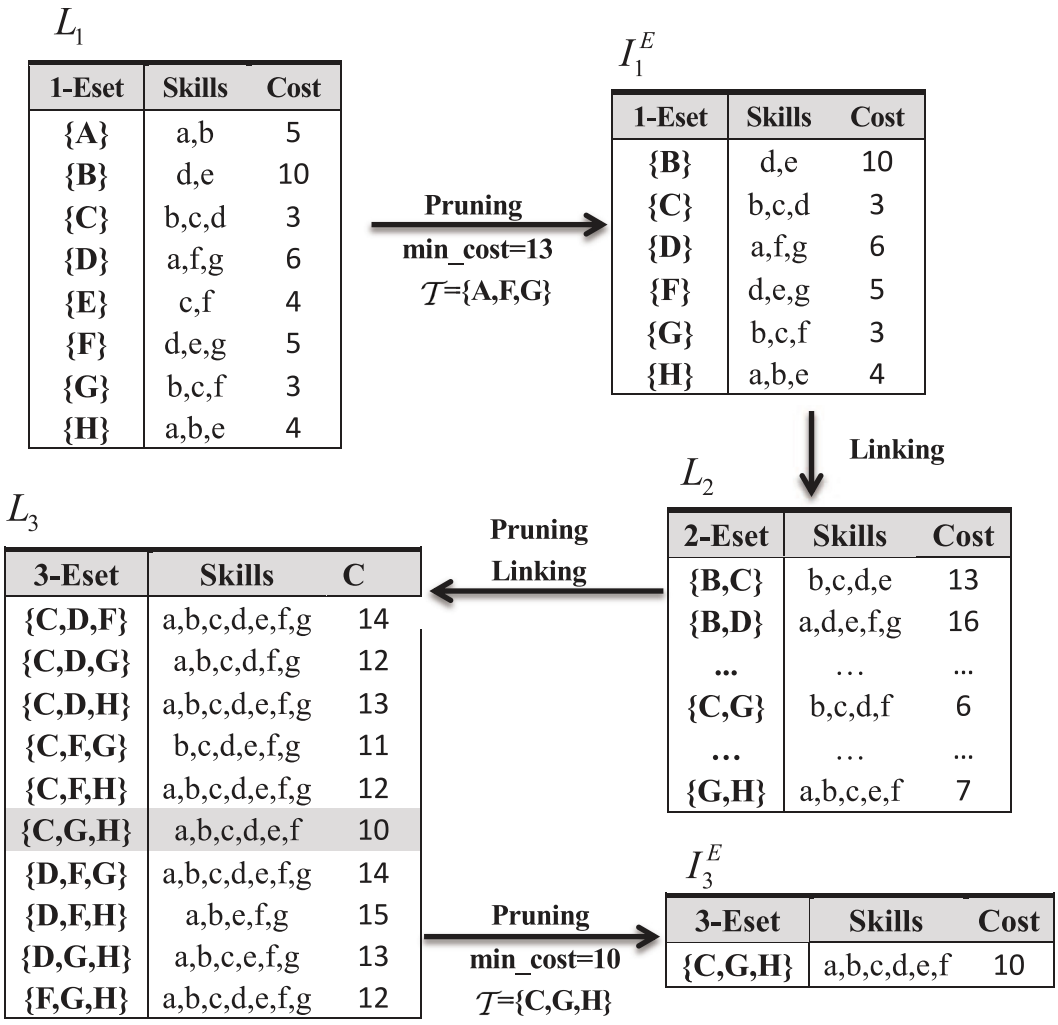

Fig 4. Procedure of LPA.

https://doi.org/10.1371/journal.pone.0201596.g004

{*C*, *G*, *H*} which is preferable to {*A*, *F*, *G*}. Then, we update $\mathcal{T} = \{C, G, H\}$ and *min_cost* = 10. Afterwards, we apply pruning and the 3-candidate expert set I_3^E is formed. There exists only one 3-*Eset* {*C*, *G*, *H*} in I_3^E suggesting that L_4 can not be generated from only one 3-*Eset*. Therefore, we terminate the running process and return {*C*, *G*, *H*} as the best team $\mathcal{T}$.

Integer programming based algorithm

In essence, problem PFTH can be solved using integer programming. Let $\mathcal{A} = \{a_1, \cdots, a_k\}$ denote the set of skills required by the projects in $\mathcal{P}$ without loss of generality. For any expert $x_i \in \mathcal{X}$, we remove its skills not in $\mathcal{A}$. Let y_{ij} indicate whether expert $x_i \in \mathcal{X}$ apply skill $a_j \in \mathcal{A}$ to the project set $\mathcal{P}$. That is to say, $y_{ij} = 1$ if x_i participate in the project set $\mathcal{P}$ by his/her skill a_j, 0 otherwise. As a result, PFTH can be formulated as

$$\begin{aligned} \textbf{min} \quad & \sum_{i=1}^{n} \sum_{j=1}^{k} c(x_i) \times y_{ij} \\ \textbf{s.t.} \quad & \sum_{i=1}^{n} \sum_{j=1}^{k} y_{ij} = k \end{aligned} . \tag{3}$$

Algorithms for pcth

In this section we describe three algorithms for solving PCTH: the ProjectGreedy, the ExpertGreedy and the ExpertProjectGreedy. We now introduce them respectively. These algorithms can be applied to different problem instances. ProjectGreedy can yield a team with less cost and small size, and has relatively high time efficiency. ExpertGreedy can yield a team with high participation rate, i.e. the fraction of the total number of experts assigned to projects to the total cost of the team. ExpertProjectGreedy can yield a team with high skill utilization, i.e. the skills can be fully used. In Section, we will conduct experiments to show these advantages.

The ProjectGreedy algorithm

The ProjectGreedy algorithm randomly picks projects, one at a time, which is then performed by the experts greedily selected by this technique. An expert x_i is assigned to the current project p_j if it maximizes:

$$\frac{|s(p_j) \cap s(x_i)| \cdot w(x_i)}{c(x_i)}, \tag{4}$$

which shows an intuitive way to curtail the overall compensation. According to Eq 4, experts mastering more pertinent skills are preferred when they are vying for the same project. Meanwhile, to minimize the overall compensation, loose participation constraints and low costs are expected.

Having engaged in a project and still conforming to his participation constraint, the expert will be greedily assigned to other projects depending on the relevance of his skills (i.e., similarity of their skill sets). The similarity between two skill sets s' and s'' is defined as follows:

$$sim(s', s'') = \frac{|s' \cap s''|}{|s' \cup s''|}. \tag{5}$$

Therefore, given an expert x_i, a project p_j is chosen to be covered such that it maximizes $sim(s(p_j), s(x_i))$. The pseudo-code of ProjectGreedy is listed in Algorithm 2.

Algorithm 2 Pseudo-code of ProjectGreedy.

Require: expert set $\mathcal{X}$, project set $\mathcal{P}$, participation constraint function w, cost function c, skill function s
Ensure: a team $\mathcal{T}$ and the corresponding cost $c(\mathcal{T})$

$\mathcal{T} \leftarrow \emptyset$
while $\mathcal{P} \neq \emptyset$ **do**
  $p_j \leftarrow randomselect(\mathcal{P})$
  while $s(p_j) \neq \emptyset$ **do**
    $x_i \leftarrow \arg\max_{x_i \in \mathcal{X}} \frac{|s(p_j) \cap s(x_i)| \cdot w(x_i)}{c(x_i)}$
    for $t \leftarrow 1$ to $w(x_i) - 1$ **do**
      $p' \leftarrow \arg\max_{p' \in \mathcal{P} \setminus \{p_j\}} sim(s(p'), s(x_i))$
      $s(p') \leftarrow s(p') \setminus s(x_i)$
    end for
    $\mathcal{X} \leftarrow \mathcal{X} \setminus \{x_i\}$
    $s(p_j) \leftarrow s(p_j) \setminus s(x_i)$
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{x_i\}$
  end while
  $\mathcal{P} \leftarrow \mathcal{P} \setminus \{p_j\}$
end while
return $\mathcal{T}, c(\mathcal{T})$

$\mathcal{T}$ is initialized to an empty set at the very beginning. After that, we start the iterative process on the space of projects. In each iteration, we first randomly select a project p_j from $\mathcal{P}$

(line 3), and then we greedily choose experts, one at a time, to perform p_j (line 5). The process does not cease until p_j is totally covered (lines 4-13). Afterwards, we exclude p_j from the project set $\mathcal{P}$ (line 14). To put it differently, an expert who has joined a project p_j will be continually and greedily assigned to other projects on condition that he still complies with his participation constraint (lines 6-9).

According to Table 3, the size of $\mathcal{P}$ is m (line 2), so the algorithm will iterate at most m times through the space of $\mathcal{P}$. For each project, we assume the average number of skills of it is q (line 4), so is the maximum amount of iteration through lines 4-13 for our algorithm. In each iteration, it takes n times to select an ideal expert (line 5), and at most $w^* m$ (w denotes the average number of projects in which an expert can participate simultaneously) times to assign the expert to other projects (lines 6-9). Therefore, the worst-case running time of ProjectGreedy adds up to $O(mq(n + wm))$, i.e., $O(mqn + wm^2q)$.

The ExpertGreedy algorithm

The ExpertGreedy algorithm greedily picks experts, one at a time, and then it greedily assigns the expert to projects. The expert is chosen from the expert set such that it maximize:

$$\frac{|ss \cap s(x_i)| \cdot w(x_i)}{c(x_i)}, \tag{6}$$

where $ss \leftarrow \cup_{p_j \in \mathcal{P}} s(p_j)$ denotes the union of skills of all the remaining projects. When choosing experts, ExpertGreedy perceives all the remaining projects as a whole and experts who cover more skills of the whole are preferred. Moreover, to minimize the compensation, the method favors experts with loose participation constraints and low costs. Once an expert has been chosen by the algorithm, he will be greedily assigned to projects according to the similarity of their skills. The pseudo-code of ExpertGreedy is listed in Algorithm 3.

Algorithm 3 pseudo-code of ExpertGreedy.

Require: expert set $\mathcal{X}$, project set $\mathcal{P}$, participation constraint function w, cost function c, skill function s
Ensure: a team $\mathcal{T}$ and the corresponding cost $c(\mathcal{T})$

1: $\mathcal{T} \leftarrow \emptyset$
2: $ss \leftarrow \cup_{p_j \in \mathcal{P}} s(p_j)$
3: **while** *true* **do**
4: $x_i \leftarrow \arg\max_{x_i \in \mathcal{X}} \frac{|ss \cap s(x_i)| \cdot w(x_i)}{c(x_i)}$
5: **for** $t \leftarrow 1$ to $w(x_i)$ **do**
6: $p_j \leftarrow \arg\max_{p_j \in \mathcal{P}} sim(s(x_i), s(p_j))$
7: $s(p_j) \leftarrow s(p_j) \setminus s(x_i)$
8: **if** $s(p_j) = \emptyset$ **then**
9: $\mathcal{P} \leftarrow \mathcal{P} \setminus \{p_j\}$
10: **end if**
11: **end for**
12: $\mathcal{T} \leftarrow \mathcal{T} \cup \{x_i\}$
13: $ss \leftarrow \cup_{p_j \in \mathcal{P}} s(p_j)$
14: **if** $ss = \emptyset$ **then**
15: **return** $\mathcal{T}, c(\mathcal{T})$
16: **end if**
17: $\mathcal{X} \leftarrow \mathcal{X} \setminus \{x_i\}$
18: **end while**

We also start with an empty $\mathcal{T}$ set and initialize set ss to the union of skills of all the projects (lines 1-2). Then, the algorithm keeps iterating until ss becomes empty (lines 3-18). In each iteration, we opt for the expert x_i who maximizes Eq 6 (line 4). After that, x_i is greedily assigned

to projects with the relevance of his skills being the chief determinant (lines 5-11). Moreover, *ss* gradually shrinks to an empty set as the project assignment progresses. If so, we return the team $\mathcal{T}$ and the corresponding cost and terminate the algorithm (lines 14-16).

In ExpertGreedy, the size of *ss* is k (number of skills), so it iterates at most $k * m$ times through lines 3-18 (assume each chosen expert can cover only one skill of one project). Then, it iterates n times in line 4 and $w * m$ (w denotes the average number of projects in which an expert can participate simultaneously) times through lines 5-11. Therefore, the worst-case running time of ExpertGreedy amounts to $O(km(n + wm))$, i.e., $O(kmn + km^2w)$.

The ExpertProjectGreedy algorithm

Algorithm 4 pseudo-code of ExpertProjectGreedy.

Require: expert set $\mathcal{X}$, project set $\mathcal{P}$, participation constraint function w, cost function c, skill function s
Ensure: a team $\mathcal{T}$ and the corresponding cost $c(\mathcal{T})$
1: $\mathcal{T} \leftarrow \emptyset$
2: $U \leftarrow \mathcal{X} \times \mathcal{P}$
3: $c_{temp} \leftarrow c$
4: **while** $U \neq \emptyset$ **do**
5: $(x_i, p_j) \leftarrow \arg\max_{(x_i,p_j)\in U} \frac{|s(x_i)\cap s(p_j)|}{c_{temp}(x_i)}$
6: $U \leftarrow U \backslash \{(x_i, p_j)\}$
7: $c_{temp}(x_i) \leftarrow 1$
8: $s(p_j) \leftarrow s(p_j) \backslash s(x_i)$
9: $w(x_i) \leftarrow w(x_i) - 1$
10: **if** $s(p_j) = \emptyset$ **then**
11: remove each $(\cdot, p_j) \in U$
12: **end if**
13: **if** $w(x_i) = 0$ **then**
14: remove each $(x_i, \cdot) \in U$
15: **end if**
16: $\mathcal{T} \leftarrow \mathcal{T} \cup \{x_i\}$
17: **end while**
18: **return** $\mathcal{T}, c(\mathcal{T})$

ProjectGreedy and ExpertGreedy start with project selection and expert selection respectively. Unlike the two algorithms, in this section, we propose another alternative dubbed ExpertProjectGreedy which combines an expert and a project into a match pair and perceives them as a whole. A match pair (x_i, p_j) represents that an expert x_i is assigned to a project p_j. We also define the marginal gain of assignment (x_i, p_j) as $\frac{|s(x_i)\cap s(p_j)|}{c(x_i)}$. Initially possible match pairs totaling $|\mathcal{X} \times \mathcal{P}|$ constitute a set U. During the execution, the algorithm greedily picks match pairs from U, one at a time, such that it maximizes the marginal gain. When a match pair (x_i, p_j) is picked by our algorithm, the expert x_i is assigned to the project p_j, and we update the status of x_i ($w(x_i) = w(x_i) - 1$) and p_j ($s(p_j) = s(p_j) \backslash s(x_i)$). If an expert x_i is not available ($(w(x_i) = 0)$) any more, we will remove all the match pairs involving x_i $(x_i, \cdot)$ from U. By the same token, if a project p_j is totally covered, we too will remove all the match pairs involving p_j $(\cdot, p_j)$ from U. The pseudo-code of ExpertProjectGreedy is displayed in Algorithm 4.

First we initialize $\mathcal{T}$ to an empty set and the set of potential match pairs to $\mathcal{X} \times \mathcal{P}$, with c_{temp} being a duplicate of c. The match pair yielding the largest marginal gain is picked in line 5. After a match pair (x_i, p_j) has been chosen, we exclude it from U and update the status of x_i and p_j (lines 7-9). The reason for setting $c_{temp}(x_i) = 1$ is that the cost of a selected expert is only considered at most once in the entire process. The algorithm retains the possible matches in U in lines 10-15. It can be observed that ExpertProjectGreedy iterates at most $|\mathcal{X} \times \mathcal{P}|$ times to attain an ideal team, but in practice, the value can be much smaller.

The while-loop iterates at most $m * q$ (q counts the average number of skills of each project) times (although the size of U is $n * m$, the while-loop will be halted once all the projects have been performed). Within each iteration of the while-loop, it takes at most $n * m$ times to pick a match pair reaping the most benefit (line 5). So ExpertProjectGreedy gives the worst-case running time $O(mq(n * m))$, i.e., $O(m^2 nq)$.

Experiments

In this section, we evaluate the performance of the proposed algorithms through experiments. Our algorithms are implemented using Java. All the experiments are conducted on a PC with Intel(R) Core(TM) 2.94GHz CPU and 2.0GB memory.

Datasets

Our experiments are performed on both real and synthetic datasets. The real datasets are collected from two large labor markets: freelancer.com and guru.com, which we refer to as Freelancer and Guru respectively. On both websites, employers post projects with the required skills that they are avidly seeking. Experts with different skillsets and salary demands apply for one or more projects, and are evaluated by the employers. Besides, for each expert, we impose a participation constraint on him which restricts the maximum number of projects he can enter in parallel. We randomly generate this constraint ranging from 1 to 3 for all experts. Additionally, the synthetic data which is named SynData is also produced in a random manner. Summary statistics from these datasets are exhibited in Table 4. In Table 4, $|\mathcal{P}|$, $|\mathcal{X}|$ and $|\mathcal{A}|$ count the number of projects, the number of experts and the number of skills respectively. For example, we glean information on 6363 experts and 1239 projects which embody 592 skills for Guru dataset. $\overline{|s(p_j)|}$, $\overline{|s(x_i)|}$ and $\overline{|c(x_i)|}$ stand for the average number of skills per project, the average number of skills per expert and the average cost per expert respectively. The maximum/minimum number of skills regarding all projects is denoted by $|s(p_j)|_{max}$ and $|s(p_j)|_{min}$. Analogous to the treatment for projects, $|s(x_i)|_{max}$ and $|s(x_i)|_{min}$ represent the maximum/minimum number of skills concerning all the experts.

Performance evaluation for pfth

In this section, we evaluate the efficiency of the integer programming compared with the Linking-Pruning algorithm (LPA) and the Brute Force Search (BFS). We draw on the knowledge from Section that LPA is particularly sensitive to the number of skills of the merged project and the number of experts. Hence, the effect of $|s(P')|$ (i.e., the number of skills of the merged

Table 4. Summary statistics from datasets.

Statistics	Guru	Freelancer	SynData
$\|\mathcal{P}\|$	1239	464	500
$\|\mathcal{X}\|$	6363	1147	1500
$\mathcal{A}$	592	55	100
$\overline{s(p_j)}$	4.098	3.254	17.518
$\|s(p_j)\|_{max}$	28	5	30
$\|s(p_j)\|_{min}$	1	1	5
$\overline{s(x_i)}$	9.606	4.59	3
$\|s(x_i)\|_{max}$	104	5	5
$\|s(x_i)\|_{min}$	1	1	1
$\overline{\|c(x_i)\|}$	37.891	19.609	12.598

https://doi.org/10.1371/journal.pone.0201596.t004

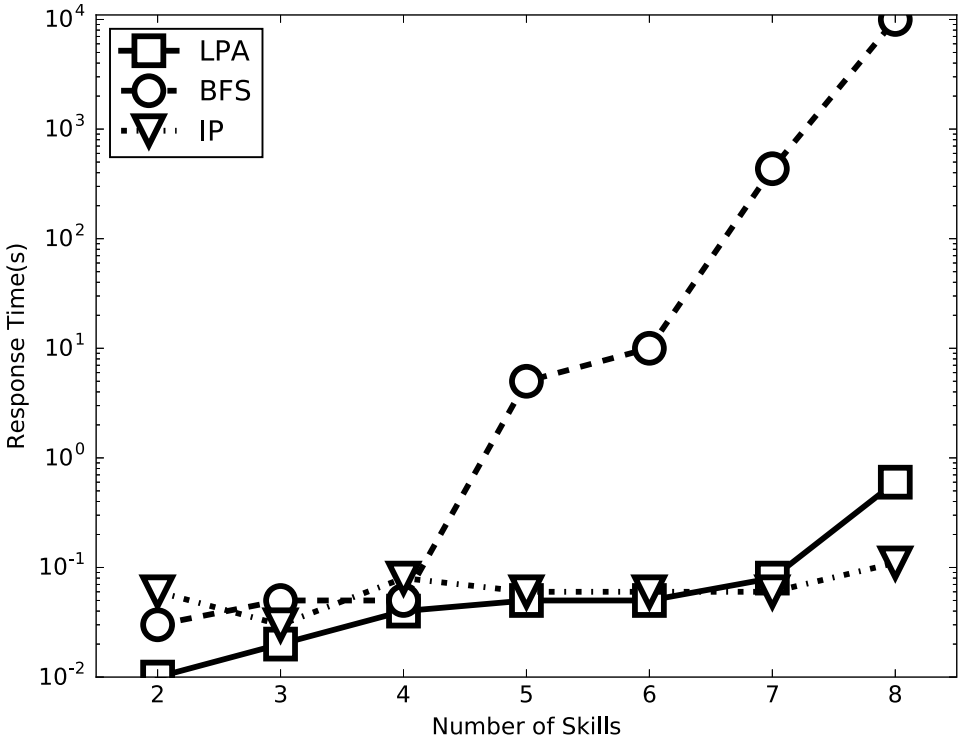

Fig 5. Performance evaluation of LPA and BFS with respect to $|s(P')|$.

https://doi.org/10.1371/journal.pone.0201596.g005

project P') and $|\mathcal{X}|$ (i.e., the number of experts) are assessed in this section. Since the scale of project differs enormously between the two real-world datasets, too small in Freelancer (each project asks for at most 5 skills) and Guru too large, we opt for SynData as our experimental data. In each experiment, we randomly select projects and merge them into a larger one P' and our evaluation focuses on P'. Then, we report the results which are plotted in Figs 5 and 6.

Fig 5 shows the corresponding response time when the number of skills required by the project varies. The number of experts is set to 100 (i.e., $|\mathcal{X}| = 100$). Obviously, LPA greatly outperforms BFS. And LPA slightly outperforms integer programming (abbreviated as IP in the figure) when the number of skills is small. From Fig 6 we also notice that LPA tends to be less radically affected by $|s(P')|$ than BFS due to the effectiveness of the pruning strategies, and IP could be scarcely influenced by the number of skills.

We alter the number of experts to compare their response time in Fig 6. Three different experimental setups are in place, $|s(P')| = 7$ for BFS and $|s(P')| = 7$ for LPA, and $|s(P')| = 7$ for integer programming (abbreviated as IP in the figure). From this figure we can see that LPA and IP are distinctly superior to BFS in every experimental setup. And LPA has the comparable time efficiency to IP when the number of skills is small. Moreover, LPA and IP are far less sensitive to $|\mathcal{X}|$ than BFS.

From the preceding comparison we can reach the conclusion that IP and LPA significantly outperforms BFS, in terms of both the runtime and the scale of the merged project and experts. When the number of skills is small, LPA have the comparable time efficiency to IP.

Performance evaluation for pcth

In this section, we evaluate the algorithms proposed for PCTH. To this end, we report the overall cost, team size, skill utilization, participation rate and response time achieved by each

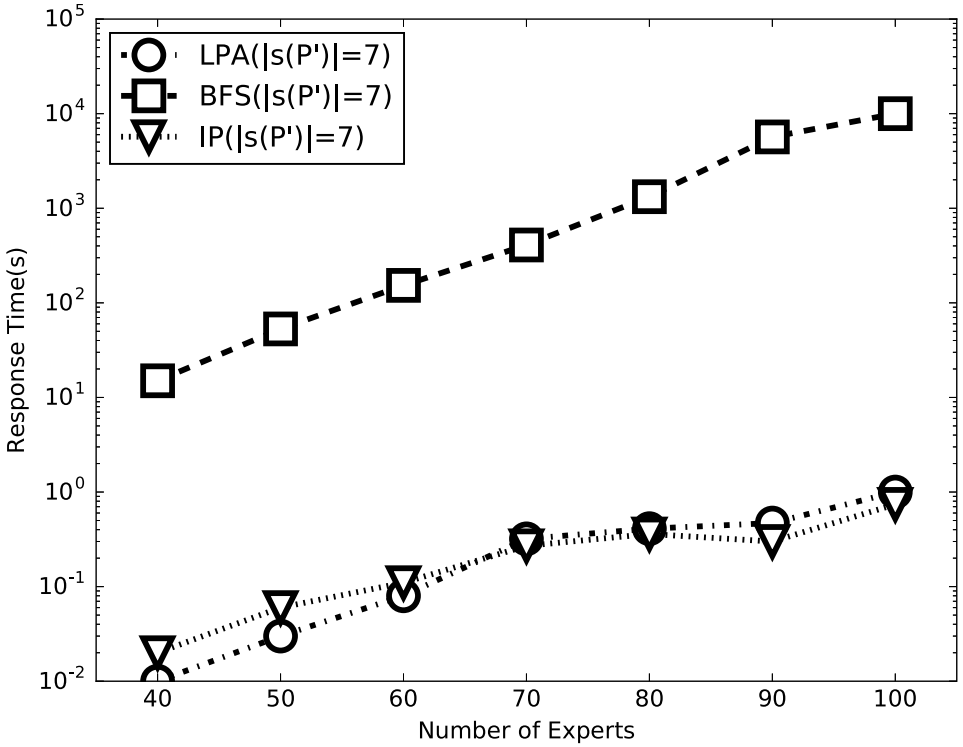

Fig 6. Performance evaluation of LPA and BFS with respect to $|X|$.

https://doi.org/10.1371/journal.pone.0201596.g006

algorithm respectively, by providing them with different amounts of projects, i.e., $|\mathcal{P}| \in \{10, 20, 30, 40, 50\}$. Apart for the three algorithms described in Section, we also employ a naive greedy heuristics algorithm called RandomExpert as an additional baseline. RandomExpert randomly selects experts, one at a time, from the space of expert set. Then it greedily assigns the expert to the projects based on the similarity of skills. The algorithm does not cease selecting experts until all the projects have been fully covered.

We carried out our experiments on three datasets: SynData, Guru and Freelancer. In each experiment, we compare the performance of the proposed algorithms. The projects are selected randomly, and for each evaluation our experiments are repeated 100 times with the average results being reported.

Cost evaluation. First we assess the cost of the team incurred by each algorithm on the three datasets. With the increase of the number of projects, the carried costs on SynData, Guru and Freelancer are ploted respectively in Figs 7, 8 and 9. It can be observed that with the increase of the number of projects, the associated costs of all the algorithms also escalate. All the algorithms except RandomExpert perform comparably which implies that there exist a multitude of skilled and cost-effective experts who can accomplish the required projects in each dataset. Therefore, no matter we concentrate on the projects (ProjectGreedy), the experts (ExpertGreedy) or both (ExpertProjectGreedy), the outcomes seem remarkably alike. RandomExpert bears higher cost than others because it disregards the expenses claimed by the experts. Additionally, from Table 4 we can find that the average number of skills per project of SynData vastly exceeds the other two datasets' while the average number of skills per expert of SynData is below its two counterparts'. Therefore, even though the average cost per expert of SynData is lower than the other two datasets', the total costs of SynData far surpass the other two datasets'.

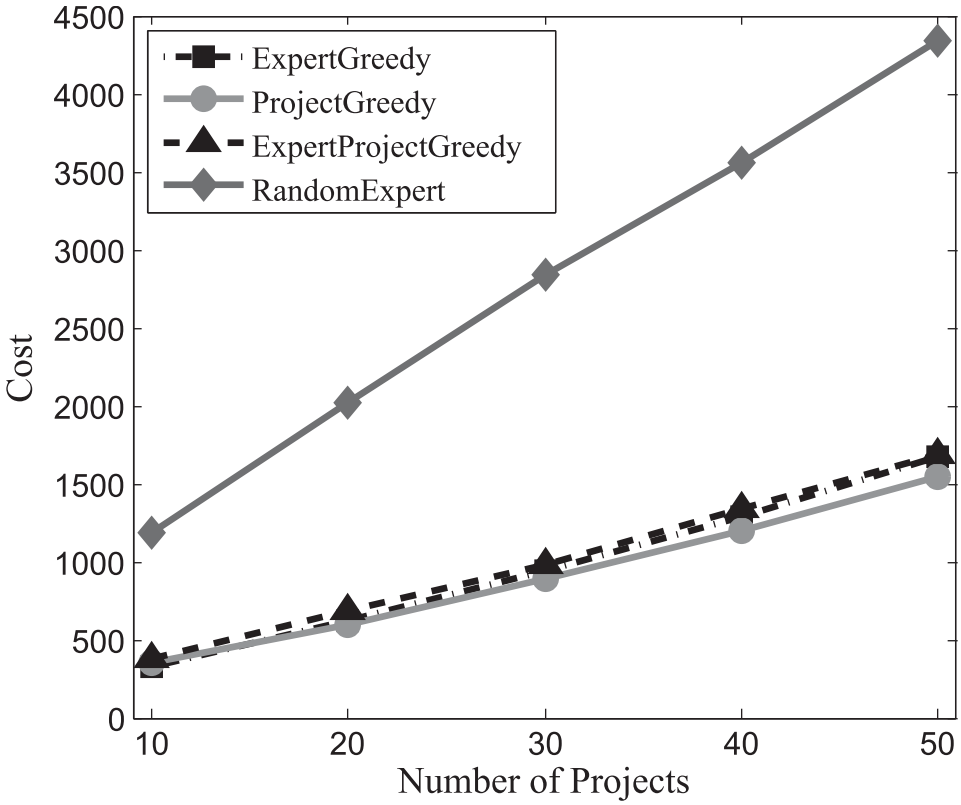

Fig 7. The evaluation of the achieved costs on SynData.

https://doi.org/10.1371/journal.pone.0201596.g007

Generally, we can draw the conclusion that when involving compensation costs, the three proposed algorithms behave similarly and are superior to the baseline.

Team size evaluation. The success of a project hinges not only on the expertise of the individuals, but also on how effectively they communicate with each other [14]. Generally speaking, the larger the team size, the harder for experts communicate with each other. Therefore, team size occupies a vital role in the success of a project. In this section, we gauge the impact of team size on our algorithms and the results on SynData, Guru and Freelancer are shown in Figs 10, 11 and 12 respectively.

As can be seen in these figures, RandomExpert are prone to assemble large teams, followed by the ExpertProjectGreedy. This can be interpreted as: when deciding on an expert, RandomExpert randomly selects an expert, and overlooks the coverage of skills and the number of projects that the expert can be assigned to simultaneously, which in turn gives rise to large teams for the projects. ExpertProjectGreedy considers the similarity between the skills of experts and projects. However, many experts in the team created by ExpertProjectGreedy may not be fully exploited, i.e., the number of projects which an expert engages in falls below his participation constraint. On the other hand, ExpertGreedy and ProjectGreedy not only take into account the skill relevance of an expert and his participation constraint but also are directed at harnessing the capabilities of experts to the fullest. That is why the team size of ExpertGreedy and ProjectGreedy are smaller than the other two algorithms'. For comparison, the size of projects of SynData is deliberately set to top the other two datasets'. It can be observed that SynData yields the largest team size regardless of other factors including the number of projects, the size of projects or the type of running algorithm on the three datasets.

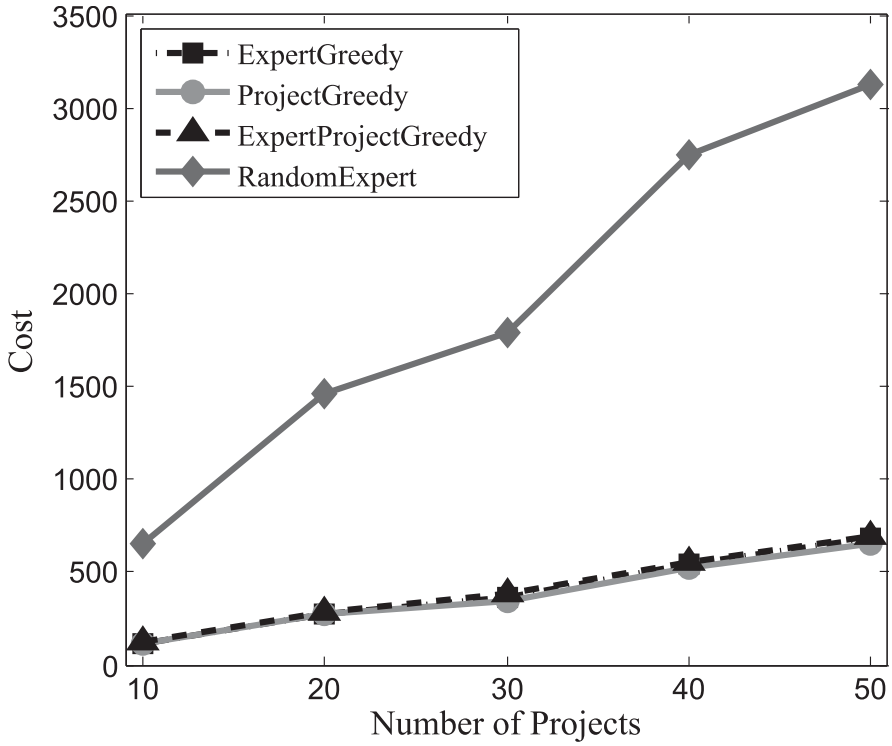

Fig 8. The evaluation of the achieved costs on Guru.

https://doi.org/10.1371/journal.pone.0201596.g008

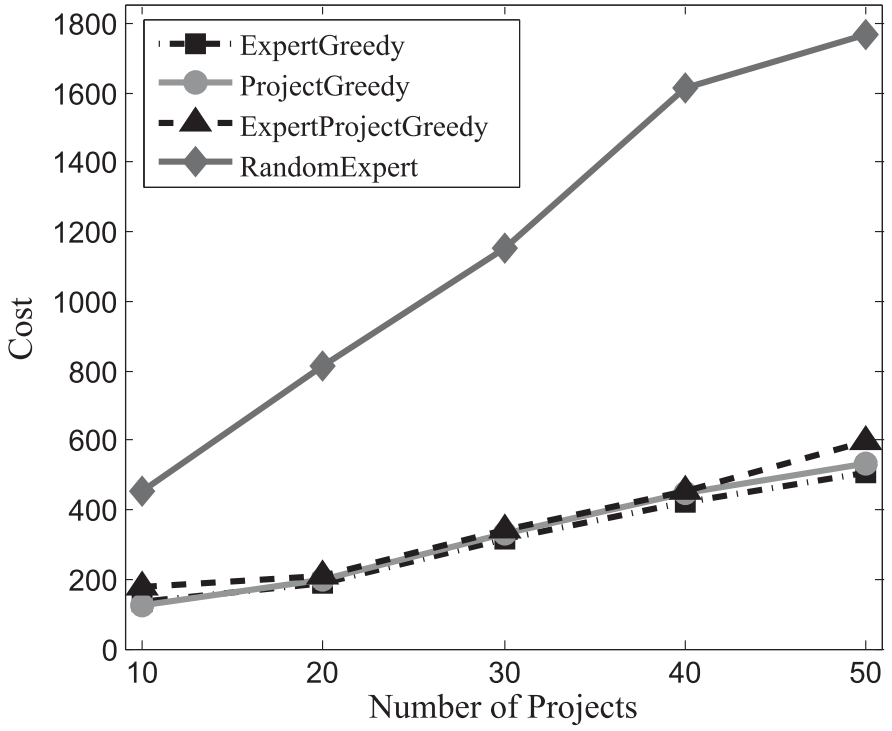

Fig 9. The evaluation of the achieved costs on Freelancer.

https://doi.org/10.1371/journal.pone.0201596.g009

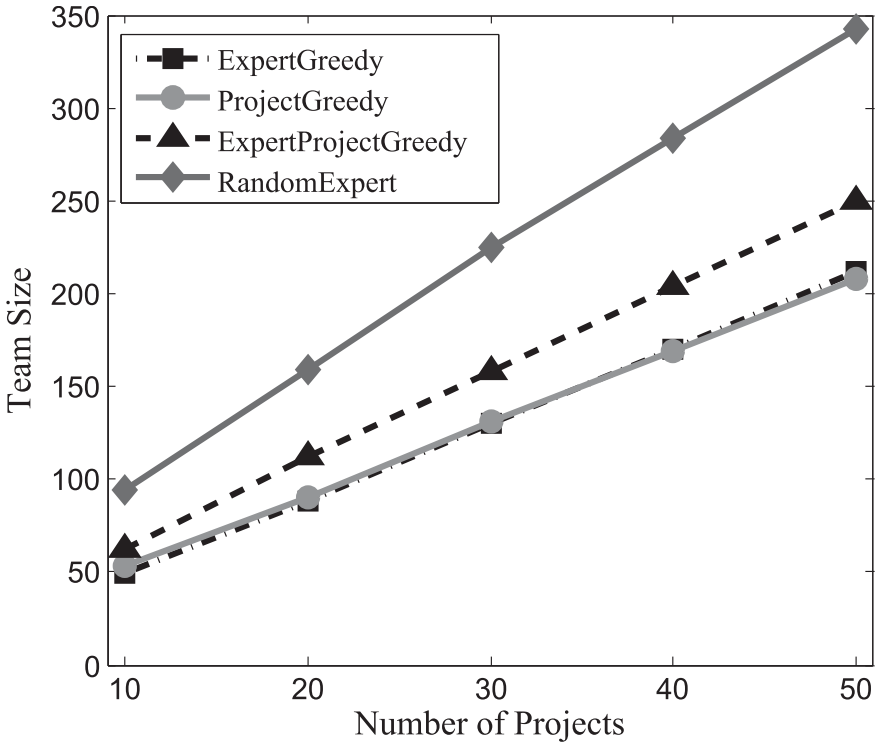

Fig 10. The evaluation of team size on SynData.

https://doi.org/10.1371/journal.pone.0201596.g010

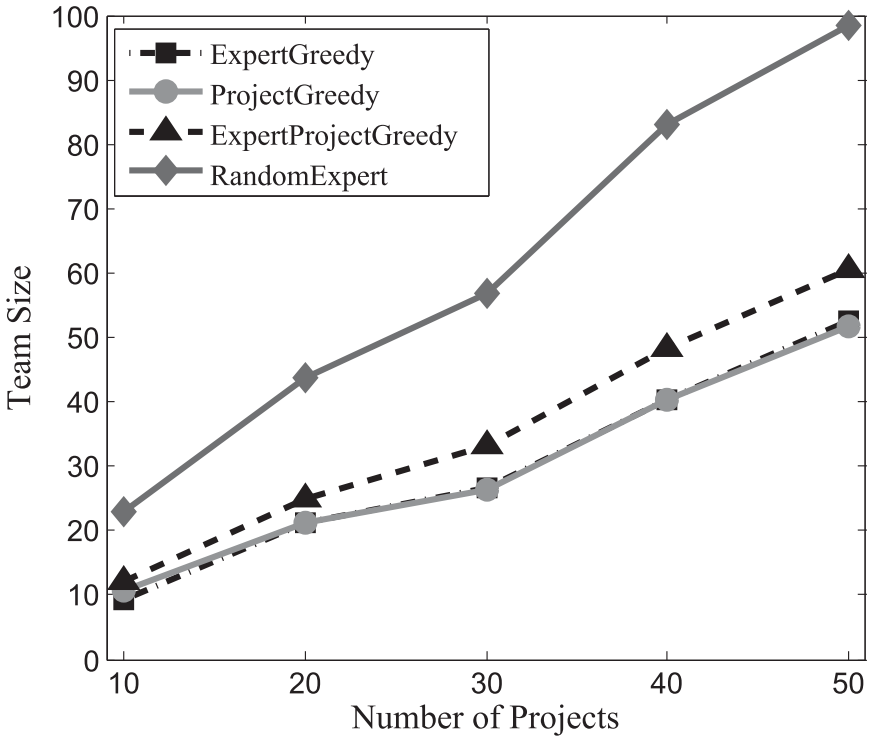

Fig 11. The evaluation of team size on Guru.

https://doi.org/10.1371/journal.pone.0201596.g011

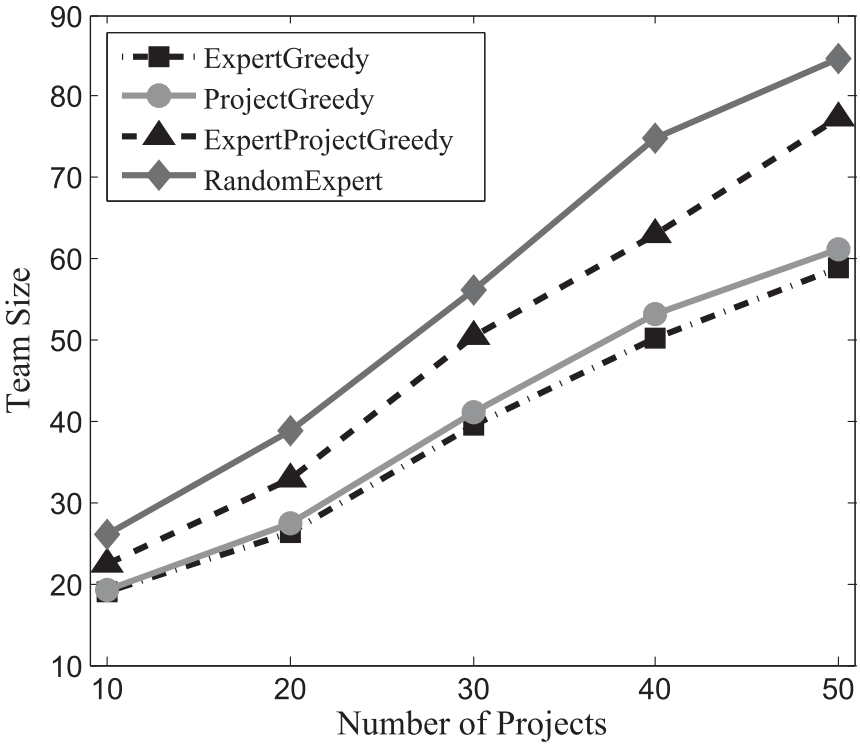

Fig 12. The evaluation of team size on Freelancer.

https://doi.org/10.1371/journal.pone.0201596.g012

Generally, we can conclude that the team size of ExpertGreedy and ProjectGreedy are smaller compared with their peers.

Skill utilization evaluation. Then we analyze skill utilization of the proposed algorithms. Given a project set $\mathcal{P}$ and a team of experts $\mathcal{T}$ that can perform the projects, the skill utilization ψ is defined as follows:

$$\psi = \frac{\sum_{p_j \in \mathcal{P}} |s(p_j)|}{\sum_{x_i \in \mathcal{T}} |s(x_i)| \cdot w(x_i)}, \tag{7}$$

where the numerator returns the number of skills required by projects (i.e., the number of skills experts utilized), and the denominator denotes the sum of the number of experts' skills, with the participation constraint being considered. Obviously, it reflects the ratio of skill utilization.

The results attained by each algorithm on SynData, Guru and Freelancer are shown in Figs 13, 14 and 15 respectively. From the figures, it is noteworthy that ExpertProjectGreedy fares much better than the others regarding skill utilization. In fact, ExpertProjectGreedy accomplishes this through two measures. First, it greedily selects expert-project match pairs with respect to their similarity of skills, which manages to exploit the skills of the experts to the most possible extent. Second, after a match pair (x_i, p_j) has been selected, the cost of x_i plunges to 1 (see Algorithm 4) indicating a strong likelihood that the expert will be chosen later.

Note that skill utilization declines with the increase of the number of projects for Guru, which contrasts starkly with the other two datasets. Recall that in Table 4 the average number

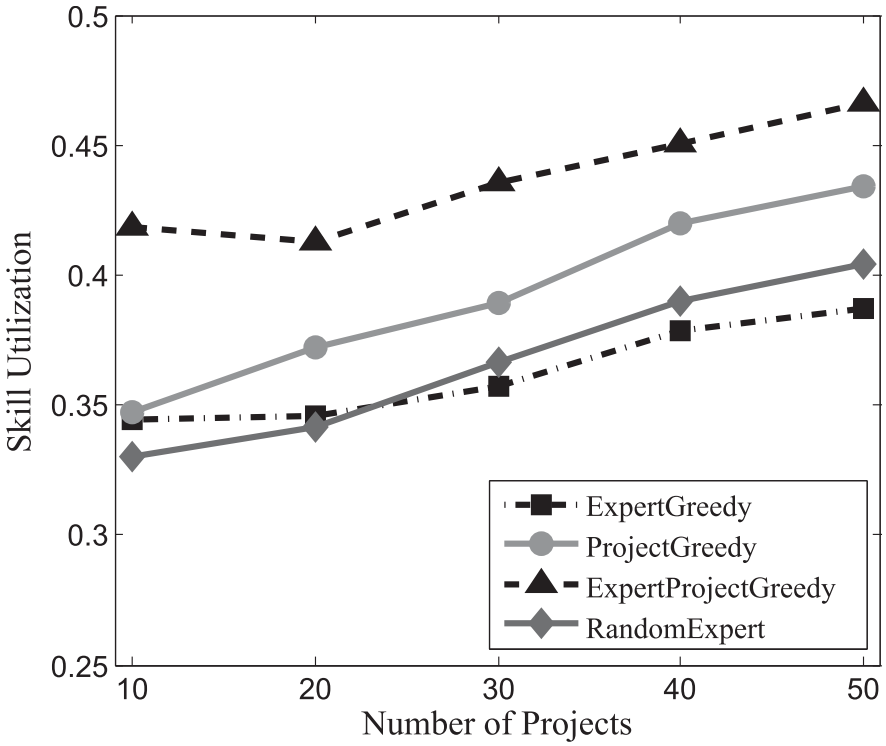

Fig 13. The evaluation of the skill utilization on SynData.

https://doi.org/10.1371/journal.pone.0201596.g013

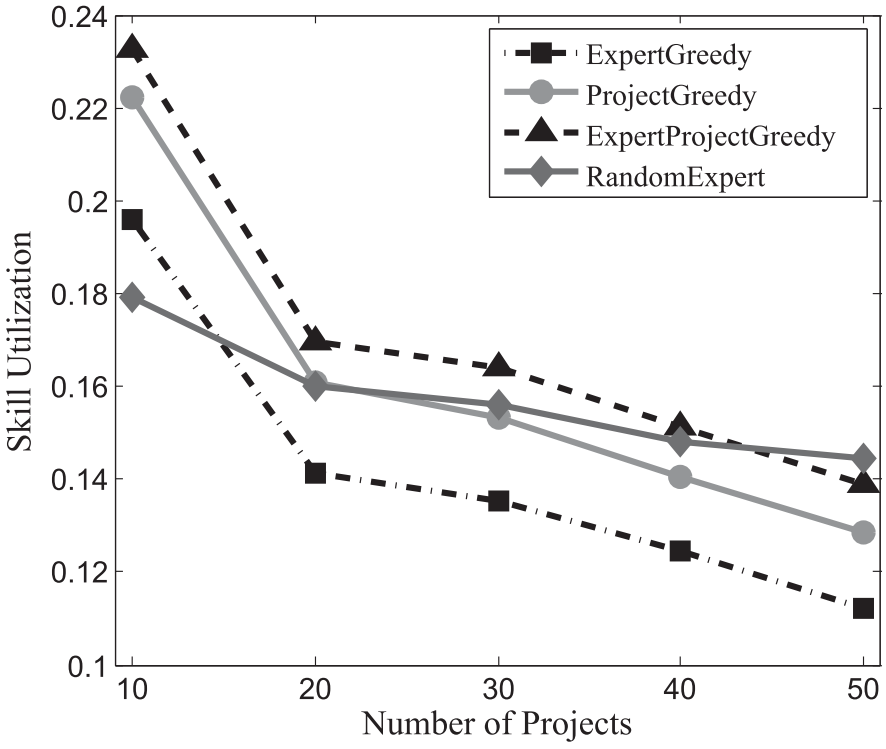

Fig 14. The evaluation of the skill utilization on Guru.

https://doi.org/10.1371/journal.pone.0201596.g014

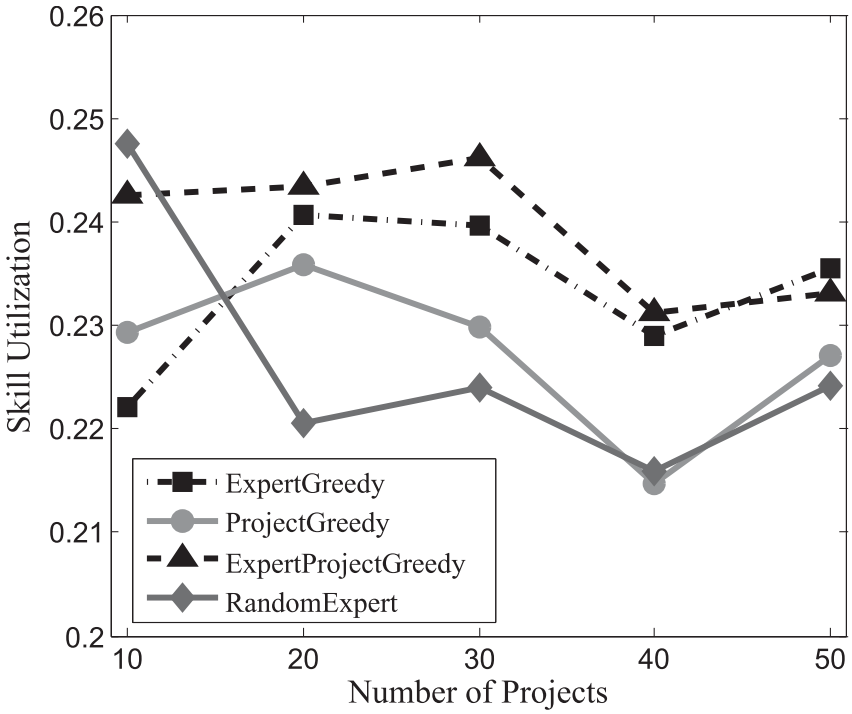

Fig 15. The evaluation of the skill utilization on Freelancer.

https://doi.org/10.1371/journal.pone.0201596.g015

of skills of expert more than doubles that of project for Guru, suggesting that the increase of team members can amplify the effect that more irrelevant skills reduce the ratio of utilization. As was discussed earlier, team size is determined by both the number of projects and the skills of experts. Thus, it makes sense to see that the ratio of skill utilization drops with the increase of the quantity of projects for Guru.

On the whole, the skill utilization attained by ExpertProjectGreedy is superior to the others'.

Participation rate evaluation. Since the participation constraint of experts is an essential condition in our problem, we examine the participation rate of the teams formed by all the algorithms. The participation rate β is defined as follows:

$$\beta = \frac{\sum_{p_j \in P} |com(p_j)|}{\sum_{x_i \in T} w(x_i)}. \tag{8}$$

The denominator of the fraction represents the sum of $w(x_i)$ of each team members x_i, and the numerator gives the sum of the number of projects each team member is involved in. Obviously, the value of this fraction ranges from 0 to 1. The number of projects that experts engage in is maximized when the participation rate β reaches 1. Conversely, $\beta = 0$ indicates that no expert is assigned to any project.

Figs 16, 17 and 18 depict the participation rates characterizing four algorithms on SynData, Guru and Freelancer. A consistent trend emerges followed by all the algorithms on different datasets that positive correlation between the participation rate β and the amount of projects can be identified. This can be primarily ascribed to the fact that more projects bring about larger skill sets which allow for more possibilities for experts to take on different jobs in parallel. From these figures, we can also observe that ExpertGreedy and ProjectGreedy all perform better than ExpertProjectGreedy because they employ fairly distinctive greedy strategies. In

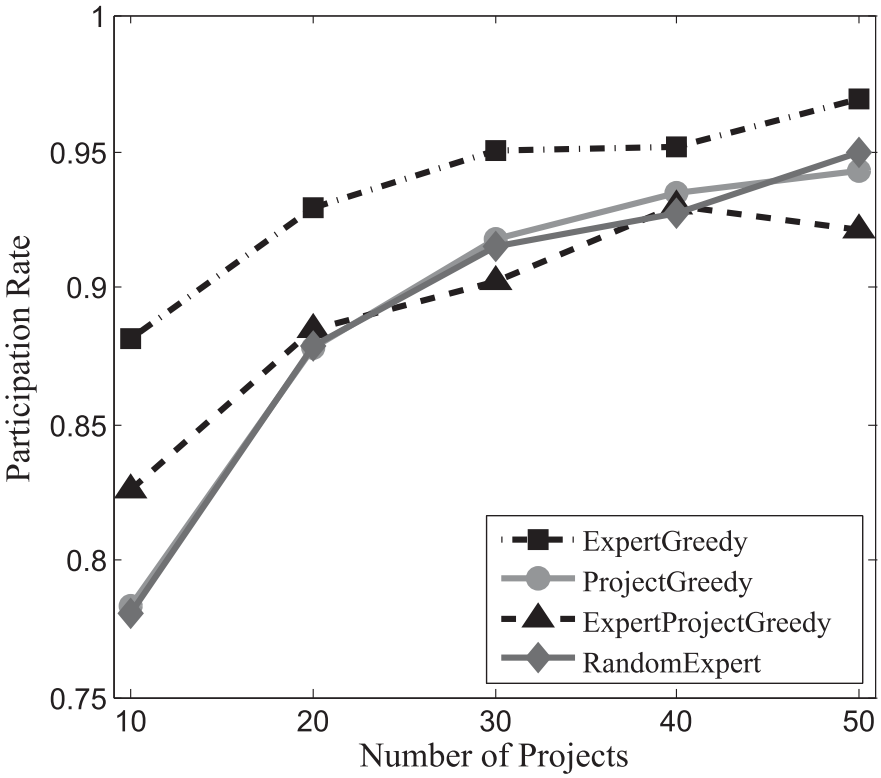

Fig 16. The evaluation of participation rate on SynData.

https://doi.org/10.1371/journal.pone.0201596.g016

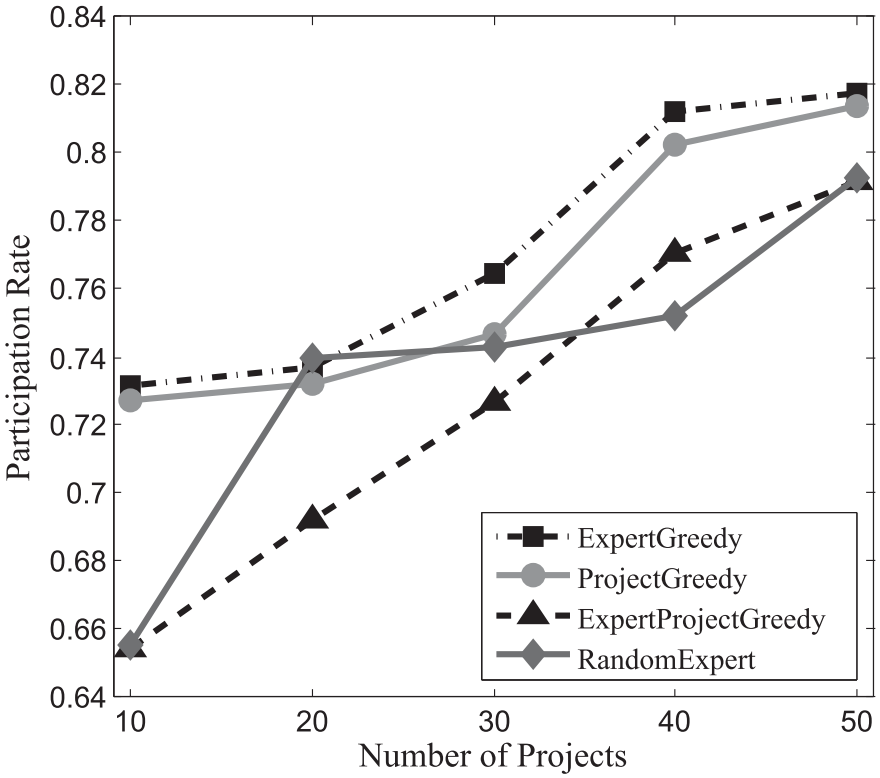

Fig 17. The evaluation of participation rate on Guru.

https://doi.org/10.1371/journal.pone.0201596.g017

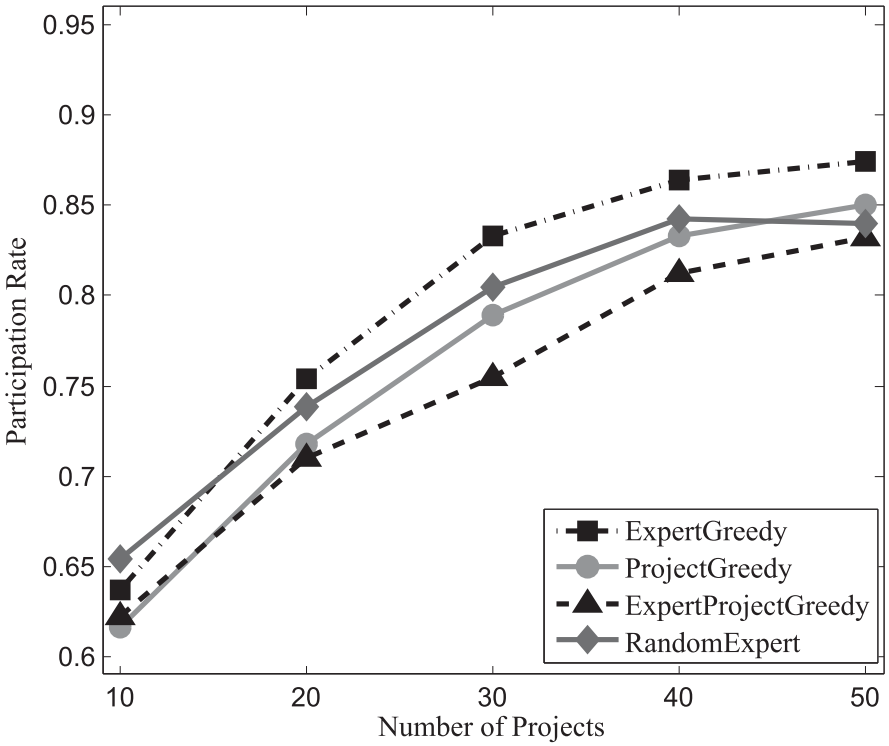

Fig 18. The evaluation of participation rate on Freelancer.

https://doi.org/10.1371/journal.pone.0201596.g018

ExpertGreedy, experts are assigned to projects before his participation constraint falls to zero or simply he cannot fulfill the duties of the remaining projects. Similarly in ProjectGreedy, an expert can still engage in other projects after he has already been selected for one job, provided his participation constraint will not be violated. ExpertProjectGreedy which merely concentrates on the best match pair in every iteration differs substantially from the preceding two alternatives. Furthermore, ExpertGreedy holds a narrow lead over ProjectGreedy, which can be ascribed to the fact that in every iteration we always opt for the expert with the most pertinent skills so he in turn will be more inclined to join in other projects at the same time. Although ProjectGreedy approaches this problem from the perspective of one specific project, the skillset of a chosen expert still bears the strongest resemblance to that of the project.

Hence we can arrive at the conclusion that ExpertGreedy surpasses the other 3 algorithms in terms of participation rate.

Response time evaluation. In order to investigate the efficiency of the algorithms, we continue to conduct experiments on the three datasets, and the experimental results are displayed in Figs 19, 20 and 21. From these figures, we can observe that the response time of both ProjectGreedy and RandomExpert barely rises, vastly outperforming the other two from beginning to end. This can be explained by the fact that the two algorithms iterate fewer times on the space of expert set than ExpertGreedy or ExpertProjectGreedy does. In all the three datasets, the number of experts considerably surpasses that of projects. Therefore, iterating too many times on the space of experts will consume more time. Specifically, ProjectGreedy tends to select experts in a way that each project can be performed one by one. With the iteration progressing, the number of the remaining projects drops fast, and as a consequence, so does that of iteration on the space of experts. This differentiates

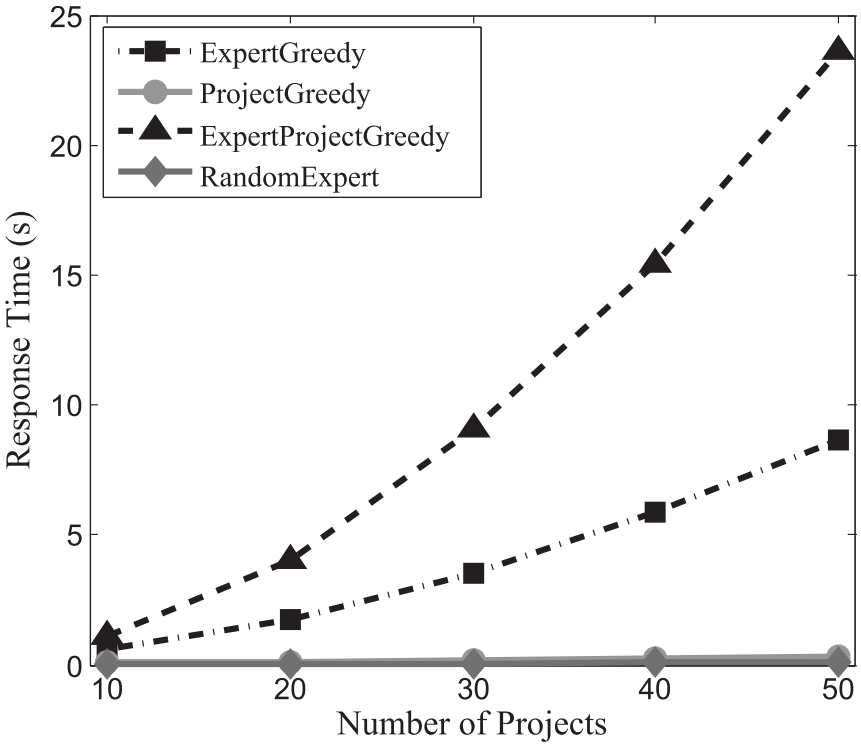

Fig 19. The evaluation of response time on SynData.

https://doi.org/10.1371/journal.pone.0201596.g019

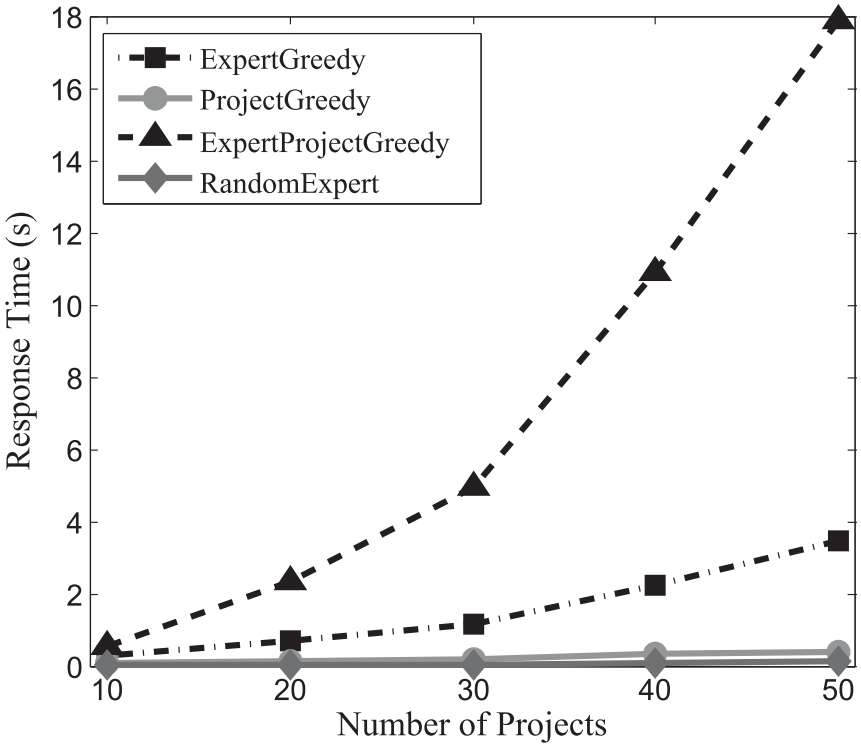

Fig 20. The evaluation of response time on Guru.

https://doi.org/10.1371/journal.pone.0201596.g020

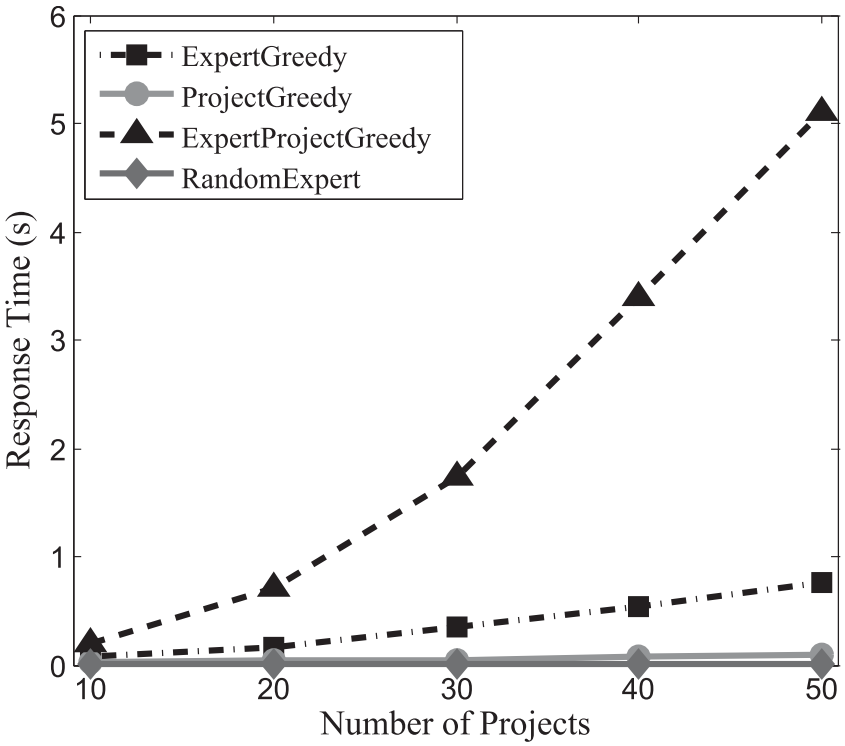

Fig 21. The evaluation of response time Freelancer.

https://doi.org/10.1371/journal.pone.0201596.g021

ProjectGreedy from ExpertGreedy which treats the projects as a whole when deciding on an expert. Therefore, ProjectGreedy outperforms ExpertGreedy regarding response time under the same circumstances. Additionally, ExpertProjectGreedy iterates too many times on the space of $\mathcal{X} \times \mathcal{P}$ which apparently entails more time than the others. For this reason, as can be observed from the figures, ExpertProjectGreedy manifests greater susceptibility to the number of projects than the others.

Generally speaking, RandomExpert and ProjectGreedy are the most efficient algorithms among the four.

Conclusions

In this paper, we proposed formalizations and algorithmic solutions for the participation constrained team hire problem (PCTH), where the goal is to hire a team of participation-constrained experts to complete all given projects such that the total cost is minimized. This is the first work to investigate the participation constrained team hire problem. We studied a special case of PCTH and introduced an efficient algorithm that identifies an exact solution for it. For the general PCTH, we proved that it is NP-hard and presented three algorithms. In a thorough experimental evaluation, we appraised the performance of our algorithms, and compared them with reasonable baseline approaches. We conclude that our algorithms on both synthetic and real datasets outperform the baseline algorithms significantly. In the future, we will embark on exploring how the preferences of experts regarding projects can shape this issue. That is, we would like to consider the scenario when an expert explicitly expresses his intense interest for a particular project, which will certainly serve as a vital factor in assigning experts to jobs.

Supporting information

S1 Dataset. Freelancer.
(ZIP)

S2 Dataset. Guru.
(ZIP)

S3 Dataset. SynData.
(ZIP)

Author Contributions

Conceptualization: Liang He.

Data curation: Ke Liu, Mengjie Wan.

Formal analysis: Heli Sun, Jianbin Huang, Xiaolin Jia.

Funding acquisition: Heli Sun.

Methodology: Heli Sun, Jianbin Huang.

Project administration: Ke Liu.

Software: Ke Liu, Mengjie Wan, Yu Zhou, Chen Cao.

Supervision: Heli Sun, Jianbin Huang.

Validation: Ke Liu, Mengjie Wan.

Visualization: Mengjie Wan.

Writing – original draft: Heli Sun, Ke Liu, Mengjie Wan.

Writing – review & editing: Heli Sun, Xiaolin Jia, Liang He.

References

1. Fitzpatrick EL, Askin RG. Forming effective worker teams with multi-functional skill requirements. Computers & Industrial Engineering. 2005; 48(3): 593–608. https://doi.org/10.1016/j.cie.2004.12.014
2. Huang JB, Sun XJ, Zhou Y, Sun HL. A team formation model with personnel work hours and project workload quantified. The Computer Journal, 2017; 60(9) pp. 1382–1394. https://doi.org/10.1093/comjnl/bxx009
3. Nuria RA, Agnieszka P, Sabine S, Roger G, Marta SP. Leader Evaluation and Team Cohesiveness in the team development: a matter of gender? Plos One. 2017; 12(10): 1–20.
4. Zhou Y, Huang JB, Jia XL, Sun HL. On Participation Constrained Team Formation. Journal of Computer Science and Technology. 2017; 32(1): 1–16. https://doi.org/10.1007/s11390-017-1710-6
5. Huang JB, Lv Z, Zhou Y, Li H, Sun HL, Jia XL. Forming Grouped Teams with efficient collaboration in social networks. The Computer Journal, 2017; 60(11) pp. 1545–1560.
6. Zhou Y, Huang JB, Sun HL, Jia XL. Nonredundant Cost-Constrained Team Formation. International Journal of Data Warehousing and Mining. 2017; 13(3): 25–46. https://doi.org/10.4018/IJDWM.2017070102
7. Kargar M, An A. TeamExp: Top-k Team Formation in Social Networks. Proceedings of the 11th IEEE international conference on Data Mining Workshops. 2011; pp. 1231-1234.
8. Zzkarian A, Kusiak A. Forming teams: an analytical approach. IIE Transactions. 2011; 31(1): 85–97. https://doi.org/10.1080/07408179908969808
9. Liu XJ, He Q, Tian YY, Lee WC, McPherson J, Han JW. Event-based social networks: linking the online and offline social worlds. Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. 2012; pp. 1032-1040.

10. Feng KY, Cong G, Bhowmick SS, Ma S. In search of influential event organizers in online social networks. Proceedings of the 2014 ACM SIGMOD international conference on Management of data. 2014; pp. 63-74.
11. Li KQ, Lu W, Bhagat S, Lakshanan LV, Yu C. On Social Event Organization. Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. 2014; pp. 1206-1215.
12. Huang JB, Zhou Y, Jia XL, Sun HL. A Novel Social Event Organization Approach for Diverse User Choices. The Computer Journal. 2017; 60(7): 1078–1095.
13. Golshan B, Lappas T, Terzi E. Profit-maximizing cluster hires. Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. 2014; pp. 1196-1205.
14. Lappas T, Liu K, Terzi E. Finding a team of experts in social networks. Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, 2009; pp. 938–950.
15. Kargar M, Zihayat M, An A. Finding Affordable and Collaborative Teams from a network of experts. Proceedings of the 13th SIAM International Conference on Data Mining. 2013; pp. 587-595.
16. Majumder A, Datta S, Naidu K. Capacitated team formation problem on social networks. Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining. 2012; pp. 1005-1013.
17. Anagnostopoulos A, Becchetti L, Carlos C, Gionis A, Leonardi S. Power in unity: forming teams in large-scale community systems. Proceedings of the 19th ACM international conference on Information and knowledge management. 2010; pp. 599-608.
18. Kargar M, An A. Discovering top-k teams of experts with/without a leader in social networks. Proceedings of the 20th ACM international conference on Information and knowledge management. 2011; pp. 985-994.
19. Dorn C, Dustdar S. Composing near-optimal expert teams: A trade-off between skills and connectivity. On the Move to Meaningful Internet Systems. 2010; pp. 472–489. https://doi.org/10.1007/978-3-642-16934-2_35
20. Li CT, Shan MK. Team Formation for generalized tasks in expertise social networks. Proceedings of the IEEE international conference on Social Computing. 2010; pp. 9-16.
21. Gajewar A, Sarma AD. Multi-skill Collaborative Teams based densest subgraphs. Computer Science. 2011;abs/1102.3340: 1077-1088.
22. Anagnostopoulos A, Becchetti L, Castillo C, Gionis A, Leonardi S. Online team formation in social networks. Proceedings of the 21st international conference on World Wide Web. 2012; pp. 839-848.
23. Lund C, Yannakakis M. On the hardness of approximating minimization problems. Journal of the ACM. 1994; 41(5): 960–981. https://doi.org/10.1145/185675.306789
24. Feige U. A threshold of ln n for approximating set cover. Journal of the ACM. 1998; 45(4): 634–652. https://doi.org/10.1145/285055.285059
25. Clarkson KL, Varadarajan K. Improved approximation algorithms for geometric set cover. Discrete & Computational Geometry. 2007; 37(1): 43–58. https://doi.org/10.1007/s00454-006-1273-8
26. Noga A, Baruch A, Yossi A, Niv B, Joseph SN. The online set cover problem. Siam Journal on Computing. 2009; 39(2): 361–370. https://doi.org/10.1137/060661946
27. Bar-Yehuda R, Even S. A linear-time approximation algorithm for the weighted vertex cover problem. Journal of Algorithms. 1981; 2(2): 198–203. https://doi.org/10.1016/0196-6774(81)90020-1
28. Guo J, Niedermeier R. Exact algorithms and applications for tree-like weighted set cover. Journal of Discrete Algorithms. 2006; 4(4): 608–622. https://doi.org/10.1016/j.jda.2005.07.005
29. Klein P, Ravi R. A nearly best-possible approxiamtion algorithm for node-weighted Steiner trees. Journal of Algorithms. 1995; 19(1): 104–115. https://doi.org/10.1006/jagm.1995.1029
30. Varadarajan K. Weighted geometric set cover via quasi-uniform sampling. Proceedings of the forty-second ACM symposium on Theory of computing. 2010; pp. 641-648.
31. Hua QS, Yu DX, Lau F, Wang YX. Exact algorithms for set multicover and multiset multicover problems. Algorithms and Computation. 2009; pp. 34–44. https://doi.org/10.1007/978-3-642-10631-6_6
32. Chuzhoy J, Naor J. Covering problems with hard capacities. Siam Journal of Computing. 2006; 36(2): 498–515. https://doi.org/10.1137/S0097539703422479
33. Chekuri C, Clarkson KL, Har-Peled S. On the set multicover problem in geometric settings. ACM Transactions on Algorithms. 2012; 9(1) 1–9.
34. Sridhar R, Vazirani Vijay V. Primal-dual rnc approximation algorithms for (multi)-set (multi)-cover and covering integer programs. Siam Journal on Computing. 1993; pp. 322–331.

35. Dumais ST, Nielsen J. Automating the assignment of submitted manuscripts to reviewers. Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval. 1992; pp. 233-244.

36. Karimzadehgan M, Zhai CX, Belford G. Multi-aspect expertise matching for review assignment. Proceedings of the 17th ACM conference on Information and knowledge management. 2008; pp. 1113-1122.

37. Kou NM, Hou UL, Mamoulis N, Gong ZG. Weighted Coverage based Reviewer Assignment. Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. 2015; pp. 2031-2046.

38. Mimno D, McCallum A. Automating the assignment of submitted manuscripts to reviewers. Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. 2007; pp. 500-509.

39. Vazirani Vijay V. Approximation Algorithm. Springer, Berlin. 2001.

40. Agrawal R, Srikant R. Fast algorithms for mining association rules in Large Databases. Proceedings of the 20th International Conference on Very Large Data Bases. 1994; pp. 487-499.