Aalto University
School of Science
Degree Programme of Computer Science and Engineering

Darshan Mallenahalli Shankaralingappa

Outsource or Train: A team formation problem

Master's Thesis
Espoo, November 16, 2016

Supervisor: Professor Aristides Gionis, Aalto University

Aalto University
School of Science
Degree Programme of Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Darshan Mallenahalli Shankaralingappa

Title:
Outsource or Train: A team formation problem

Date:	November 16, 2016	**Pages:**	55
Professorship:	Data Mining	**Code:**	T-110
Supervisor:	Professor Aristides Gionis		

Many large projects in organizations need to be broken down into manageable tasks for their completion, yet such tasks still demand a diverse set of skills to be completed. Team formation is a way of acquiring workers with different set of skills to complete the tasks and minimizing the cost of assigned workers for the benefit of the organization. It is also found that workers work effectively when they are happy with the task they are working on.

In this project, we are given a set of tasks that need to be completed and a set of workers who can be assigned to the tasks. Each task will require a set of skills to be completed. Each worker will possess a set of skills. Each worker has some cost of working on a task. This cost can also be seen as a dissatisfaction factor. And this cost might be different for different tasks. We want to find an assignment of workers to tasks, while making sure that all the skills required to complete a task are covered. In addition to that, we also have to minimize the cost (dissatisfaction) of the workers assigned to the tasks.

In real world applications, not all tasks can be completely covered by the available workers. This is because, not all the required skills to complete the tasks are possessed by the workers. In this project, we propose two approaches to overcome these downfalls. One approach is to outsource the entire task for a cheaper cost. And the other approach is to train the workers. We provide algorithmic solutions for both the approaches.

We first prove that this team formation problem is NP-Complete. And then we propose and analyze different algorithms for both the approaches. These algorithms are inspired from solutions to matching and set cover problems.

We used the data from stackexchange Q & A discussion forum and bibsonomy social bookmarking and publication-sharing website to model workers and tasks for our experiments. From the results we found that the difference in the performance of the algorithms was very little and almost all algorithms gave good results. In the end, we also propose some future work that can be considered for interested readers.

Keywords: data mining, algorithms, team formation, task assignment

Language: English

Acknowledgements

First and for most, I would like to express my deep gratitude to my supervisor Professor Aristides Gionis, for giving me the opportunity to work on this project that I present in this thesis.

I further wish to extend my gratitude to Professor Evimaria Terzi from Boston University for her collaboration, and for defining the problem in the field of team formation that is the motivation of the work presented in this thesis.

I also wish to thank my office mate, Abdulmelik Mohammed for giving me useful insights while formulating the primal dual problems and with the writing of this thesis.

I am grateful for the assistance and support given by whole technical staff at the Department of Information and Computer Science. I also acknowledge the use of computational resources provided by Aalto Science-IT project and the resources provided by CSC for the experiments carried out in this project.

Last but not least, I wish to express my deepest gratitude to my parents and my brother Uday for supporting and nurturing me over the past 25 years. I dedicate this thesis to my beloved mom Shanthamma, my dearest father Shankaralingappa and my cute little niece Rituparna Uday.

Espoo, November 16, 2016

Darshan Mallenahalli Shankaralingappa

Contents

Chapter 1

Introduction

When organizations formulate new projects, the projects are broken down into smaller tasks. And each task would require a set of diverse skills to be completed. To complete each task, the workers are hired groups, since the number of skills required to complete a task is way more than the number of skills an individual worker possesses. There are many settings with this problem. Like a worker can be part of many groups and one group can be used to complete multiple tasks. In this project, we are concentrating on a setting where one group will be used to complete a task and a worker can be part of only one group. From an organization perspective, the hiring must be done in a cost effective manner. And from a worker persective, he should be least dis satisfied with the task he is working on. Notice that, we are trying to address two different problems here. One is the assignment of the workers to tasks called as the task assignment problem. And the other one is the coverage of all the skills required to complete a task. This is called the set cover problem.

Task assignment problem is a very well researched problem in the field of operations research. Let us, consider one of the formulations of the problem from seminal work of Graham [1]: given a set of tasks, a set of workers and a cost function of assigning a worker to a task, the problem is to find the best assignment of workers to tasks, such that the total cost is minimized. There are different variations of the problem where a single worker can be assigned to multiple tasks and multiple workers can be assigned to a single task. For this project, let us consider that a worker can be assigned to only one task. We can look at this problem as each worker with a single skill and each task requires only one skill to be completed. This problem statement can be mapped to the minimum cost bipartite matching. If a worker is interested in being part of a task, then there will be a edge between the task and the worker with a cost. Otherwise, there will be no edge. A task assignment

problem without weights is illustrated in Figure 1.1. In the figure, there are a set of workers and a set of tasks. The black lines indicate that workers are interested are working on the tasks and red lines show the assignments of workers to tasks. The solution to this problem can be found by making use of the hungarian method [2][3]. Note that the hungarian method, requires that the number of tasks and number of workers have to be same. This can be overcome by padding of zero values.

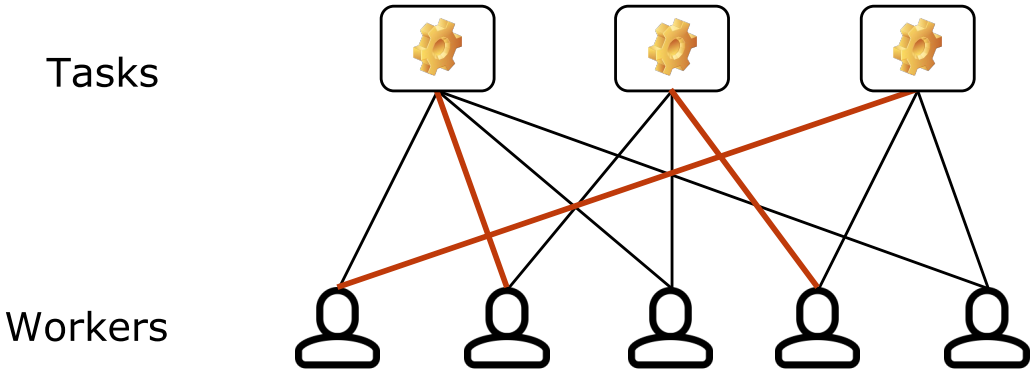


Figure 1.1: Illustration of a Task Assignment problem

Let us now consider a new problem where there is one complex task and this complex task requires a set of skills to complete. We will be given a set of workers and each worker possesses a subset of skills required to complete the complex task. And each worker will have a cost of working on the cost. The problem is to find the subset of workers from the available pool of workers, such that all the skills required to complete the complex task is possessed by this subset of workers. And making sure that the total cost is minimized. An unweighed version of the problem is illustrated in Figure 1.2. The skills required to complete the task is at the top of the figure and the skills possessed by the workers are shown below the each worker. This problem can be mapped to the minimum set cover problem. This is a well studied problem which can be solved by greedy algorithm or a primal dual method among others.

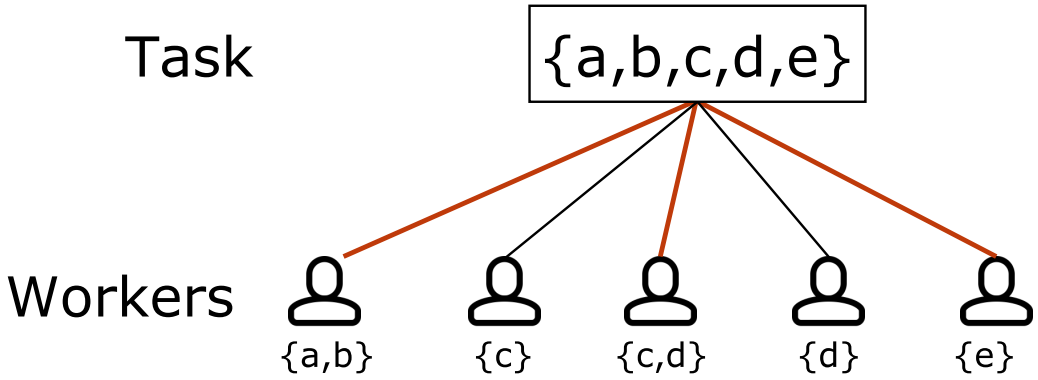


Figure 1.2: Illustration of a completing a complex task with a set of skills

In this project, we are combining both the above problem definitions. We will be a set of tasks and a pool of workers. Each task will have a set of skills to be covered. Each worker will have a set of skills that can be used to partly cover the skills of a task. The worker will also have some preference on which task they want to work on i.e., a cost of working on a task t. We want to complete all the complex tasks by covering all the skills by assigning the workers to tasks. And the total cost is minimized. A task can have more than one workers assigned to it, but a worker can be assigned to atmost one task. An unweighed version is illustrated in Figure 1.3. The skills required to complete the task is at the top of each task and the skills possessed by the workers are shown below each worker. Our solutions are inspired by the solutions to minimum cost bipartite matching and minimum set cover problems.

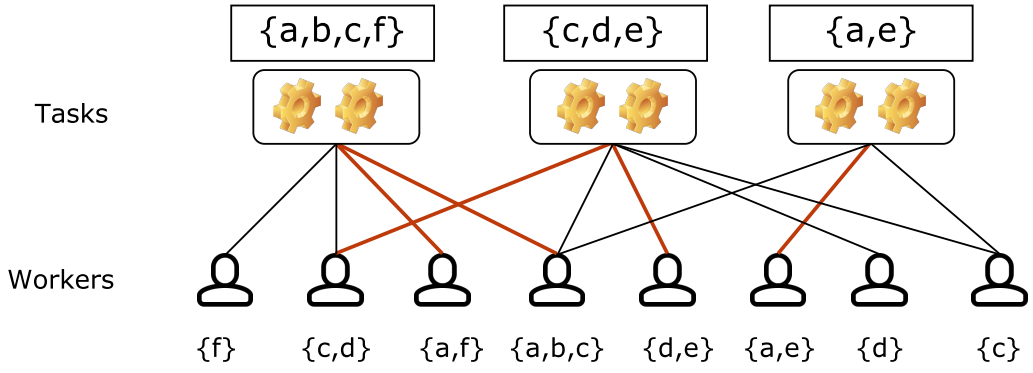


Figure 1.3: Illustration of team formation problem

Some times in real world applications, it is not possible to cover all the

tasks. In that case, it is good to outsource the whole project or hire the workers and train them on the skills that are not covered. This was the technical overview of the thesis. In the next section, we will be discussing the use cases of this project. In the last setion of this chapter, we will discuss some related papers.

In the next chapter, we will formally define the problem and prove that our problem is NP complete. In the third chapter, we will discuss the different algorithms. After that, we will see how the dataset is prepared and also the experiments to see the performance of the algorithms.

1.1 Motivation

In this section, we will see some of the real world application of this project. Imagine a big IT company receives a contract to complete a set of new projects. These projects require some skills to be covered. Then a set of workers apply for the job. When the workers apply, they also specify their interests on working on a project. The organization will would want to hire and assign the workers to the new projects. When assigning these workers to the projects, the organization has make sure that all the workers assigned to the projects are less dis-satisfied as possible. If there are not enough workers to cover all the skills of all the tasks, the whole project can be outsourced to another company or the organization can hire the workers and then train the workers on the skills that they don't possess. This way, we can cover all the skills of all the tasks.

In our next example, let us consider an imaginary startup "MSD". The co-founders of MSD, Uday and Darshan identify that 2 tasks frontend and backend to complete their initial product.

```
frontend:  {css,javascript,html,flash,jquery}
backend: {java,mysql,apache tomcat,JSF}
```

Then, they receive 5 applications for these tasks.

```
Romeo: {css,javascript,jquery}
Ali: {html,css,javascript}
John: {html,flash,css}
Jane: {java,mysql}
Rose: {java,JSF}
```

The co-founders will hire and assign Romeo and Ali to frontend task. Then they notice that they cannot complete the second task with the applicants present. Since apache tomcat is not possessed by any of the applicants,

hence the co-founders can choose to outsource the whole task. Or hire Jane and Rose for the second task and train one of the hired workers on apache tomcat.

1.2 Related Work

This work is clearly related to other task assignment problems and team formation problems. In this section we give an overview of all such projects.

The work from Anagnostopoulos et al.[4] is very closely related to our project. They aim to create one team for each project, such that over time, the maximum number of teams that each expert participates in is minimized. They consider that a worker can be part of more than one task. And they also assume that the tasks will arrive in streams (online version). Their problem is defined as follows: for every task T_t, find a team Q_t that has all the required skills for T_t, minimizing the workload of the person. In this problem, one team can be assigned to multiple tasks and tasks arrive in streams. These are the two main differences with our problem.

Another closely related work is by the work by Golshan et al.[5], they want to find one team that can complete as many tasks as possible. They assume a collection of tasks P, where each task requires a certain set of skills to complete. Then a pool of experts X is given, where each expert has his own skillset and cost for working. Their problem is of hiring a team of experts $T \subseteq X$, so that the overall cost does not exceed a given budget B, and also making sure that the total benefit of the tasks that this team can collectively cover is maximized. In our problem, we assign the worker to only one task and we don't have a given budget.

In refereed conferences and journals, every submission will be reviewed by many members in the committee. Automated systems such as Easychair, Linklings and Softconf matches papers to reviewers based on their knowledge of submissions and committee members, with the help of scores that are computed automatically from keywords provided by committee members and authors. This is a matching problem. This kind of problem is also well researched by Garg et al.[6]. In our project, we have focused on both covering and matching problem.

There is a lot of research in the problem of team formation in the context of social networks [7][8][9][10][11][12]. Given a pool of experts and a set of skills that needed to be covered, the goal is to select a team of experts that can collectively cover all required skills, also ensuring communication cost between the team's members is minimized. Different works have been published that focuses on using different definitions of communication costs

between experts. Including communication on diameter distance, steiner distance, sum of distances and distance from the leader of the team.

There are some works[13][14] which analyse a team's performance from an anthropological/sociological perspective.

Another class of problems related to our problem is the submodular welfare maximization problem [15][16][17][18]. In this problem, m items are given and have to be distributed among n players. Each player has utility function (w_i) or a welfare of assigining some items to a player is given: $w_i : 2^{[m]} \rightarrow \mathcal{R}_+$. The utility functions are assumed to be monotone and submodular. If player i gets a subset S_i, the goal is to partition the items into n disjoint subsets in order to maximize the total welfare $\sum_i^n w_i(S_i)$ of the players. We can map the players to task and items to workers. Instead of a cost function, we can have a preference function of workers to task. We want to maximize the preferences of the workers assigned to tasks. The main difference is that they are not considering the set coverage perspective into their problem definition. This can be considered for future work.

One of the main pre-requisites of our problem is the profiles of skills and tasks. There has been a lot of research on how to infer the skills of the experts [19][20][21][22][23][24]. Most of these works are based on first building a 'expert document' corpus and then building formal probabilistic models and graph models to infer the skills of the experts. Also, in our recently published research work [25], we extracted skills of an expert by analysing his personal communication data. We made use of the stackexchange dataset for this purpose. Also, there are social networking websites like `http:\linkedin.com`, where people share their skills.

1.3 Background

In this section, we will formally define set cover and task assignment problems and also discuss the solutions with their approximation guarantees.

1.3.1 Set cover

Two of the commonly used algorithms to solve set cover is the greedy method and the primal dual inspired solution.

The set cover problem is one of Karp's 21 NP-complete problems shown to be NP-complete in 1972 [26]. An instance of the minimum weighed set

cover problem is given below.

$$\begin{aligned} \text{Given: } & \mathcal{U} = \{e_1, e_2, e_3, ..., e_n\} \\ & \mathcal{S} = \{S_1, S_2, S_3, ..., S_m\}, \forall S_i \subseteq \mathcal{U} \\ & c_s \geq 0, \forall s \in \mathcal{S} \\ \text{We want to find, } & \mathcal{Q} \subseteq \mathcal{S} \\ & \textit{such that } \cup_{Q \in \mathcal{Q}} Q = \mathcal{U} \quad \& \quad \sum_{Q \in \mathcal{Q}} C(Q) \text{ is minimised} \end{aligned}$$

We are given an Universal set of elements $\mathcal{U}$ and set of subsets $\mathcal{S}$. And each set $S \in \mathcal{S}$ is subset of the Universal set $\mathcal{U}$ i.e., $S \subseteq \mathcal{U}$. And each set $S \in \mathcal{S}$ has a cost associated with it $S(S) \geq 0$. We want to find a subset $\mathcal{Q} \subseteq \mathcal{S}$, such that $\sum_{Q \in \mathcal{Q}} C(Q)$ is minimised.

1.3.2 Greedy Algorithm

A greedy algorithm for the set cover with $log(n)$ [27] approximation is given below,

Algorithm 1 GREEDY-SET-COVER

INPUT: Universal Set $\mathcal{U}$, Sets $\mathcal{S}$, Cost function $C()$.
OUTPUT: $\mathcal{Q} \subseteq \mathcal{S}$.
while $\mathcal{U} \neq \phi$ **do**
  $S \leftarrow argmin_S \quad \frac{C(S)}{|S \cap \mathcal{U}|}$
  $\mathcal{U} = \mathcal{U} \setminus S$
  $S = \mathcal{S} \setminus S$
end while

In each iteration, the set which covers more number of elements is added to the final solution. The algorithm ends when all the elements are covered by the sets in the final solution.

1.3.3 Primal Dual Algorithm

In addition to the greedy algorithm, the set cover problem can also be solved using the primal dual method. The primal of the minimum set cover is,

$$min \quad \sum_{S \in \mathcal{S}} c_S X_S \tag{1.1}$$

$$\sum_{e \in S} X_S \geq 1 \quad \forall e \in \mathcal{U} \tag{1.2}$$

$$X_S \in \{0, 1\} \quad \forall S \in \mathcal{S} \tag{1.3}$$

The corresponding dual is of the set cover problem is a "set-packing" problem.

$$min \quad \sum_{e \in \mathcal{U}} Y_e \tag{1.4}$$

$$\sum_{e \in S} Y_e \leq c_S \quad \forall S \in \mathcal{S} \tag{1.5}$$

$$X_S \geq 0 \quad \forall e \in \mathcal{U} \tag{1.6}$$

In the primal-dual algorithms, one will alternate between the primal and the dual. Update a primal vector x and a dual vector y in steps so that eventually both vectors become feasible. If (relaxed) complementary slackness holds for the final x, y (with parameters α, β) then x is an ($\alpha\beta$-approximate) optimum solution of the primal [28]. The starting values and the update steps for x, y require careful and problem-specific design. Our initial primal-dual solution is very similar to the primal-dual solution of minimum set cover problem.

1.3.4 Task Assignment

We said earlier that the solution to a task assignment problem can be obtained in polynomial time using the hungarian algorithm. In this section we will be explaining the hungarian algorithm and also the changes that have to be made to our problem settings.

In our problem setting of task assignment, we assume a set of workers $\mathcal{W}$ and set of tasks $\mathcal{T}$ are given to us. Each worker will possess only one skill and each task will have one skill to complete the task. Also, the worker will have a cost of working on a task. The assignment problem is to assign tasks to workers so as to minimize the total cost. Since each worker can perform at most one task and each task can be assigned to only one worker.

In this problem, we assume, the number of workers will be greater than the number of tasks i.e., $|\mathcal{W}| > |\mathcal{T}|$. Hence, the cost matrix $\mathcal{C}$ will be rectangular

matrix of size $|\mathcal{W}| \times |\mathcal{T}|$. The value c_{ij} is the cost of assigning worker i to task j. If the worker i does not possess the skill to complete task j, then $c_{ij} = 0$.

In the hungarian algorithm, it is expected that the matrix is a square matrix. So, we modify the cost matrix in such a way that size of C is $|\mathcal{W}| \times |\mathcal{W}|$, and for the extra columns, we will pad with 0 values. Given the cost matrix C, we will use the hungarian algorithm to assign $|\mathcal{T}|$ workers to $|\mathcal{T}|$ tasks such that the total cost is reduced.

The algorithm is explained below,

1. For each row of the matrix, find the smallest element and subtract it from every element in its row.

2. For each column of the matrix, find the smallest element and subtract it from every element in its column.

3. Next, draw lines through rows and columns so that all the zero entries of the cost matrix and the minimum number of such lines is used.

4. If the minimum number of covering lines is $|W|$, an optimal assignment of zeros is possible and we are finished. If there is a zero which is unique in a row and column, the worker indicated by the row will be assigned to the task indicated by the column.

5. If the minimum number of covering lines is less than $|T|$, then determine the smallest entry not covered by any line. Subtract this entry from each uncovered row (row which is not marked), and then add it to each covered column. Return to Step 3.

Let us consider an example of the task assignment problem. Imagine there are 3 tasks and 5 workers. And the cost matrix is given below:

$$\begin{bmatrix} 2 & 8 & 9 \\ 5 & 2 & 8 \\ 8 & 7 & 6 \\ 6 & 5 & 3 \\ 6 & 7 & 1 \end{bmatrix}$$

After padding with zeros, the matrix is looks like below,

$$\begin{bmatrix} 2 & 8 & 9 & 0 & 0 \\ 5 & 2 & 8 & 0 & 0 \\ 8 & 7 & 6 & 0 & 0 \\ 6 & 5 & 3 & 0 & 0 \\ 6 & 7 & 1 & 0 & 0 \end{bmatrix}$$

The row minimum is zero. Hence subtracting row minimums has no effect. So, we will subtract the column minimum from each column.

$$\begin{bmatrix} 0 & 6 & 8 & 0 & 0 \\ 3 & 0 & 7 & 0 & 0 \\ 6 & 5 & 5 & 0 & 0 \\ 4 & 3 & 2 & 0 & 0 \\ 4 & 5 & 0 & 0 & 0 \end{bmatrix}$$

Now, lets draw lines through rows and columns which have zeros.

$$\begin{bmatrix} 0 & 6 & 8 & 0 & 0 \\ 3 & 0 & 7 & 0 & 0 \\ 6 & 5 & 5 & 0 & 0 \\ 4 & 3 & 2 & 0 & 0 \\ 4 & 5 & 0 & 0 & 0 \end{bmatrix}$$

We see that the number of striked lines is equal to 5. Workers 1,2 and 5 are assigned to tasks 1,2 and 3 respectively. And note that imaginary tasks 4 and 5 are not assigned to any workers.

For the rest of the paper, we will assume that we have a function HUNGARIAN which will take a cost matrix as input and output an assigment of workers to tasks. For the implementation, we will be using this package [29] in python and the padding is internally done by the package.

Chapter 2

Problem Statement

In this section we formally define the problem.

Input: There is a set of skills $\mathcal{S}$, a set of tasks $\mathcal{T}$ and a set of workers $\mathcal{W}$. Each task $t \in \mathcal{T}$ requires a set of skills, $S_t \subseteq \mathcal{S}$ to be completed. Also each worker $w \in \mathcal{W}$ possesses a set of skills, $S_w \subseteq \mathcal{S}$. For every worker-task pair we associate cost $d(w, t)$. This cost quantifies what is the cost of associating worker w to task t. This can also be the seen as how much w dislikes t (inverse of preference). We call this the DisSatisfaction function. Note that this can also be seen as the cost of paying money to worker for working on a task.

Coverage: We use $f : \mathcal{W} \to \emptyset \cup \mathcal{T}$ to denote an assignment of workers to tasks such that for every worker w, $f(w)$ is either a single task or no task at all. Given f we declare a task t to be f - *covered* or a task t is covered if:

$$(\cup_{w:f(w)=t} S_w) \cap S_t = S_t. \tag{2.1}$$

We consider an additional constraint that a worker $w \in \mathcal{W}$ can be assigned to only one task $t \in \mathcal{T}$.

Output: We want to find an assignment of workers to tasks, such that all the skills required to complete each and every task is covered by their respective assigned workers.

Problem 1. Team-Formation*: Given a set of tasks $\mathcal{T}$ to complete, a set of workers $\mathcal{W}$ to be assigned tasks and a* DisSatisfaction *function d(). Find an assignment of workers to tasks, such that all the skills required to complete each and every task is covered by the assigned the workers with an additional constraint that a worker can be assigned to only one task.*

Let us use an indicator variable X_{wt} to indicate the task the worker is

assigned to.

$$X_{wt} = \begin{cases} 1, & \text{if } f(w) = t \\ 0, & \text{otherwise.} \end{cases} \tag{2.2}$$

The integer linear programming formulation for the above problem statement is given below,

$$\min \quad \sum_{t \in T} \sum_{w \in W} d(w,t) X_{wt} \tag{2.3}$$

$$\sum_{s \in w} X_{wt} \geq 1 \quad \forall s \in t, \forall t \in T \tag{2.4}$$

$$\sum_{t \in T} X_{wt} \leq 1 \quad \forall w \in W \tag{2.5}$$

$$X_{wt} \in \{0, 1\} \tag{2.6}$$

So, equation (2.4) is about minimizing the dis-satisfaction of the assigned workers. Equations (2.5) and (2.6) indicates that all skills must covered for all the tasks and each worker will be assigned to only one task respectively.

Proposition 1. Team-Formation *problem is NP complete*

Proof. To prove that Team-Formation is NP-Complete we need to,

1. First, prove that our problem is in NP.
2. Reduce one of the known NP complete problem to our problem.

Let us consider a simplified instance Team-Formation problem. The number of tasks $|T| = 1, T = \{t\}$ and $d(w,t) = 1, \forall w \in W$. The ILP formulation for this simplified version is,

$$min \quad \sum_{w \in W} X_{wt} \tag{2.7}$$

$$\sum_{s \in w} X_{wt} \geq 1 \quad \forall s \in t \tag{2.8}$$

The above formulation is nothing but the minimum set cover problem.

This does not essentially prove that Team-Formation is NP complete. To prove that our problem is NP complete, let us write down the decision version of our problem. The problem is to find whether there is a team of at most k workers that covers task t. So, we will have a set of skills S_t and set of workers $\mathcal{W}$. For each worker $w \in \mathcal{W}$, will have a set of skills S_w. And our goal is to check whether there exists a set $C \subseteq \mathcal{W}$, such that $|C| = k$ and

$\cup_{w \in C} S_w = S_t$ Given a set C, a certifier can easily check the length of the set. Also a certifier can check whether the union of sets in C is equal to U or not. Hence this problem is in NP.

Now we can reduce the minimum set cover problem to our simplified instance. In the set cover problem , we have a universe set U and set of subsets $S_i \in S$ such that $S_i \subseteq U$ and we want to find out whether there are at most m subsets from S called $C, |C| = m$ such that $\cup_{S' \in C} S' = U$

If we map S_t to U and $S_i \in S$ to $w \in W$, k to m, then the problems become identical. Hence TEAM-FORMATION problem is a special instance of the minimum set cover problem. Hence TEAM-FORMATION problem is also NP-Complete. □

Most of the times in real world applications, it is not possible to find workers to cover all the tasks. This is mainly because of the rare skills that are required to complete the task and not enough workers possess these skills. For example, imagine there is a rare skill $\mathcal{A}$ which is required to complete 6 tasks, but only 3 workers possess that skill. In that case, 3 tasks cannot be completely covered.

We also abuse notation of $f()$ and use $f^{-1}(t)$ to denote the set of workers that are assigned to t if t is *f - covered*. We also use T_f to denote all the *f - covered* tasks due to f and $\overline{T}_f = T \setminus T_f$.

Considering this situation, our goal is to find an assignment that associates every worker with at most one task such that we maximize the number of tasks that get covered and we also keep the workers as satisfied as possible.

Our objective function changes to

$$min \quad \sum_{t \in T} \sum_{w \in W} d(w,t) X_{wt} + Z \tag{2.9}$$

where, Z is the penalty you pay for not covering some tasks. We propose two approaches to penalize uncovered tasks or to find an alternative for uncovered tasks.

1. **Penalize all uncovered Tasks**: In this approach we penalize all the tasks that are not covered. Then,

$$Z = \lambda |\overline{T_f}| \tag{2.10}$$

we pay a penalty of λ for every task that is not *f - covered*. This can also be thought as outsourcing the task to a new organization. This way all the tasks are covered. Let us call this problem OUTSOURCING-TASKS.

2. **Training skills**: In this approach, to cover the skills of the tasks that are not covered, we pay an extra cost to train the workers.

$$Z = \sum_{t \in \overline{T_f}} \sum_{s \in (S_t \setminus (\cup_{w:f(w)=t} S_w))} \gamma \tag{2.11}$$

The cost of training a worker on any skill is γ. After we train the workers with the new skills, all the tasks will be covered. Let us call this problem Training-Workers.

So we have two different problem settings of the Team-Formation problem. In this project, we use an innovative way of including the outsourcing tasks cost and training cost into the pool of workers $\mathcal{W}$.

2.1 Outsourcing Tasks

In Outsourcing-Tasks problem setting, if we can not complete tasks we can outsource the tasks to a third party organisation. Imagine these organisations as organizations who specialize only on a certain task. Their preference quantity of working on a task will be infinity for tasks they don't specialize in. We can add this organization into the pool of workers $\mathcal{W}$. This worker will have all the skills required to cover the task they specialize in.

$$\begin{aligned} &\forall t \in \mathcal{T}, \\ &w = \text{new worker} \\ &S_w = S_t \\ &d(w, t') = \begin{cases} \lambda, & \text{if } t' = t \\ \infty, & \text{otherwise} \end{cases} \\ &\mathcal{W} = \mathcal{W} \cup w \end{aligned}$$

2.2 Training Workers

In Training-Workers problem setting, if we can not complete tasks we can train the workers on the skills to completely cover the tasks. We can set a training cost for every skill as γ. And we can create a worker for every skill and for every task and the preference quantity for this worker is γ for the

task it is associated with and ∞ for other tasks.

$$\begin{aligned} &\forall t \in \mathcal{T}, \\ &\forall s \in S_t, \\ &w = \text{new worker} \\ &S_w = s \\ &d(w, t') = \begin{cases} \gamma, & \text{if } t' = t \\ \infty, & \text{otherwise} \end{cases} \\ &\mathcal{W} = \mathcal{W} \cup w \end{aligned}$$

The value of γ can be constant, meaning that the training costs of all the skills are the same. Also, one can infer the cost of γ from the skills that the worker already possesses.

Chapter 3

Algorithms

We have already proved that the problem is NP hard to solve. Hence we have proposed different variations of greedy set cover algorithms and variations of assignment algorithms. The algorithms to solve the problem can be classified into three categories. They are,

- Iterating over tasks
- Iterating over workers
- Iterating over skills

3.1 Iterating over tasks:

In this algorithm, we will choose a task and try to cover all the skills required to complete the task greedily. We do this till all the tasks are covered. Note that, when we take only one task our problem reduces to a minimum weighed set cover problem.

First, we choose a task. Then, we solve an instance of the min set cover problem. All the workers assigned to this task are removed from the pool of workers as the workers cannot be assigned to more than one task. This is continued till all the tasks are covered.

Algorithm 2 GREEDY-TASKS

INPUT: Tasks $\mathcal{T}$, Workers $\mathcal{W}$, Dislike function $d()$.
OUTPUT: $f : \mathcal{W} \rightarrow \phi \cup \mathcal{T}$.
for $t \in \mathcal{T}$ **do**
    $\mathcal{Q} \leftarrow$ GREEDY-SET-COVER$(S_t, \mathcal{W}, d(\mathcal{W}, t))$
    $f(w) \leftarrow t, \forall w \in \mathcal{Q}$
    $\mathcal{W} \leftarrow \mathcal{W} \setminus \mathcal{Q}$
end for

The output from this algorithm is not deterministic as the order in which the tasks are chosen impacts the final output.

3.2 Iterating over Workers

In this category, we assign a worker to a task in a greedy manner. For every worker-task pair we will compute a $c(w,t) = \frac{d(w,t)}{|S_w \cap S_t|}$. We will find the worker task pair for which $c(w,t)$ is minimum. We assign the worker w to task t. We remove this worker from the pool of workers and also remove the skills of the worker from the task.

Algorithm 3 GREEDY-WORKERS

INPUT: Tasks $\mathcal{T}$, Workers $\mathcal{W}$, DISSATISFACTION function $d()$.
OUTPUT: $f : \mathcal{W} \rightarrow \phi \cup \mathcal{T}$.
while $T \neq \phi$ **do**
    for every worker-task pair (w,t) **do**
        $c(w,t) \leftarrow \frac{d(w,t)}{|S_w \cap S_t|}$
    end for
    $w, t \leftarrow argmin_{w,t} \quad c(*,*)$
    $f(w) \leftarrow t$
    $\mathcal{W} \leftarrow \mathcal{W} \setminus w$
    $S_t \leftarrow S_t \setminus S_w$
    if t is completely covered **then**
        $T \leftarrow T \setminus t$
    end if
end while

3.3 Iterating over Skills

The algorithms in this category will give importance to skills. And cover the skills in a greedy manner. The basic idea is to find the best assignment of a worker to cover the skill of a task. We have two algorithms using this idea. They are,

- Extended-Hungarian Algorithm
- Primal-Dual Algorithm

3.3.1 Extended Hungarian Algorithm

Hungarian Algorithm is a optimization algorithm that solves the assignment problem. In this problem, we are given a non negative nxn matrix, where the element in the i-th row and j-th column represents the cost of assigning the j-th task to the i-th worker. We have to find an assignment of the jobs to the workers that has minimum cost. This problem can also be formulated as a Bipartite Graph matching problem. In its native form, the hungarian algorithm does not consider skill association with workers and tasks. But we can modify this algorithm to our benefit. In Extended-Hungarian Algorithm, we choose a skill s. Next, we identify the set of workers $W' \subseteq \mathcal{W}$ who possess the skill. Also, we identify the set of tasks $T' \subseteq \mathcal{T}$ that require this skill s to be covered. Then we create a matrix M of workers to tasks of size $|W'|X|T'|$, where the cost of assigning worker $w \in W'$ to task $t \in T'$ is equal to $d(w, t)$. Note that the matrix will not always be square. Hence the matrix will be padded with 0 to make it a square matrix. We make use of the Hungarian Algorithm to make an assignment of workers to tasks. The assigned workers are removed from the pool of workers and the skills covered by the assigned workers are removed from the tasks. This is continued till all the skills of all the tasks are covered.

Algorithm 4 EXTENDED-HUNGARIAN

INPUT: Tasks $\mathcal{T}$, Workers $\mathcal{W}$, DISSATISFACTION function $d()$.
OUTPUT: $f : \mathcal{W} \rightarrow \phi \cup \mathcal{T}$.
while $T \neq \phi$ **do**
  Choose a skill s
  $T' \subseteq T, \quad T' \leftarrow \{t | s \in T'\}$
  $W' \subseteq W, \quad W' \leftarrow \{w | s \in W'\}$
  $M_{wt} \leftarrow \frac{d(w,t)}{|S_w \cap S_t|}, \forall w \in W', t \in T'$
  $g() \leftarrow$ HUNGARIAN(M)
  for each $(w, t) \in g()$ **do**
    $f(w) \leftarrow t$
    $S_t = S_t \setminus S_w$
    $\mathcal{W} \leftarrow \mathcal{W} \setminus w$
    if t is completely covered **then**
      $\mathcal{T} \leftarrow \mathcal{T} \setminus t$
    end if
  end for
end while

3.3.2 Primal Dual Algorithm

This algorithm is inspired by the primal dual algorithm for minimum set cover. Our algorithm is also very similar but with a few changes. And this algorithm also gives the same approximation as for the min set cover. Let us approximate the team formation problem using a primal-dual algorithm.
The primal of the ILP is given as:

$$min \quad \sum_{t \in T} \sum_{w \in W} d(w,t) X_{wt} \tag{3.1}$$

$$\sum_{s \in w} X_{wt} \geq 1 \quad \forall s \in t, \forall t \in T \tag{3.2}$$

$$\sum_{t \in T} X_{wt} \leq 1 \quad \forall w \in W \tag{3.3}$$

$$X_{wt} \in \{0, 1\} \tag{3.4}$$

The respective dual is given as:

$$max \quad \sum_{t \in T} \sum_{s \in t} Y_{st} - \sum_{w \in W} Z_w \tag{3.5}$$

$$\sum_{s \in w} Y_{st} - Z_w \leq d(w,t) \quad \forall w \in W, \forall t \in T \tag{3.6}$$

$$Y_{st} \geq 0 \tag{3.7}$$

$$Z_w \geq 0 \tag{3.8}$$

The dual variables, Y_{st} is the packing cost of skill s of task t. Z_w is an indicator variable.

$$Z_w = \begin{cases} > 0, & \text{if } w \text{ is assigned} \\ 0, & \text{otherwise} \end{cases} \tag{3.9}$$

In this new primal dual algorithm, we first randomly select a task $t \in \mathcal{T}$ and a skill $s \in \mathcal{S}_t$. Then, we increase the cost of that dual variable Y_{st} such that for some $w \subseteq W, \sum_{s \in w} Y_{st} = d(w,t)$ becomes tight. Intuitively, the packing cost of all that skill is paid. And this worker w will be assigned to the task t. This worker will be removed from the pool of available workers.

Algorithm 5 Primal-Dual-1

```
INPUT: Tasks 𝒯, Workers 𝒲, DisSatisfaction function d().
OUTPUT: f : 𝒲 → ϕ ∪ 𝒯 .
Set Y_st = 0, ∀s ∈ t, ∀t ∈ T
Set f() = ϕ, Z_w = 0, ∀w ∈ W
while T ≠ ϕ do
    Select some t ∈ T & s ∈ t
    Increase the price of Y_st such that for some w ⊆ W, Σ_{s∈w} Y_st = d(w,t)
    becomes tight
    Let W' be the set of workers for whom, the condition becomes tight.
    for all w ∈ W' do
        W = W \ w
        S_t = S_t \ S_w
        if t is completely covered then
            T ← T \ t
        end if
    end for
end while
```

According to the definition of the relaxed complementary slackness, we

will have:

$$X_{wt} > 0 \implies \sum_{s \in w} Y_{st} - Z_w = d(w,t) \quad \alpha = 1 \tag{3.10}$$

$$Y_{st} > 0 \implies s \in s_w \quad Z_w > 0 \implies \sum_{s \in w} X_{wt} \leq d \quad \beta = d \tag{3.11}$$

$$d = max_{s \in \mathcal{S}} |\{j : s \in w \in W\}| \tag{3.12}$$

From the algorithm we see that, whenever we have $X_{wt} > 0$ (that is, $X_{wt} = 1$) it holds that the corresponding dual constraint is tight, that is $\sum_{s \in t} Y_{st} = d(w,t)$. Thus, relaxed primal complementary slackness holds with $\alpha = 1$. Second, whenever we have $Y_{st} > 0$, let us observe that at most d workers became tight for s since at most d workers contain s.Thus, $\sum_{w:s \in w} X_{wt} \leq d$ which implies that relaxed dual complementary slackness holds with $\beta = d$. If C is the final solution, it follows that C is a d-approximation for our team formation problem. Indeed, $|C| = \sum_{w,t} X_{wt}$ and by relaxed complementary slackness we have $\sum_{w,t} X_{wt} \leq d \sum_{s,t} Y_{st}$,where y is dual feasible and hence a lower bound for the fractional (and hence integral) optimum of the primal.

The Primal-Dual algorithm described above too random and the results depend on the skills chosen and results were also not so desirable. Hence, inspired by the primal dual algorithms in the text book by Williamson and Shmoys [30], we came up with a new algorithm. In this new primal dual algorithm, we increase the cost of all the dual variables uniformly, until one of the constraints is satisfied. Intuitively, the packing cost of all the skills of all the tasks are increased until one of the contraints become tight. A constraint for a worker, task (w,t) pair might become tight and this worker w will be assigned to the task t.

Algorithm 6 Primal-Dual-2

INPUT: Tasks $\mathcal{T}$, Workers $\mathcal{W}$, DisSatisfaction function $d()$.
OUTPUT: $f : \mathcal{W} \rightarrow \phi \cup \mathcal{T}$.
Set $Y_{st} = 0, \forall s \in t, \forall t \in T$
Set $f() = \phi, \forall w \in W$
while $T \neq \phi$ **do**
  for all $w \in W$ **do**
    for all $t \in T$ **do**
      $a(w, t, (s_w \cap s_t)) \leftarrow \frac{(d(w,t) - \sum_{s \in w} Y_{st})}{|s_w \cap s_t|}$
    end for
  end for
  $(w, t, S) \leftarrow argmin_{w,t,S} \quad a()$
  for $s \in S$ **do**
    $Y_{st} \leftarrow min \quad a()$
  end for
  $W = W \setminus w$
  $S_t = S_t \setminus S_w$
  if t is completely covered **then**
    $T \leftarrow T \setminus t$
  end if
end while

Note that the dual variable Z_w does not have any impact on the algorithm. But for combinatorial purposes the value of Z_w should be infinitely large so that if a worker is assigned, he should not be available for other tasks.

Henceforth, when we say, Primal-Dual we will be referring to Primal-Dual-2.

3.4 Chocolate and Chocolate Filling

In the above algorithms, we might have hired redundant workers. Imagine a set of workers assigned to a task. If we remove a worker and still the task is covered then the worker is redundant. We can do this step right at the end as post processing or after every assignment of worker to task. We call the algorithm using the post processing step as "Chocolate version" (C), analogous to the chocolate layer over the doughnut. The algorithm using the other version, where a check for the redundant worker is made after every assignment is called "Chocoloate-Filling version" (CF). This is analogous to the chocolate filling inside the doughnut. An example of both the versions using the Greedy-Workers is given below. These steps are represented by text in red color.

Algorithm 7 GREEDY-WORKERS-C

INPUT: Tasks $\mathcal{T}$, Workers $\mathcal{W}$, DISSATISFACTION function $d()$.
OUTPUT: $f : \mathcal{W} \rightarrow \phi \cup \mathcal{T}$.
$S_w \leftarrow$ be the set of skills of worker w
$Q_t, S_t \leftarrow$ be the set of skills of task t
while $T \neq \phi$ **do**
  for every worker-task pair (w, t) **do**
    $c(w, t) \leftarrow \frac{d(w,t)}{|S_w \cap S_t|}$
  end for
  $w, t \leftarrow argmin_{w,t} \quad c(*, *)$
  $f(w) \leftarrow t$
  $W \leftarrow W \setminus w$
  $S_t \leftarrow S_t \setminus S_w$
  if t is completely covered **then**
    $T \leftarrow T \setminus t$
  end if
end while
for all $t \in \mathcal{T}$ **do**
  for all $w \in f^{-1}(t)$ **do**
    if $\cup_{v \in f^{-1}(t) \setminus w} S_v = Q_t$ **then**
      $f(w) \leftarrow \phi$
    end if
  end for
end for

Algorithm 8 GREEDY-WORKERS-CF

INPUT: Tasks $\mathcal{T}$, Workers $\mathcal{W}$, DISSATISFACTION function $d()$.
OUTPUT: $f : \mathcal{W} \rightarrow \phi \cup \mathcal{T}$.
$S_w \leftarrow$ be the set of skills of worker w
$Q_t, S_t \leftarrow$ be the set of skills of task t
while $T \neq \phi$ **do**
  for every worker-task pair (w, t) **do**
    $c(w, t) \leftarrow \frac{d(w,t)}{|S_w \cap S_t|}$
  end for
  $w, t \leftarrow argmin_{w,t} \quad c(*, *)$
  $f(w) \leftarrow t$
  $W \leftarrow W \setminus w$
  $S_t \leftarrow S_t \setminus S_w$
  if t is completely covered **then**
    $T \leftarrow T \setminus t$
  end if
  for all $w \in f^{-1}(t)$ **do**
    if $\cup_{v \in f^{-1}(t) \setminus w} S_v = Q_t \setminus S_t$ **then**
      $f(w) \leftarrow \phi$
    end if
  end for
end while

If the algorithm is not using Chocolate of Chocolate Filling versions, then let us call that algorithm "VANILLA version".

Finally, the total cost or the total DISSATISFACTION of the workers assigned, is calculated as follows:

$$\text{Total cost} = \sum_{t \in T} \sum_{w \in W} d(w, t) X_{wt}. \tag{3.13}$$

Chapter 4

Data Preparation

4.1 StackExchange

To simulate a team formation problem, we made use of StackExchange [31] publicly available dataset [32]. The StackExchange is a Q&A forum where people ask questions about programming issues and also discuss solutions. Each question in StackExchange has atmost 5 tags. The users who post questions and also users who answer are associated with those tags. The union of the tags is considered as the set of skills $\mathcal{S}$. If we combine the tags of all the questions and answers of a user, then we have the set of skills associated with a user. Now, we have a set of users and each user will have a set of skills. Next, we consider some users as tasks and some of them as workers. Let us sort the users based on the number of skills they possess. And consider the top 55 users as tasks and the rest as workers. For our experiments, we only used cs.StackExchange.com users. Finally, we have a set of skill $\mathcal{S}$, set of workers $\mathcal{W}$ and set of tasks $\mathcal{T}$.

4.2 Bibsonomy

Our second dataset is extracted from Bibsonomy [33], a social-bookmarking and publication-sharing system. This dataset is very similar to the stackexchange dataset, where each publication is annotated with a set of tags. We use the set of tags associated with the papers of an author to represent the set of skills for that author. We divide the set of authors into two sets: one set representing the workers and another set representing the tasks. We consider the tasks to be the high ranked authors in the dataset.

The dis-satisfaction quantity of a worker w for working on a task t is

calculated as follows,

$$d(w,t) = \left(1 - \frac{|S_w \cap S_t|}{|S_w \cup S_t|}\right)^3.$$

Notice that, the term $\frac{|S_w \cap S_t|}{|S_w \cup S_t|}$ is the Jaccard Similarity between the worker and the task. The jaccard similarity would indicate a satisfaction cost of a worker working on a task. Since, we needed to calculate the DisSatisfaction cost, we used $1 - \frac{|S_w \cap S_t|}{|S_w \cup S_t|}$. We wanted to give more importance to workers who cover more skills of the task compared the workers who cover less skills of the task. So, we made the DisSatisfaction function cubic.

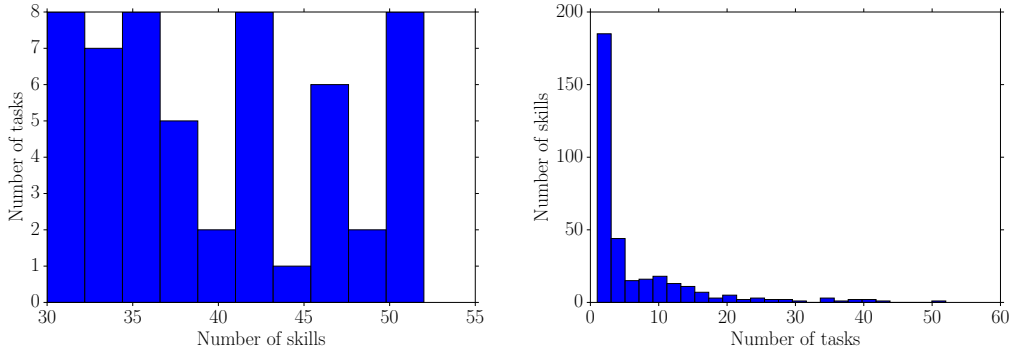
Let's see some statistics of the StackExchange dataset. Below is the table which shows the basic statistics like the number of skills, workers and tasks in the dataset.

Number of of Skills	423
Number of of Workers	5377
Number of of Tasks	55

Table 4.1: Basic statistics from StackExchange dataset

Next, we will see some comparisons between skills, workers and tasks. We will be plotting the following histograms,

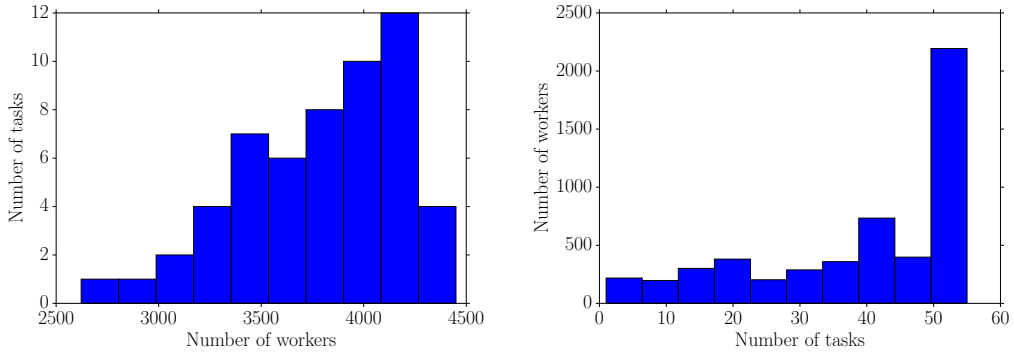
- Histogram of number of skills vs tasks: This plot will visualise the number of tasks having n skills.
- Histogram of number of tasks vs skills: This plot will visualise the number of skills that is present in k tasks.
- Histogram of number of workers vs tasks: This plot will visualise the number of tasks to which l workers can be assigned to. If a worker can cover atleast one of a task, then he can be assigned to that task.
- Histogram of number of tasks vs workers: This plot will visualise the number of workers who can be assigned to k tasks.
- Histogram of number of skills vs workers: This plot will visualise the number of workers having n skills.
- Histogram of number of workers vs skills: This plot will visualise the number of skills that is present in l workers.



(a) Histogram of number of skills vs tasks (b) Histogram of number of tasks vs skills

Figure 4.1: Statistics of StackExchange dataset - 1

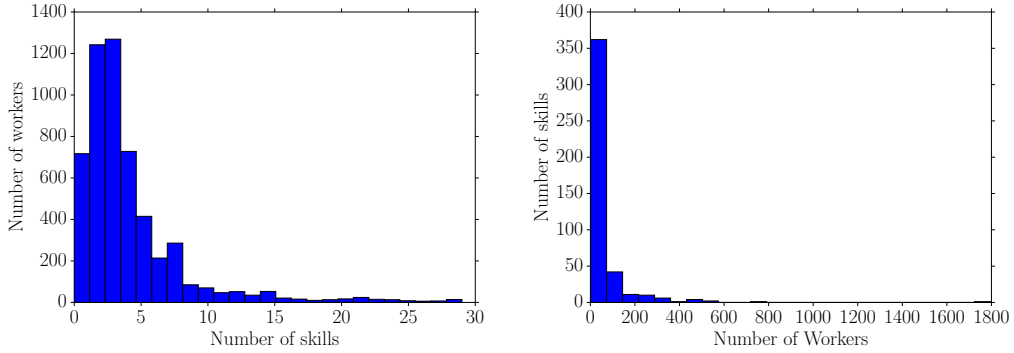
In 4.1(a), we see that number of skills in each task range between 30 to 55. We see from 4.1(b) most skills appear in less number of tasks and there is one skill which appears in all 55 tasks.



(a) Histogram of number of workers vs tasks (b) Histogram of number of tasks vs workers

Figure 4.2: Statistics of StackExchange dataset - 2

The figure 4.2(a) explain the number of workers that can be part of a task. And 4.2(b) visualises the number of task each worker can be part of. A worker can take part in a task, if he covers atleast one skill of the task. From 4.2(b), we see about 2000 workers can be part all tasks.



(a) Histogram of number of skills vs workers (b) Histogram of number of workers vs skills

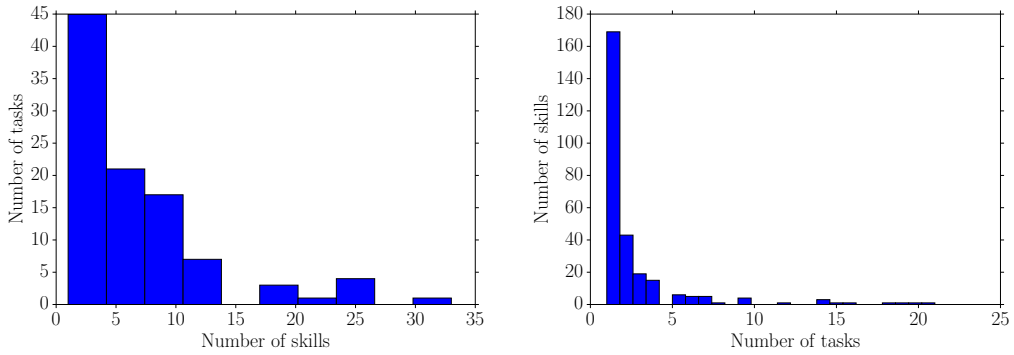
Figure 4.3: Statistics of StackExchange dataset - 3

In 4.3(a), we see that number of skills in workers is less and very few workers have more than 25 skills. Similar to tasks to skills histogram, 4.3(b) most skills appear in less number of workers.

Next, we will be seeing the same set of comparisons and statistics for BIBSONOMY dataset.

Number of skills	793
Number of workers	5000
Number of tasks	99

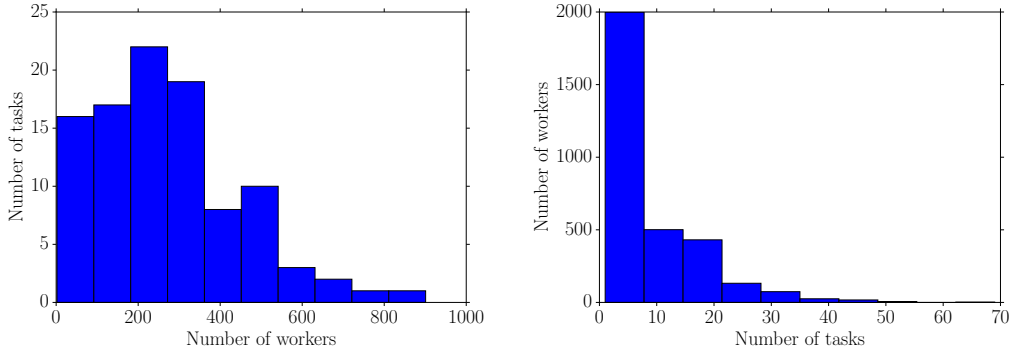
Table 4.2: Basic statistics from BIBSONOMY dataset



(a) Histogram of number of skills vs tasks (b) Histogram of number of tasks vs skills

Figure 4.4: Statistics of BIBSONOMY dataset - 1

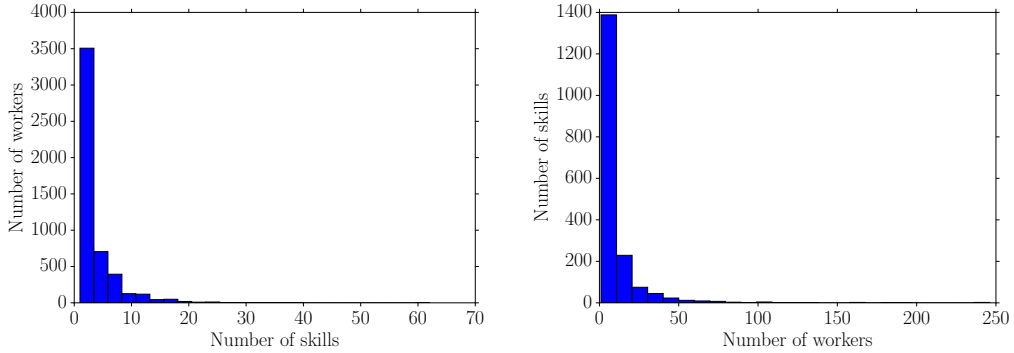
In 4.4(a), we see that number of skills in each task not larger than 35 and most tasks have less amount of skills. We see from 4.4(b) most skills appear in only one task.



(a) Histogram of number of workers vs tasks (b) Histogram of number of tasks vs workers

Figure 4.5: Statistics of BIBSONOMY dataset - 2

We see from 4.5(b) many workers can be part of very few tasks. This can be attributed the fact that there many unique skills.



(a) Histogram of number of tasks vs skills (b) Histogram of number of tasks vs skills

Figure 4.6: Statistics of BIBSONOMY dataset - 3

In 4.6(a), we see that number of skills most tasks have less amount of skills and there is no worker who has more than 70 skills. We see from 4.6(b) most skills appear in only one worker like we saw in the case of tasks vs skills histogram.

Chapter 5

Experiments

In this chapter, we will compare different versions of algorithms and also compare the algorithms against each other for the two different problem settings. The problem settings are Outsourcing-Tasks and Training-Workers. We will also use a lower bound algorithm which is very similar to the Greedy-Tasks, but instead of removing the assigned workers from the available pool of workers they are made available for the next tasks. This is not a feasible solution, but we can use this as a lower bound for our experiments. First, we choose a task. Then we solve an instance of the min set cover problem. All the workers assigned to this task are not removed from the pool of workers. This is continued till all the tasks are covered. Note that in this case $f(w)$ is a list and the tasks that the worker is assigned to, are appended to this list.

Algorithm 9 Lower-Bound

INPUT: Tasks $\mathcal{T}$, Workers $\mathcal{W}$, Extended-Hungarian function $d()$.
OUTPUT: $f : \mathcal{W} \rightarrow \phi \cup \mathcal{T}$.
for $t \in \mathcal{T}$ **do**
  $\mathcal{Q} \leftarrow$ GREEDY-SET-COVER$(S_t, \mathcal{W}, d(\mathcal{W}, t))$
  $f(w) \leftarrow t, \forall w \in \mathcal{Q}$
end for

5.1 Outsourcing tasks

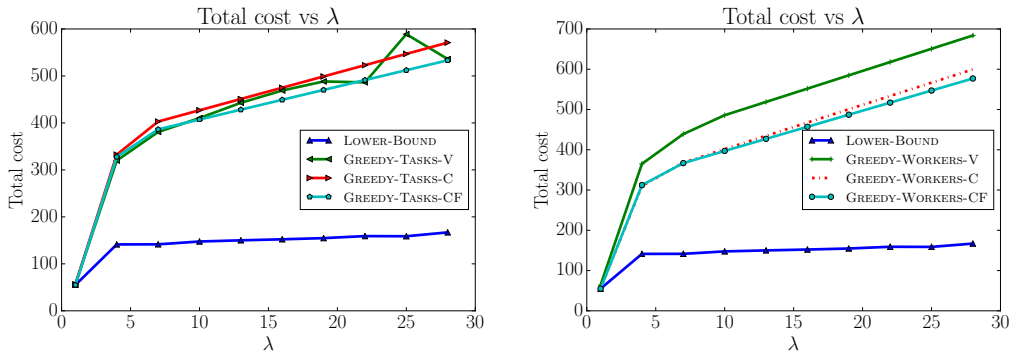
In this Outsourcing-Tasks problem setting, if we cannot completely cover the tasks using the available pool of workers, we will outsource the tasks to a third party organization. From the problem definition we saw that the cost of outsourcing is λ. We vary this parameter λ and see which algo-

rithm performs better. Observe that, when λ is very small, all the tasks are outsourced. When λ is large, only tasks which cannot be completely covered by the available workers will be outsourced. First, we will compare the Vanilla, Chocolate and Chocolate Filling versions of the same algorithm. Next, we will compare different algorithms.

Another thing to be noted, in Extended Hungarian algorithm, the iteration is over the rarest skills amongst the workers. The intuition behind using this logic is, we want to cover rare skills first by the set of available workers.

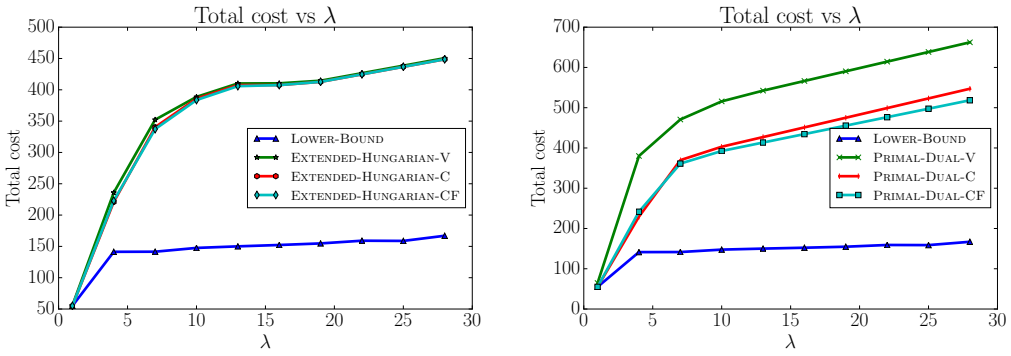
5.1.1 Comparison between Vanilla, Chocolate and Chocolate Filling

To find the best version, we compare the total cost produced by different versions of the algorithm. A good performing version will have a lower total cost. The comparsion between Vanilla, Chocolate and Chocolate Filling of all the algorithms for both the datasets is given below,



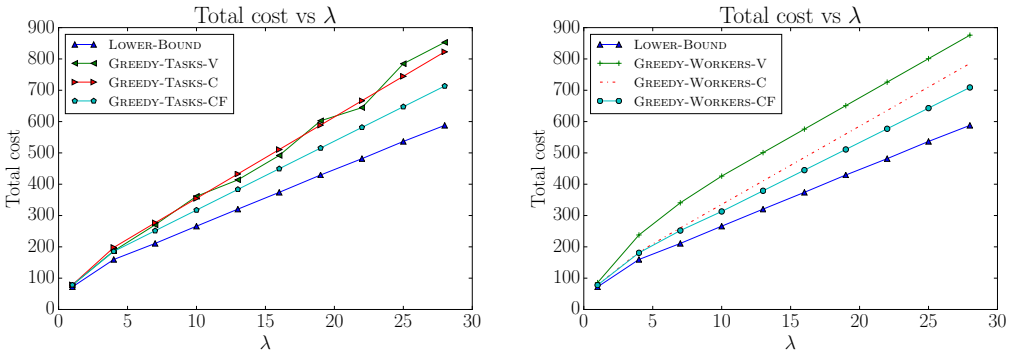
(a) Greedy-Tasks (b) Greedy-Workers

Figure 5.1: Comparison between Vanilla, Chocolate and Chocolate Filling - StackExchange dataset



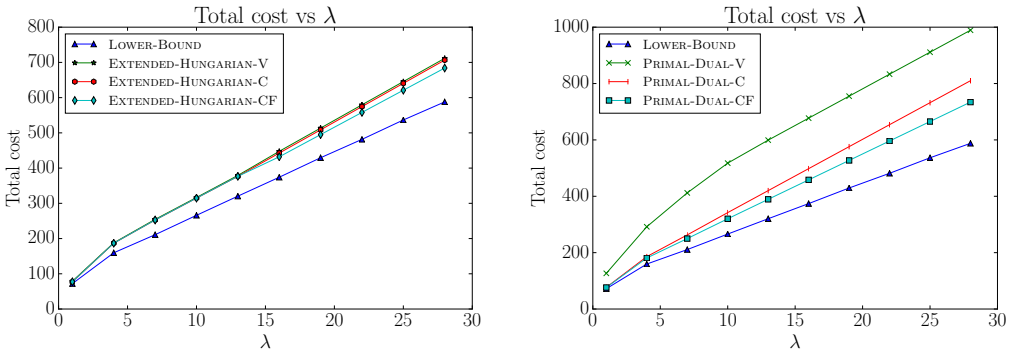
(a) Extended-Hungarian (b) Primal-Dual

Figure 5.2: Comparison between Vanilla, Chocolate and Chocolate Filling - StackExchange dataset



(a) Greedy-Tasks (b) Greedy-Workers

Figure 5.3: Comparison between Vanilla, Chocolate and Chocolate Filling - Bibsonomy dataset



(a) Extended-Hungarian (b) Primal-Dual

Figure 5.4: Comparison between Vanilla, Chocolate and Chocolate Filling - Bibsonomy dataset

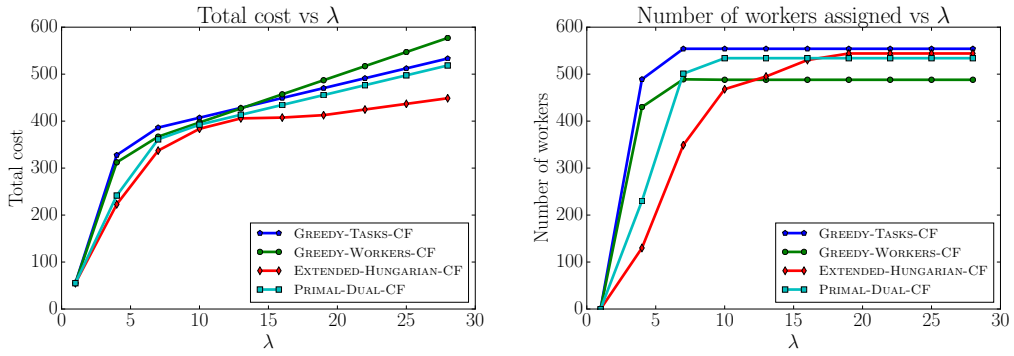
We notice that in all the algorithms and in both the datasets the Chocolate Filling (CF) version was the best. This is due to the fact that the no redudant workers are assigned to the tasks. All the workers are contributing in some way to cover the tasks. CF versions are better than C versions because, the redundant workers who were unassigned in from other tasks are used in some other tasks. Those unassigned workers might be a better assignment than other workers. And we will be comparing the (CF) versions of all the algorithms in the next sub section of determine the best performing algorithm.

5.1.2 Comparison between Algorithms

In this section, we will be comparing the Chocolate Filling version of all the algorithms, to see which algorithm performs well. First, we will see for the StackExchange dataset. The comparison is done using the following parameters:

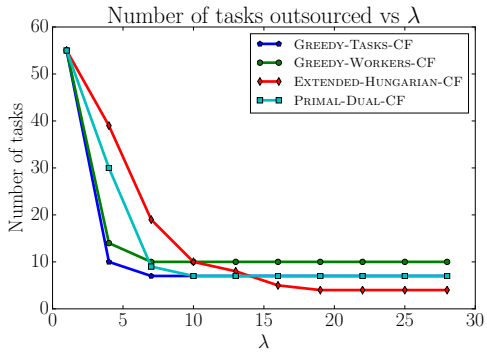
- Total cost.
- Number of workers assigned.
- Number of tasks outsourced.

A good performing algorithm must ideally have a very low cost, must have assigned more number of normal workers and less number of tasks must be outsourced. This way, most number of assigned workers are less dis satisfied.



(a) Cost (b) Workers Assigned

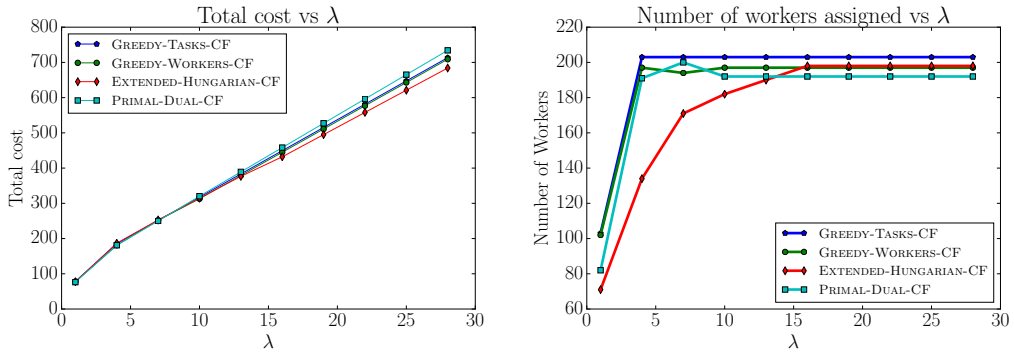
Figure 5.5: Comparison between algorithms StackExchange dataset



(a) Tasks Outsourced

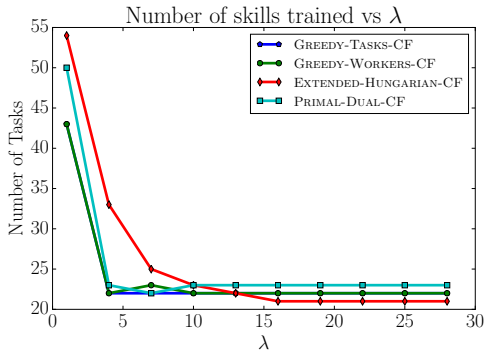
Figure 5.6: Comparison between algorithms StackExchange datase

We see that from figure 5.5(a), the total cost(dis-satisfaction) of the assigned workers from Extended-Hungarian-CF algorithm was the least and Greedy-Tasks-CF was the worst. We also see that from figure 5.5(b) that more number of workers was also more for Extended-Hungarian-CF. This is a good that more number of workers were assigned. And finally from figure 5.6(a) we see that lesser number of tasks are outsourced. From all these factors we can say that Extended-Hungarian-CF algorithm is the best for Outsourcing tasks feature. But, note that the difference to other algorithms is not significant.
Next, let's see the comparison of algorithms using the Bibsonomy dataset.



(a) Cost (b) Workers Assigned

Figure 5.7: Comparison between algorithms - Bibsonomy dataset



(a) Tasks Outsourced

Figure 5.8: Comparison between algorithms - Bibsonomy dataset

The performance in the Bibsonomy dataset is very similar to StackExchange dataset, but the difference between the performance of algorithms is even more less significant.

5.2 Training Workers

In this Training-Workers problem setting, if we cannot completely cover the tasks using the available pool of workers, we will train the assigned workers on the skills that are not covered. From the problem definition we saw that the cost of training is γ. We vary this parameter γ and see which algorithm performs better. Observe that, when γ is very small, all the

skills will be trained. When γ is large, only skills from the tasks which are not completely covered are trained on the assigned workers.

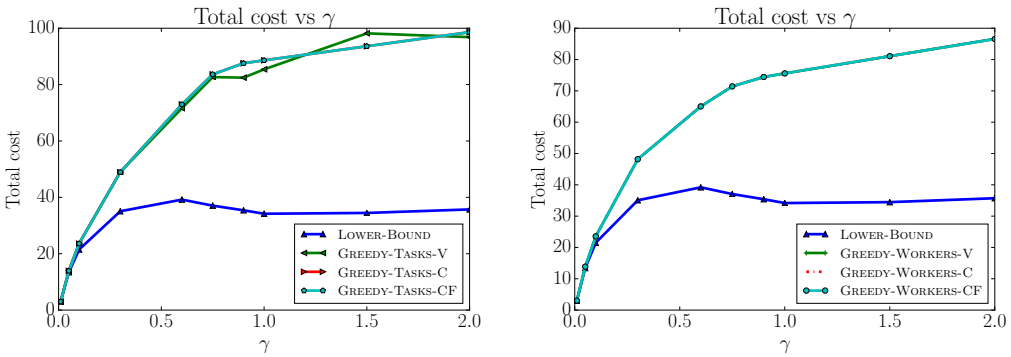
In Extended-Hungarian algorithm, the iteration is should be over the popular skills amongst the workers. The intuition is that, we want to cover popular skills among the workers and be able to train on the rare skills. The running time for Hungarian algorithm is $O(n^3)$. When we iterate over the popular skills the size of the matrix used in Hungarian algorithm for matching is very huge. Hence, to facilitate the Extended-Hungarian algorithm, we will reduce the number of tasks and workers. We will sample a small number of tasks and small number of workers for this purpose.

The updated stats for StackExchange and Bibsonomy datasets are given below,

	StackExchange	Bibsonomy
Number of skills	335	774
Number of workers	500	500
Number of tasks	10	10

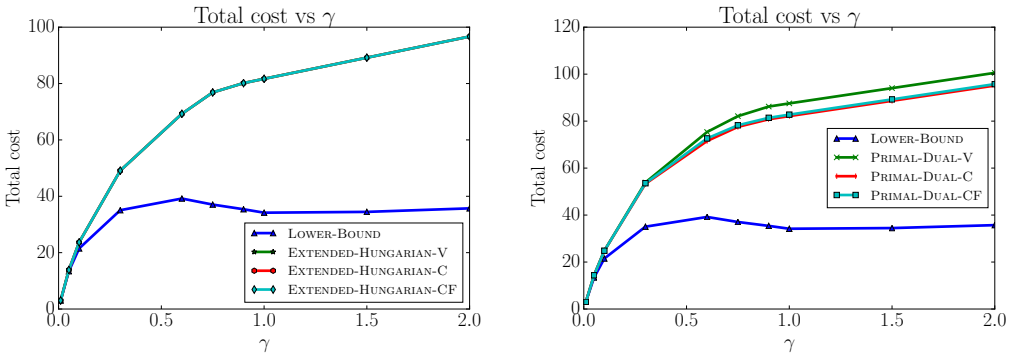
Again here, we follow the same procedure. First we will compare the Vanilla, Chocolate and Chocolate Filling versions of the same algorithm. Then, we will compare the different algorithms.

5.2.1 Comparison between Vanilla, Chocolate and Chocolate Filling



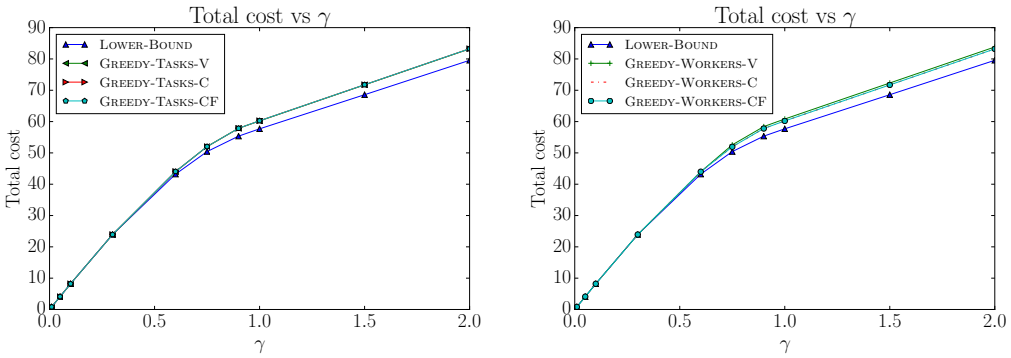
(a) Greedy-Tasks (b) Greedy-Workers

Figure 5.9: Comparison between Vanilla, Chocolate and Chocolate Filling - StackExchange dataset



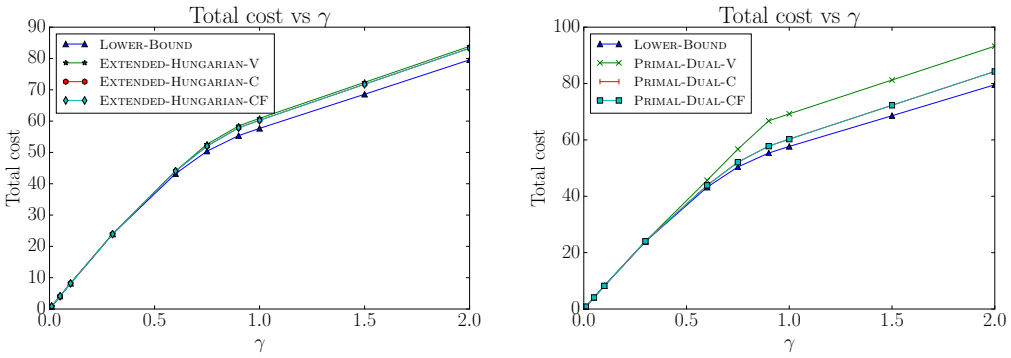
(a) Extended-Hungarian (b) Primal-Dual

Figure 5.10: Comparison between Vanilla, Chocolate and Chocolate Filling - StackExchange dataset



(a) Greedy-Tasks (b) Greedy-Workers

Figure 5.11: Comparison between Vanilla, Chocolate and Chocolate Filling - Bibsonomy dataset



(a) Extended-Hungarian (b) Primal-Dual

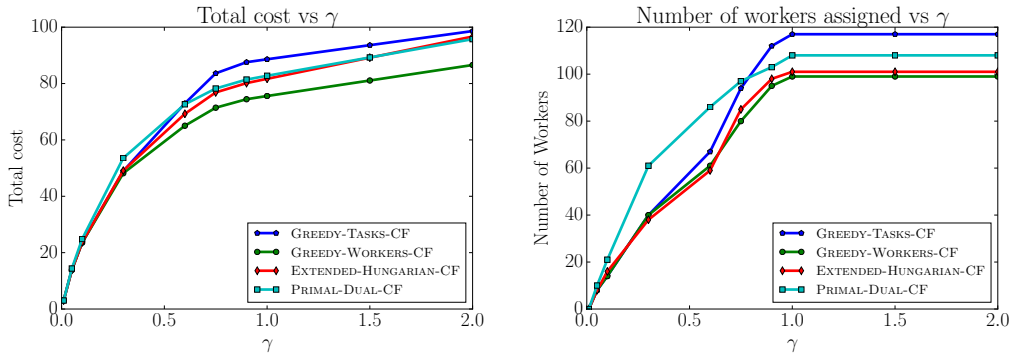
Figure 5.12: Comparison between Vanilla, Chocolate and Chocolate Filling - Bibsonomy dataset

Notice that there is not much difference between the different versions of the algorithm. Because in outsourcing tasks, when the task is outsourced, all the workers that are assigned to the task are un assigned and they are made available for other tasks. This is not the case in training workers. When a skill is trained, that means that skill is not possesed by any of the workers. There will be no worker that will be un assigned in this case. Also because, we considered a very small number of tasks and workers to facilitate the Hungarian algorithm, the difference in the performance is less significant. Since CF version is the most intuitively best algorithm, we will be comparing the (CF) versions of all the algorithms in the next sub section of determine the best performing algorithm.

5.2.2 Comparison between Algorithms

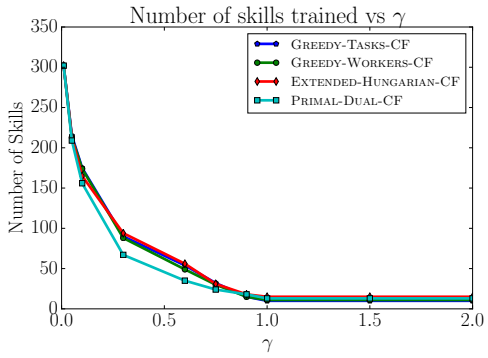
In this section, we will compare the different algorithms. We are the doing similar set of comparisons as for the Outsourcing-Tasks setting. But instead of comparing the number of outsourced tasks, we are comparing the number of skills trained.

A good performing algorithm must ideally have a very low cost, must have assigned more number of normal workers and less skills to be trained on. This way, most number of assigned workers are less dis satisfied.



(a) Cost (b) Workers Assigned

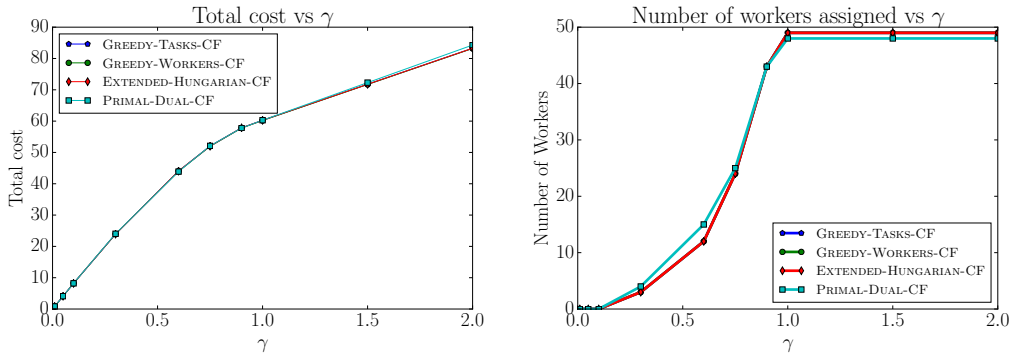
Figure 5.13: Comparison between algorithms - StackExchange dataset



(a) Skills Trained

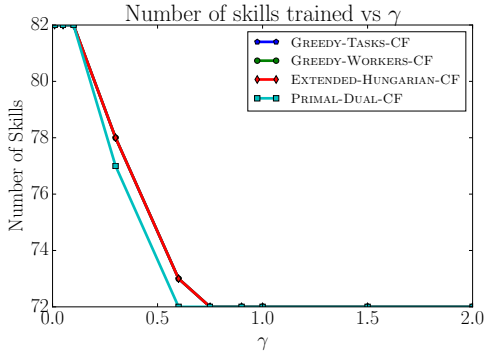
Figure 5.14: Comparison between algorithms - StackExchange dataset

We see that from figure 5.13(a), the total cost(dis-satisfaction) of the assigned workers from Greedy-Workers-CF algorithm was the least and Greedy-Tasks-CF was the worst. We also see that from figure 5.13(b) that more number of workers was also more for Greedy-Workers-CF. This is a good thing that more number of workers were assigned. And finally from figure 5.14(a) we see that almost all algorithms are training on the same number of skills.



(a) Cost (b) Workers Assigned

Figure 5.15: Comparison between algorithms - Bibsonomy dataset



(a) Skills Trained

Figure 5.16: Comparison between algorithms - Bibsonomy dataset

Unfortunately, we dont see much difference in algorithms for the Bibsonomy dataset. This can be attributed to the fact that they are a lot of unique skills not possessed by the workers and also because of the presence of large number of unique skills present in the tasks.

Chapter 6

Conclusion and Future Work

In this project, we motivated and formulated a new instance of team formation problem. We considered realistic cases, where the tasks can be outsourced or some skills can be trained, if the tasks are not completely covered. We appended these cases into the problem statement. Then, we came up with new algorithms namely, Greedy-Workers, Greedy-Tasks, Extended-Hungarian and Primal-Dual to solve the problem. In addition to this, we also considered some post processing steps to improve the algorithms. We simulated the tasks, skills and workers using the open source data sets of StackExchange and Bibsonomy. And finally, we compared the performance of the algorithms for outsourcing and training cases. We saw that Extended-Hungarian and Greedy-Workers performed better in some cases. But, there was little difference between the performance of the algorithms.

6.1 Training Cost inferred from the Skill Graph

When we want to train workers on certain skills, the training cost can be constant for all tasks or we can infer from the skill graph. The skill graph is generated from the co-occurrences of the skills. From this graph we can assume that if skill "x" and skill "y" co-occur most of the times, then we can easily train a worker on skill "x" if he possesses skill "y". The farther (graph distance) the skill "x" from the skills that the worker possesses, the more expensive the training will be.

A skill graph is a graph $G(V, E)$ with nodes V and edges E. Each node is a skill and the edge indicates whether 2 skills are related or not. From stack exchange we can infer the edges and the edge weights. If two tags (skills) appear in the same question then there is a edge, the number of times 2 skills

appear determines the weight on the edge.

Let x be the number of times they appear together. And the weight on the edge can be any function like, $w = e^{-x}$. Let $z(s, s')$ be the weighted shortest path between skill $s \in V$ and $s' \in V$. Let's also define the distance between a node $s \in V$ and subset of nodes $V' \subseteq V$ as $z(s, V') = min_{s' \in V'} z(s, s')$.

The equation to computing training skills in equation 2.10 can be modified as,

$$Z = \sum_{t \in \overline{T_f}} \sum_{s \in (S_t \setminus (\cup_{w:f(w)=t} S_w))} z(s, \cup_{w:f(w)=t} S_w) \tag{6.1}$$

6.2 Considering the Social Network of the workers

As part of future work one can also consider the social network of the workers. You want to find teams in the social network such that all the tasks are covered and also reduce the communication cost of the formed teams. The problem can be defined as follows: we have a set of workers, each one of them will have a set of skills. We also have set of tasks, with each task requiring a set of skills to complete. In addition, we are also given a social network of workers. The problem is to assign workers to tasks, making sure that all the tasks are covered. Also, the communication cost of the workers assigned to one task is minimized. This can also be thought as a community detection problem. Each community will be assigned to a task. The community that is assigned to a task will cover all the skills of the task.

Bibliography

[1] R. L. Graham, "Bounds on multiprocessing anomalies and related packing algorithms," in *Proceedings of the May 16-18, 1972, Spring Joint Computer Conference*, AFIPS '72 (Spring), (New York, NY, USA), pp. 205–217, ACM, 1972.

[2] H. W. Kuhn, "Variants of the hungarian method for assignment problems," *Naval Research Logistics Quarterly*, vol. 3, no. 4, pp. 253–258, 1956.

[3] J. Munkres, "Algorithms for the assignment and transportation problems," 1957.

[4] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi, "Power in unity: Forming teams in large-scale community systems," in *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, (New York, NY, USA), pp. 599–608, ACM, 2010.

[5] B. Golshan, T. Lappas, and E. Terzi, "Profit-maximizing cluster hires," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, (New York, NY, USA), pp. 1196–1205, ACM, 2014.

[6] N. Garg, T. Kavitha, A. Kumar, K. Mehlhorn, and J. Mestre, "Assigning papers to referees," *Algorithmica*, vol. 58, no. 1, pp. 119–136, 2010.

[7] M. Kargar, M. Zihayat, and A. An, *Finding Affordable and Collaborative Teams from a Network of Experts*, pp. 587–595.

[8] A. Majumder, S. Datta, and K. Naidu, "Capacitated team formation problem on social networks," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, (New York, NY, USA), pp. 1005–1013, ACM, 2012.

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

[10] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi, "Online team formation in social networks," in *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, (New York, NY, USA), pp. 839–848, ACM, 2012.

[11] T. Lappas, K. Liu, and E. Terzi, "Finding a team of experts in social networks," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, (New York, NY, USA), pp. 467–476, ACM, 2009.

[12] C. T. Li and M. K. Shan, "Team formation for generalized tasks in expertise social networks," in *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pp. 9–16, Aug 2010.

[13] 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*, vol. 51, pp. 111–124, May 2004.

[14] E. L. Fitzpatrick and R. G. Askin, "Forming effective worker teams with multi-functional skill requirements," *Computers Industrial Engineering*, vol. 48, no. 3, pp. 593 – 608, 2005. GroupTechnology/Cellular Manufacturing.

[15] J. Vondrák, "Optimal approximation for the submodular welfare problem in the value oracle model," in *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pp. 67–74, ACM, 2008.

[16] B. Lehmann, D. Lehmann, and N. Nisan, "Combinatorial auctions with decreasing marginal utilities," in *Proceedings of the 3rd ACM Conference on Electronic Commerce*, EC '01, (New York, NY, USA), pp. 18–28, ACM, 2001.

[17] V. Mirrokni, M. Schapira, and J. Vondrak, "Tight information-theoretic lower bounds for welfare maximization in combinatorial auctions," in *Proceedings of the 9th ACM Conference on Electronic Commerce*, EC '08, (New York, NY, USA), pp. 70–77, ACM, 2008.

[18] S. Dobzinski and M. Schapira, "An improved approximation algorithm for combinatorial auctions with submodular bidders," in *Proceedings of*

the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm, SODA '06, (Philadelphia, PA, USA), pp. 1064–1073, Society for Industrial and Applied Mathematics, 2006.

[19] N. Craswell, D. Hawking, A.-M. Vercoustre, and P. Wilkins, "P@ noptic expert: Searching for experts not just for documents,"

[20] K. Balog, L. Azzopardi, and M. de Rijke, "Formal models for expert finding in enterprise corpora," in *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, (New York, NY, USA), pp. 43–50, ACM, 2006.

[21] K. Balog and M. De Rijke, "Determining expert profiles (with an application to expert finding)," in *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, (San Francisco, CA, USA), pp. 2657–2662, Morgan Kaufmann Publishers Inc., 2007.

[22] J. Zhang, M. S. Ackerman, and L. Adamic, "Expertise networks in online communities: Structure and algorithms," in *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, (New York, NY, USA), pp. 221–230, ACM, 2007.

[23] H. Deng, I. King, and M. R. Lyu, "Formal models for expert finding on dblp bibliography data," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, (Washington, DC, USA), pp. 163–172, IEEE Computer Society, 2008.

[24] C. S. Campbell, P. P. Maglio, A. Cozzi, and B. Dom, "Expertise identification using email communications," in *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, (New York, NY, USA), pp. 528–531, ACM, 2003.

[25] M. S. Darshan, D. F. M. Gianmarco, and G. Aristides, "Extracting skill endorsements from personal communication data," in *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '16, ACM, 2016.

[26] R. M. Karp, *Reducibility among Combinatorial Problems*, pp. 85–103. Boston, MA: Springer US, 1972.

[27] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of Operations Research*, vol. 4, no. 3, pp. 233–235, 1979.

[28] V. V. Vazirani, *Approximation Algorithms*. New York, NY, USA: Springer-Verlag New York, Inc., 2001.

[29] "Hungarian assignment algorithm." `https://pypi.python.org/pypi/munkres/`.

[30] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. New York, NY, USA: Cambridge University Press, 1st ed., 2011.

[31] "Stackexchange." `http://stackexchange.com/`.

[32] "Stackexchange archive." `https://archive.org/details/stackexchange`.

[33] "Bibsonomy." `https://www.bibsonomy.org/`.

Appendix

Upper Bound for Greedy-Workers algorithm

In this appendix, we will prove the approximation of the greedy algorithm Greedy-Workers in an unweighted setting, where the $d(w,t) = 1, \forall w \in \mathcal{W}, \forall t \in \mathcal{T}$.

Greedy-Workers algorithm is $k.log(N)$ approximate in an unweighted setting. where, k is the number of tasks and N is the sum of sizes of all the tasks

Proof. Let's assume that we can find a solution from the available W workers. And $d(w,t) = 1, \forall t \in T \quad \& \quad w \in W$ Let n_t be the number of skills required to complete task t ($N = \sum_{t \in T} n_t$) and M be the optimal number of workers to complete all the tasks, also considering the constraint that a worker will be assigned to a single task. This proof is inspired from the greedy approximation of set cover in [27]. Before the start of the algorithm, the total number of skills required to cover the skills is equal to $N_0 = \sum_{t \in T} n_t$. In the first step of the algorithm: a worker,task pair (w,t') whose value $\frac{1}{|S_w \cap S'_t|}$ is minimum is chosen and this worker w will be assigned to task t'. And the number of skills that worker w covers is atleast $n_{t'}/M$. Otherwise, the there will more than M workers to complete the tasks. And after the first iteration the number of skills to cover task t' is $\leq n_{t'}(1 - \frac{1}{M})$. Hence, after the first iteration the total number of skills to be covered N_1 is,

$$N_1 \leq \sum_{t \in T, t \neq t'} n_t + n_{t'}(1 - \frac{1}{M}) \tag{1}$$

In the next iteration, lets assume that a worker is assigned to task t''. That worker will cover atleast $\frac{n_{t''}}{(M-1)}$. Then the number of skills left to cover (N_2)

is,

$$N_2 \leq \sum_{t\in T, t\notin\{t',t''\}} n_t + n_{t'}(1-\frac{1}{M}) + n_{t''}(1-\frac{1}{M-1}) \tag{2}$$

$$\text{Since, } n_{t''}(1-\frac{1}{M-1}) \leq n_{t''}(1-\frac{1}{M}) \tag{3}$$

$$N_2 \leq \sum_{t\in T, t\notin\{t',t''\}} n_t + n_{t'}(1-\frac{1}{M}) + n_{t''}(1-\frac{1}{M}) \tag{4}$$

If a second worker is assigned to task t', the number of skills to cover t' is

$$n_{t'} < n_{t'}(1-\frac{1}{M})^2 \tag{5}$$

After i iterations, the number of skills to be covered is,

$$N_i \leq \sum_{t\in T} n_t(1-\frac{1}{M})^{i_t} \tag{6}$$

where, $\sum_{t\in T} i_t = i$
If our greedy algorithm takes f iterations to complete then we will have,

$$\sum_{t\in T} n_t(1-\frac{1}{M})^{f_t} < 1 \tag{7}$$

where, $\sum_{t\in T} f_t = f$
Note that each one of the tasks has to be covered and hence,

$$n_t(1-\frac{1}{M})^{f_t} < 1, \forall t \in T \tag{8}$$

We have,

$$(1-\frac{1}{M})^{f_t} < \frac{1}{n_t} \tag{9}$$

$$e^{\frac{-f_t}{M}} < \frac{1}{n_t} \quad (\text{because}(1-x)^{\frac{1}{x}} = 1/e) \tag{10}$$

$$f_t < M log_e\, n_t \tag{11}$$

$$\text{hence, } f < \sum_{t\in T} M log_e\, n_t \tag{12}$$

$$\text{Also, } f < \sum_{t\in T} M log_e\, N \quad (n_t < N) \tag{13}$$

$$\text{where, } N = \sum_{t\in T} n_t \tag{14}$$

$$\text{Finally, } f < M\, k\, log(N) \tag{15}$$

$$\text{Where, } k = |T| \tag{16}$$

□

Hence, we say that the solution from the GREEDY-WORKERS is atmost $k.log(N)$ times more than the optimal solution.