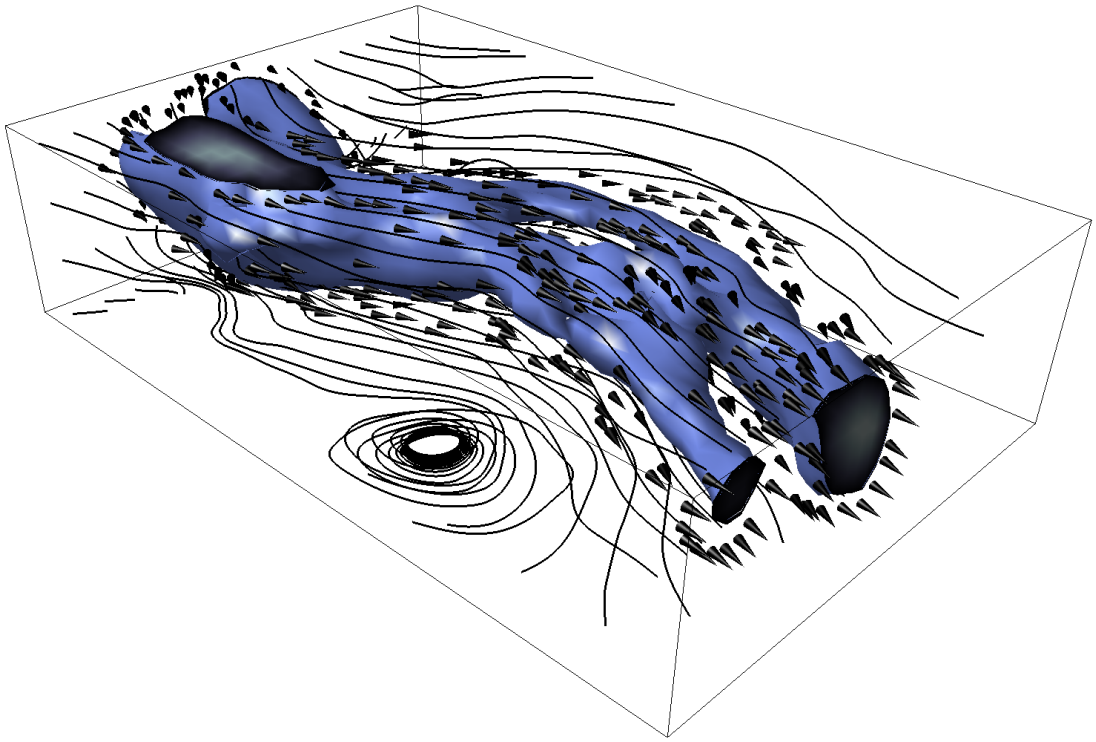


Recommendation system for workers and tasks

Recommending the optimal assignment of workers for tasks

Sebastian Bellevik, Philip Ekman

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017

Recommendation system for workers and tasks

Recommending the optimal assignment of workers for tasks

Sebastian Bellevik, Philip Ekman

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Recommendation system for workers and tasks
Recommending the optimal assignment of workers for tasks
Sebastian Bellevik, Philip Ekman

Supervisor: Christos Dimitrakakis,
Department of Computer Science and Engineering
Examiner: Alexander Schliep,
Department of Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2017

Recommendation system for workers and tasks
Recommending the optimal assignment of workers for tasks
Sebastian Bellevik, Philip Ekman
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis tries to solve the problem with matching workers with tasks, when unknown parameters are involved. Looking at the trend where outsourcing tasks, to previously unknown parties, is becoming more common, a need is definitely there to solve this problem in an efficient way. The problem can be described as a list of workers, each with an unknown list of skills, and a list of tasks, each with a known list of requirements. Any method assigning all tasks to workers, while maximizing the reward given for doing so, must be able to accurately estimate the skills of every worker to provide good results.

To solve this problem when each worker only has a single skill has been shown to be possible with an algorithm called Bounded Epsilon First. This algorithm is used as a starting point for testing data with single-skill workers and single-requirement tasks, before moving on to multi-skill workers and multi-requirement tasks. No real world data was available for multi-skill matching, which is why all experimentation is done on synthetic data, generated uniformly at random. After the first phase, different matching algorithms and methods of rating worker performance were implemented and tested, producing varying results.

Testing all implemented methods on real world data would surely produce interesting results, but overall, the results presented in this thesis show good promise. Our best solution, given time to estimate each worker's skills, give results approaching 95% of the result produces by matching with all parameters known.

Keywords:

Acknowledgements

We would like to express our gratitude to our supervisor Christos Dimitrakakis for the useful comments and remarks through the process of this master thesis. Furthermore we would like to thank our co-supervisor Aristide Tossou for providing support and suggestions every step of the way. Finally, we would also like to thank Jacob Burenstam at Just Arrived for introducing us to the subject and also providing support along the way.

Sebastian Bellevik and Philip Ekman, Gothenburg, May 2017

Contents

List of Figures **xi**

List of Tables **xiii**

Notations **xv**

1 Introduction **1**
1.1 Background . . . 1
1.2 Related work . . . 2
1.3 Goals . . . 3
1.4 General Problem Description . . . 3
1.5 Formal Problem description . . . 5

2 Theory **7**
2.1 Multi-Armed Bandit Problem . . . 7
2.2 Online Task Assignment Problem . . . 7
2.3 Matching algorithms . . . 8
2.3.1 Bounded Epsilon First . . . 8
2.3.2 Epsilon Greedy . . . 9
2.3.3 Upper Confidence Bound . . . 9
2.3.4 Hungarian algorithm . . . 9

3 Experimentation with single-skill matching **11**
3.1 Model . . . 11
3.2 Data . . . 12
3.3 Rating model . . . 12
3.4 Estimation of skills . . . 12
3.5 Algorithms . . . 13
3.5.1 Implementations . . . 13
3.5.2 Optimal . . . 15
3.5.3 Random . . . 15

4 Multi-skill matching **17**
4.1 Model . . . 17
4.2 Synthetic data . . . 17
4.3 Matching . . . 18
4.4 Rating model . . . 19

4.5 Estimation of skills . . . 20
4.6 Selection of workers . . . 21
4.7 Algorithms . . . 22
4.7.1 Modified bounded epsilon first . . . 22
4.7.2 Epsilon greedy . . . 23
4.7.3 Upper confidence Bound . . . 24
4.7.4 Hungarian min-max estimation . . . 25
4.7.5 Random . . . 26
4.7.6 Optimal . . . 26

5 Results **27**
5.1 Algorithm paramaters . . . 27
5.2 Bernoulli rating . . . 28
5.3 Performance compared to optimal . . . 29
5.4 Method comparison . . . 31
5.4.1 Hungarian min-max estimation . . . 31
5.4.2 Epsilon greedy and modified BEF . . . 32
5.4.3 UCB . . . 32

6 Discussion **33**
6.1 Limitations . . . 33
6.2 Estimation of skills . . . 34
6.3 Rating of skills . . . 35
6.4 Virtual tasks . . . 35

7 Conclusion **37**
7.1 Future work . . . 37
7.2 Problem description . . . 37

Bibliography **39**

List of Figures

3.1 Distribution of the different ratings, compared to the total number of verified ratings. 12
3.2 Performance of BEF compared to the optimal solution, depending on the number of tasks provided. Optimal is always 100% and the closer BEF gets to 100% the better are the results. 15

4.1 Performance of the modified BEF solution, comparing the old skill estimation that uses the average of all ratings, and the new model that uses min-max estimation. 21
4.2 Performance of Epsilon Greedy compared to the optimal solution, with regards to ε. Used to find the optimal ε. 23

5.1 Success rate of Hungarian min-max estimation with regard to ε, i.e. the probability that the correct skill level rating is given. 28
5.2 Performance of all algorithms compared to the optimal solution. Used to see if any of them converge towards the optimal solution, given enough tasks. Maximum number of skills per worker and requirements per task is 3. 29
5.3 Performance of all algorithms compared to the optimal solution. Used to see if any of them converge towards the optimal solution, given enough tasks. Maximum number of skills per worker and requirements per task is 1. 30

List of Tables

5.1 Parameters used for testing all implemented methods. 27
5.2 Worker and task example data, workers with 1 skill and tasks with 1 requirement. 31
5.3 Worker and task example data, workers with 2 skills and tasks with 2 requirements. 31
5.4 Matching table for 2 workers and 2 tasks, workers with 1 skill and tasks with 1 requirement. The only greedy optimal assignment possible. 31
5.5 Matching table for 2 workers and 2 tasks, workers with 2 skills and tasks with 2 requirements. First example of a greedy attempt at an optimal solution. 31
5.6 Matching table for 2 workers and 2 tasks, workers with 2 skills and tasks with 2 requirements. Second example of a greedy attempt at an optimal solution. 31

6.1 Example of correct worker skill level estimation. 34
6.2 Example of incorrect worker skill level estimation. 34

Notations

Abbreviations

BEF	Bounded Epsilon First
MBEF	Modified Bounded Epsilon First
UCB	Upper Confidence Bound
HME	Hungarian Min-max Estimation
MABP	Multi-Armed Bandit Problem
OTAP	Online Task Assignment Problem

Variables

$r_{w,t}$	The reward for worker w completing task t
$\|\|A\|\|$	The number of elements in set A
$\|x\|$	The the absolute value of x

Notations

Worker skill	A skill e.g Java or social capability
Worker skill level	Measurement of how good a worker is at a particular skill
Required skill	A required skill for a task e.g Java or social capability
Required skill level	Measurement of how good a worker is required to be

1
Introduction

This thesis will explore different versions of the assignment problem, where a group of workers has to be assigned to a number of tasks in an optimal way. First a general problem description will be presented along with a clear goal to fulfill. Chapter 2 will then, in detail, explain the theory of the problem itself as well as descriptions of several methods that will be implemented to try and solve it. The following two chapters will explain different types of workers and tasks used for testing the methods described in this thesis. Chapter 3 will focus on experimentation where workers have only one skill and tasks have one requirement and chapter 4 will cover the more complex problem where workers can have multiple skills and tasks multiple requirements. The chapter after that will show the results of all implemented methods as well as a comparison of how they fare against each other and against a, theoretical, optimal solution. Finally, in the last two chapters, the results, and the methods used to achieve them, will be discussed and a conclusion will be reached about their probable usability in real world scenarios.

1.1 Background

A trend that has been observed over the recent years is that many software companies are outsourcing tasks to workers of whom they have no previous experience in using [1]. Instead of outsourcing tasks to known, or previously used, firms, they are often outsourced to individual workers through platforms such as *Mechanical Turk* [2]. On the market today there are several outsourcing platforms, that often outsource simpler tasks that can be completed during a shorter period of time than the average time span and without a specific set of high level skills [3].

As these services continue to grow in number and active users [4], the need to find competent workers for more complex tasks, which require a more specific skill set, will increase. With an increasing number of workers and tasks there will be a greater need for automatic, or semi-automatic, recommendation methods. Otherwise, too much time will be spent manually keeping track of each worker's performance and trying to find the best matches among the myriad of available workers with different skills, and tasks with different requirements.

There exists numerous methods for optimal matching of workers and tasks when all variables are known [5]. However, with unknown variables present, a level of uncertainty is introduced that complicates the problem. Additionally, each task may require a specific set of skills, each with a certain amount of experience, to be

completed. Some platforms allow their workers to enter their own perceived skill levels. However, the company assigning the task has no way of knowing any worker's degree of honesty. Assigning simple tasks that almost anyone can do eliminates this problem. With the increasing demand for workers with certain skills, there is a need for an efficient way of eliminating the element of uncertainty, turning the problem into a standard matching problem. In other words, there is a need for a way of estimating the real skills of potential workers, while not wasting too many important tasks.

1.2 Related work

There have been several occasions where large companies have crowdsourced tasks to workers they have no prior knowledge about. Examples are *Waze* [6], where users report traffic information, *McDonald's* [7], where their customers competed to design the best burger, and *Greenpeace* [8], where people around the world competed to come up with the best slogan. These are all examples of crowdsourcing of tasks that do not require a specific skill set. In recent years, with increasing popularity of crowdsourcing, researchers have investigated if there could be a way to crowdsource more difficult tasks with the need of a specific set of skills. This is called the expert crowdsourcing problem.

An example of this is when Ho and Vaughan [9] were assigning workers, with different skill sets, to heterogeneous tasks via the crowdsourcing market *Amazon Mechanical Turk*. Their goal was to assign workers to tasks that required a specific set of skills, where each of the workers have unknown skill levels, which are to be learned by the algorithm. Because their problem was modelled as the Online Task Assignment Problem, workers arrive online and must be assigned to a task on arrival. The typical challenges for such a problem is to estimate each of the workers skills while maintaining a good result, this is referred to as a exploration-exploitation trade-off. They implemented an algorithm called Dual Task Assigner which uses an explicit exploration phase where they are estimating workers skills and an exploitation phase where they are maximizing the total reward.

Another group of researchers that have tried to solve the expert crowdsourcing problem was a team from *University of Southampton* [10]. They modelled their problem as the Multi-armed bandit problem, where an agent stands in front of a row of slot machines and will choose which machines to play. Each machine has a unique cost and gives the agent a reward when played. Their proposed algorithm is called the Bounded Epsilon First algorithm, which utilizes a portion of a total budget to estimate the skills for all workers, and later maximizes the utility within an exploitation phase. They compared their results with an hypothetical optimal algorithm, that knew the workers actual skills, and they approached about 90% of the optimal solution with an increasing budget.

1.3 Goals

The primary goal when starting this thesis was to help ease the matching process for workers and tasks for the platform *Just Arrived* [11]. *Just Arrived* is a platform for job matching, focusing on immigrants who recently arrived to Sweden. New members on the platform can enter their own perceived skill levels when signing up. Before applying for a task they get interviewed by an employee at *Just Arrived* who will try to assess some of their skills. When the initial process is completed, a worker can apply for any task and it is up to the company providing the task to select the best applicant. Also, upon task completion the company can rate the performance of the worker. The way to make this process more straightforward is to create a method for estimating the real skill levels of every worker as well as recommend the best worker for each task. That way, neither the company nor *Just Arrived* will have to find the best combinations of workers and tasks from some seemingly random list. Instead, for each set of tasks, the optimal combination can be presented, making the matching process both faster and easier in terms of time spent by humans.

To solve this problem, for the specific case of *Just Arrived*, a large data set would be provided containing numerous workers and tasks. Analyzing this data would provide relevant information about the distribution worker skills, task requirements, and how well the matching works today. However, the data was lacking in both quality and quantity, making it effectively useless for basing any methods upon. The amount of data is sure to grow as the platform grows, but pretty early on in this thesis work a decision was made to form a more general goal. The main goal instead became to construct a general method to be used by any group of people trying to match workers and tasks, where the skills of workers are an unknown factor.

With the above information, the goal can be reformulated to the following: Create a method of matching a set of workers, each with multiple unknown skill levels, with a set of tasks, each with multiple requirement levels. This should be done independently of the distribution of skill levels and requirement levels among them. Also, an additional goal is to construct an estimation method for finding the unknown true skill levels of the workers.

1.4 General Problem Description

The problem consists of a number of workers and a number of tasks to be matched in an ideal way. Every task has a number of requirements, each with a certain requirement level, representing what is needed from a worker to complete it, and every worker has a number of skills, each with a certain skill level, representing what they can do. While the tasks requirement levels are known, the workers skill levels are not, making them something a potential algorithm will have to figure out to make optimal recommendations. The ideal setting, to make as good estimations as possible about each worker's skill levels, would be where there are many more tasks available than there are workers. However, this is not always the case in a real world setting, where the number of available workers can be much higher than the number

of available tasks. To counter this, the assumption is made, in this thesis, that virtual tasks could be introduced to fill the gap. In a real setting virtual tasks can be anything from unimportant tasks that companies are willing to waste, to simple tests, all with the purpose of estimating the workers skill levels. For this thesis, and the experimentation done, no clear distinction is made between virtual tasks and real ones, as that is something that would be completely up to the task provider, or some other service that they use. For testing purposes it is simply assumed that there are more available tasks than workers.

To make the assignment process more realistic the tasks are split into blocks, where each block contains the same number of tasks as the number of workers available. Every time matchings occur, the tasks within each block can only be assigned to one worker, and a worker can only perform one task. This is made to simulate workers being unavailable while performing a task.

Matching a worker with a task gives a nonnegative reward, which represents how well the worker performed. This reward is calculated by comparing the skill levels of the workers with the requirement levels of the task, where a skill level higher than the corresponding requirement level means there is a higher probability of success. In a real world setting, this process would preferably be performed by the task provider, but since all data used is synthetic, it has been automated. Given the above information, the goal of any solution is to maximize the total reward gained from assigning all the tasks available to workers. When all variables in this type of problem are known, there exist solutions to give an optimal result, for example the Hungarian Algorithm[18], but since the skill levels of workers are unknown, an element of uncertainty is introduced. Because of this, the problem itself can be split into two separate parts, where the first part is to try and estimate the workers real skill levels and the second part is to apply any optimal matching algorithm.

1.5 Formal Problem description

There is a set of workers W, and a set of tasks T. For each worker $w \in W$ there is a set of skills M each with an unknown skill level μ, and for each task $t \in T$ there is a set of requirements Λ, each with a requirement level λ.

$$\forall w \in W : M = [\mu_1 ... \mu_k] \wedge \forall t \in T : \Lambda = [\lambda_1 ... \lambda_k], \quad \mu, \lambda \in [0, 1], \quad k > 0 \tag{1.1}$$

The tasks are split into blocks $b \in B$, where the size of each block, b, equals the size of W and a task can only exist in a single block. However, the last block might be smaller in size than W, which is permitted as a special case, the method should still work.

$$\forall b \in B : ||b|| = ||W|| \wedge b_m \neq b_n \wedge |B| = \left\lceil \frac{||T||}{||W||} \right\rceil \tag{1.2}$$

Matching a worker and a task gives a nonnegative reward $r_{w,t}$, representing how well the worker performed the task. The reward is calculated with a function $f()$, which compares each of the worker's skill levels with the corresponding requirement levels of the task.

$$r_{w,t} = f(M_w, \Lambda_t) \tag{1.3}$$

Given the above information, the goal is to maximize the cumulative reward, R, given that all tasks are completed. A worker can only be assigned one task within each block, meaning every worker will perform the same amount of tasks. This is represented by the set U, where each element $u_n \in U$ represent how many times $w_n \in W$ has been used.

$$R = \sum_{b \in B} \sum_{t \in b} r_{w \in W, t}, \quad \forall u \in U : u_n = u_m \tag{1.4}$$

2
Theory

This chapter will describe the theory behind all the methods implemented during the thesis. As mentioned in section 1.3 the goal of this thesis is to solve the problem of matching a set of workers with a set of tasks, while maximizing the cumulative rewards. This can be referred to as the *Assignment Problem*. Within this thesis, each of the problems with be modelled using parts from both the *Multi-armed bandit problem* and the *Online Task assignment problem*. For the *Multi-armed bandit problem* the goal is to find the best assignments in order to maximize the reward, and the *Online Task assignment problem* uses an agent that continuously matches a set of workers with a set of tasks.

2.1 Multi-Armed Bandit Problem

The Multi-armed bandit problem [12] is described as a gambler in front of a row of slot machines, that has to decide which machines to play, how many times, and in which order. These slot machines typically represents workers, and when the gambler pulls an arm of a slot machine this means that the worker performs a task. Each machine has a fixed cost that the gambler is required to pay to be able to pull that machines arm. After each pull of an arm, the machine gives a random reward from some probability distribution. Initially, the gambler starts with a fixed budget B and the objective of the gambler is to maximize the total reward given from all arms pulled, while not exceeding B.

2.2 Online Task Assignment Problem

The Online Task Assignment Problem [9] is a special case of the Assignment Problem. The Assignment Problem is defined as two sets of equal size, one set of workers and one set of tasks. Every worker has to be assigned to exactly one task and every task has to be performed by exactly one worker. Assigning a worker to a task has a cost, and to solve the problem the total cost has to be minimized.

With the Online Task Assignment Problem, a requester has a set of tasks to be completed and a total budget that can not be exceeded. Contrary to the normal Assignment Problem, there is no fixed set of workers, but instead the workers arrive continuously and are assigned a task upon arrival. Each new worker that arrives has an unknown set of skills, and generally workers with higher levels of skill produce better results. Though the workers skill levels are initially unknown, they can

be estimated by assigning them to tasks and analyzing the results. Estimating a worker's skill levels increases the probability of selecting the optimal worker for a given task, when that worker can perform a task again in the future. When enough workers are available for selection and the estimation of their skill levels are closer to their true skill levels, the problem can basically be converted to the offline version, i.e. the Assignment Problem, which can be solved using the Hungarian Algorithm. Also, instead of trying to minimize the total cost, the goal is to maximize the total reward given by assigning workers to tasks.

2.3 Matching algorithms

2.3.1 Bounded Epsilon First

The Bounded Epsilon First (BEF) [10] is an algorithm that tries to maximize the reward, given a set of workers, a set of tasks, and a fixed budget. The workers and tasks are modeled as the Multi-Armed bandit problem, where each worker is a slot machine and pulling the arm of a certain slot machine corresponds to assigning that worker to a specific task. Pulling the arm of a worker has a fixed cost and gives a reward. Each worker also has a limit that decides how many times they can perform any task, i.e. how many times their arm can be pulled.

The algorithm consists of an exploration phase and an exploitation phase. A certain part, $\varepsilon \in [0, 1]$, of the budget, B, is dedicated to the exploration phase. During that phase, each worker is assigned a task and is paid, in turn order until εB is depleted. This phase is used to estimate the workers skill levels. These skill levels are later used to determine which worker to use for a certain task during the exploitation phase.

The exploitation phase is used to select the best workers available for each task. Because the algorithm is unaware of the true skill levels for the workers, it uses the estimated skill levels obtained through the exploration phase. During this phase the bounded knapsack algorithm [13] is used to select the best available worker for a task. First, the workers are sorted by their density which is defined as:

$$density_i = \frac{\hat{\mu}_i}{c_i} \tag{2.1}$$

where $\hat{\mu}_i$ is the estimated skill level for worker i and c_i is their cost. After sorting the list of workers, the algorithm pulls the arms of the worker with the highest density until the limit for that worker is reached. This is repeated until the rest of the budget, 1-εB, is depleted.

2.3.2 Epsilon Greedy

This approach is similar to BEF, but there is no such clear distinction between separate phases [14]. For each given task, a random worker is selected with a probability of ε and with a probability of $1 - \varepsilon$ the best worker for the task is selected. This results in a continuous mix of exploring and exploiting. A version of the Epsilon Greedy, where ε is steadily decreasing over the course of the matching process, is sometimes called Epsilon Decreasing. The decrease, referred to here as drop rate, is a value $0 < \mathrm{d} \leq 1$, which is multiplied by ε at each turn, resulting in the formula:

$$\varepsilon_k = d^k * \varepsilon_{k-1} \tag{2.2}$$

Where ε_k is the current epsilon, d_k is the drop rate to the power of the number of tasks completed and ε_{k-1} is the epsilon during the previously performed task. As more tasks are performed, the probability to pick a random worker should decrease, because the algorithm should have learned something about the available workers. Finding a good start value and drop rate for ε can be a difficult task, this will be discussed further in section 4.7.2.

2.3.3 Upper Confidence Bound

This algorithm is similar to both BEF and Epsilon Greedy in that it continuously updates and chooses workers for tasks, and uses a pre-exploration phase where each worker is tested once, to give the algorithm a starting point [15]. When a worker has completed a task the estimated skill levels $\hat{\mu}$ are updated. After the initial exploration phase is done, for each task a worker is selected that maximizes the following formula:

$$\bar{x_j} + \sqrt{\frac{2ln(n)}{n_j}} \tag{2.3}$$

Where j is the current worker, $\bar{x_j}$ is the average reward for worker j, n is the total number of tasks performed and n_j is the number of tasks performed by worker j.

2.3.4 Hungarian algorithm

The Hungarian Algorithm [16] [17] is an algorithm used to solve the assignment problem. The algorithm uses a cost matrix, C, of size $n \times n$, where n is the number of workers and tasks, and each entry in the matrix is nonnegative. The entry $C_{i,j}$ represents the cost of assigning worker i to task j. Given this matrix the algorithm finds the optimal way of assigning a worker to each task. For the specific problem to be solved in this thesis the goal is to maximize the total reward, where the reward is any value between 0 and 1. Hence, for each entry in the constructed matrix the reward is subtracted from 1, giving a matrix that can be solved with the standard Hungarian Algorithm.

The Hungarian Algorithm uses the following four steps to find the optimal sets of matches [18]. First, the lowest entry of each row is found and subtracted from each

element in that row. This will ensure each row contains at least one zero. Second, repeat the same procedure, but with columns instead of rows. Third, all zeroes in the matrix are covered using the minimum number of horizontal and vertical lines. If n lines are required, an optimal assignment can be made with the zeroes. If less than n lines are required, the last step is to find the smallest entry in the matrix that is not covered by any line from the third step. The value of that entry is then subtracted from all elements not covered by the lines from step three and added to all elements that are covered twice. Many times, all steps are not needed, and the optimal solution might be found after any of the aforementioned steps.

3

Experimentation with single-skill matching

Due to the lack of data provided, the first experiments were conducted on a set of data from the project *Get Another Label* [19]. The format of the data provided with the *Get Another Label* project differ much from the format of the data provided from *Just Arrived*. The former only uses workers with one skill level and tasks with one requirement level, while the latter use workers with multiple skill levels and tasks with multiple requirement levels. Because of this, some experimentation was needed in order to get a good starting point for the method implementations. Get Another Label was a project that tried to categorize websites as one of the following; G (General), PG (Parental Guidance), R (Restricted) or X (X-rated). It resulted in a large set of websites as well as data from Mechanical Turk where workers tried to categorize these websites, some of which had a known true category. This project is used to make sure that the initial implementation of the BEF algorithm is done correctly, since no real data in the form needed was provided.

3.1 Model

The model for the implementation consists of tasks, $t \in T$, and workers $w \in W$. Tasks, in this case, are websites, with an unknown real rating, to be categorized. Workers have an unknown skill level μ, representing their ability to categorize websites, as well as a known public skill level $\hat{\mu}$, representing how good the algorithm think they are at categorizing websites. Each worker also has a cost c that needs to be taken into consideration, since there is a total budget B that cannot be exceeded.

This problem is modeled as the Multi-armed bandit problem, where an agent will pull the arms that maximizes the total reward given from performed tasks. The slot machines represents the workers and pulling their arm is equivalent to letting a worker rate a specific website. Each worker has a unique cost c, based on their skill level μ. This was done to try to simulate a real world setting where better workers, with a high probability, are more expensive.

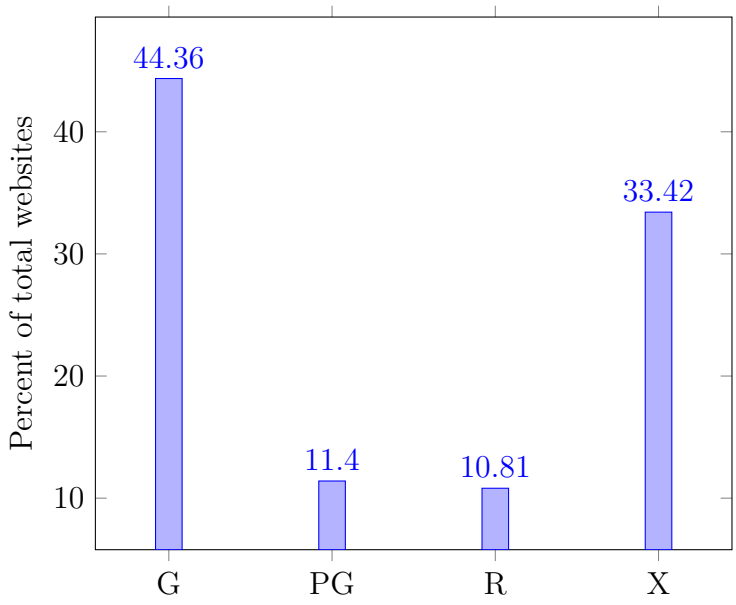
$$\forall w \in W : c_w = K * \mu_w \tag{3.1}$$

where K is an arbitrarily chosen base cost.

3.2 Data

The Get Another Label project used a number of data sets, but for this experiment only the largest one was selected. This data set contained 11040 websites, rated by Mechanical Turk workers, of which 1517 had been assigned a verified official rating, and a list of 825 workers who had performed the rating. This data was used to test the implementation of the algorithm, but also as a base for generating much larger data sets.

Figure 3.1: Distribution of the different ratings, compared to the total number of verified ratings.



3.3 Rating model

When the agent pulls an arm, i.e. assigns a task t, to a worker w, a reward $r_{w,t}$ is given, which value is between 0 and 1 and depends on the worker's ability to rate websites, their skill level μ. Their skill level is used as a probability to guess correctly, giving the model a certain degree of randomness. A worker might still fail, even with a skill level of 0.99, and a worker with a skill level of 0.01 might still succeed, with suggesting the correct rating.

3.4 Estimation of skills

The estimation of each worker's skill level $\hat{\mu}$, is the number of ratings the worker have received divided by the sum of the accumulated ratings. Since the skill level of a worker is the probability to succeed, given enough tasks the true rating should be found.

3.5 Algorithms

3.5.1 Implementations

The the first algorithm used in this thesis is BEF, which is also the only algorithm used for the data from Get Another Label. This is because it has been proven to give good results for this type of problem [12]. The theory behind BEF has been described in Section 2.3.1, and the pseudocode of the implementation can be found in Algorithm 1 below.

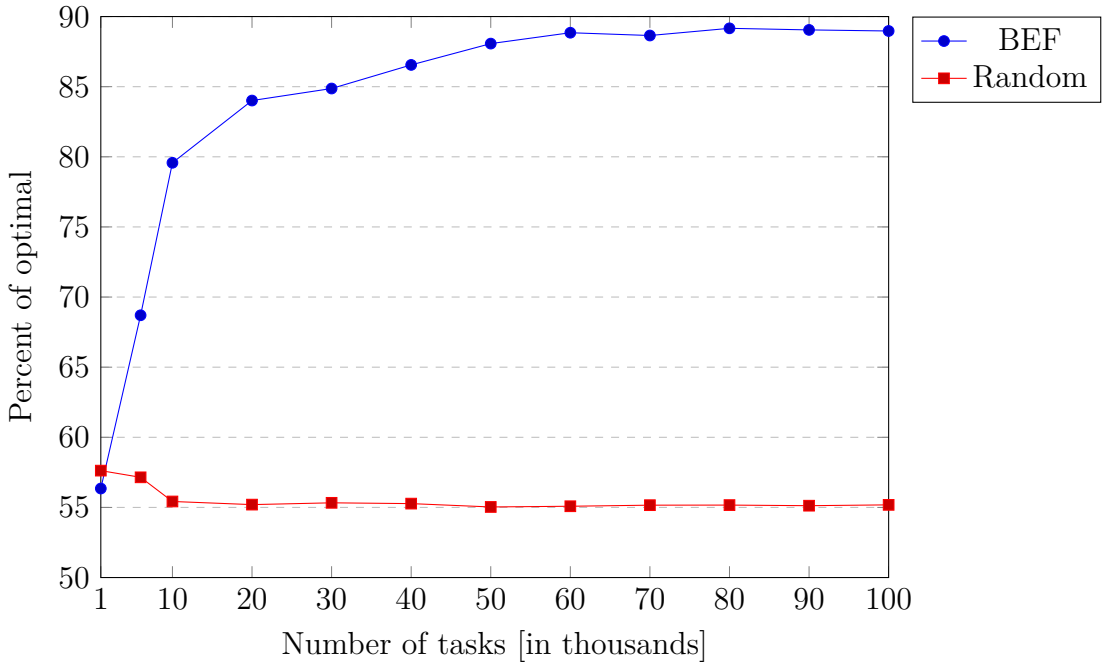
Algorithm 1: Bounded Epsilon First

input : T: a set tasks
W: a set of workers
B: the total budget to be spent
ε: the percentage of the budget used for exploration
Output: r: the average reward received by the workers

procedure *BEF(T ,W ,B ,ε)*:
  let $T_{explore}$ be exploration tasks
  let $T_{exploit}$ be exploitation tasks
  let B be $B - \varepsilon B$
  (Assign workers one by one until $T_{explore}$ is empty)
  for *task* $t \in T_{explore}$ **do**
    if *all workers assigned* **then**
      let w be first worker
    else
      let w be the next worker
    end
    assign w to t
    rate performance of w
    estimate skills of w
  end for
  sort workers by $\frac{\text{average worker performance rating}}{\text{worker cost}}$
  while B *is not empty* **do**
    let w be the best worker for task $t \in T_{exploit}$
    assign w to t
    rate performance of w
    estimate skills of w
    let B be $B - w_{cost}$
    remove t from $T_{exploit}$
    if w *cannot work more* **then**
      remove w from W
    end
  end while
  return average performance rating of all workers
end procedure

After implementation, the results of the BEF was compared to the optimal solution, to find how the results changed depending on the number of websites provided. As seen in plot 3.2, the results of BEF approach about 90% of the optimal solution when more websites are provided. This result was achieved by running both the optimal solution and BEF for each set of tasks and simply dividing the result of BEF by the result of the optimal solution. When running the algorithm, the epsilon used was 0.15, since that was the value found to give the best results according to the creators of the algorithm. Some experimentation during this thesis also showed that this was the case.

Figure 3.2: Performance of BEF compared to the optimal solution, depending on the number of tasks provided. Optimal is always 100% and the closer BEF gets to 100% the better are the results.



3.5.2 Optimal

The optimal solution is used mainly as a benchmark, used to show how close other algorithms can get to it. If the true rating and cost is known, the true density can be calculated and a true optimal solution can be found using the knapsack algorithm.

3.5.3 Random

Similarly to the optimal solution, the random solution is used for comparison. To get a better picture of how well the real algorithm performs, it is compared both to the optimal and the random solution. This solution simply selects a random available worker for each website to rate.

4
Multi-skill matching

One problem with the model implemented in the Get Another Label system described in section 3 was that it differed to much from the model described by Just Arrived. First of all, in Just Arrived's model the workers have multiple skills, with skill levels in each one, instead of just one skill. The same holds for the tasks, which have multiple requirements with a minimum requirement level for every requirement. Another major difference is that there are no varying costs for when workers are completing tasks, nor is there any total budget that can't be exceeded. Instead, the goal is to complete all the given tasks while maximizing the total rating given from each completed task. The cost is decided by the task, and since all tasks have to be assigned, the cost aspect can be omitted completely.

This problem is also modeled as a Multi-armed bandit problem. However, because of the absence of a budget as well as the removal of costs for each worker and the introduction of multiple skills, a few changes in the model was made. It can also be described as a less complex version of the Online Task Assignment Problem, namely the Offline Task Assignment Problem, where there is a static list of workers and tasks instead of workers arriving dynamically. The possible impacts that this will have on the results will be discussed in section 6.1.

4.1 Model

This model consists of a set of workers W, and a set of tasks T. Every worker $w \in W$ has a set of skills, each with an unknown true skill level μ, and a known estimated skill level $\hat{\mu}$. The estimated skill levels will gradually be updated by the algorithm and the goal is for $\hat{\mu}$ to converge towards μ. Also, every task has a set of requirements, each with a requirement level λ. Both skill levels and requirements levels are represented by a value between 0 and 1.

4.2 Synthetic data

As described in section 1.3, the quantity and quality of the data received for this thesis was lacking, and therefore a decision had to be made early on to focus on creating synthetic data. The basic structure of the workers and tasks present in the data from Just Arrived was preserved however, but since no conclusions could be drawn from analyzing the data itself, many parts of the synthetic data was created uniformly at random. Since nothing can be said about the real world distribution

of various aspects of the workers and tasks, the main focus is instead to find an optimal way to estimate the workers skill levels.

Working purely with synthetic data requires some research about the format of the relevant real world data, as well as a lot of testing. Another aspect preserved from the Just Arrived data was the cost for utilizing a worker is set by the task, which means that both the specific worker costs, and total budget can be omitted from any implemented algorithm. To speed up computing times and to make analyzing the data smoother, some restrictions were made:

- The sets of workers skills and tasks requirements are always of equal size, which, for the duration of this thesis, is 3. This is a soft restriction however, since both skill levels and requirement levels can be 0, meaning the effective number of skills or requirements can be lower.
- The generated skill levels and requirement levels are created uniformly at random, but are upon creation rounded to the closest $\frac{1}{5}$, i.e. one of $[0.0, 0.2, ..., 1.0]$. This was done to closer mimic real world data, since it would be an unrealistic expectation that humans would be able to estimate skill levels and requirement levels to any of an infinite number of levels between 0 and 1.
- The number of tasks available is equal to or greater than the number of available workers. With more workers than tasks, enough could not be learned about each individual worker, without any prior knowledge.
- All estimated skill levels are initiated to 0.5, which is the average level when nothing is known.

4.3 Matching

When matching workers with tasks, two different approaches were tested during experimentation. The first one, called unrestricted matching, is when no restrictions are put on the worker in terms of how many tasks they can perform at any time. Although this might not be applicable in many real world settings, it allowed for quick testing of the implementations. When the tasks provided are simple and take no more than a couple of seconds of a workers time, like with the Get Another Label project, this might be a realistic approach. A worker can be assigned a large number of tasks to be completed one after the other, without a real time restriction. However, in many real world scenarios tasks will take longer to complete, which is why the second method, block matching, was implemented. With this method, tasks are split into blocks with the same size as the number of workers, and within each block a worker can only perform one task, and a task can only be assigned to one worker.

4.4 Rating model

After each completed task, the information about the worker known by the algorithm is updated. For each skill the workers possess, a rating is provided, $e \in E_{w,t}$, for each skill, depending on the difference between the worker skill level and the corresponding task requirement level. A skill level equal to or above the requirement level means a higher probability of success. This rating model does not award a worker more for being above the requirement level than for just having an equal skill level. This decision was made with the assumption that task providers have a good understanding of what skills and skill levels are needed to complete a given task. For example, a task with a requirement level of 0.6 in Java, should be able to be completed by a worker with a skill level of 0.6 in Java, a greater skill level should not be needed. The rating of each skill is calculated according to formula 4.1. The ratings of each skill are then used to calculate a single value rating a, for the task completion according to formula 4.2. Also, this single value rating is used to measure the overall success of any implemented algorithm.

$$e_i = \begin{cases} Bernoulli(1-\varepsilon), & \mu_i \geq \lambda_i \\ Bernoulli(\varepsilon), & otherwise \end{cases} \tag{4.1}$$

Where e_i is the rating for skill i, μ_i is the worker's skill level for skill i, λ_i is the task's requirement level for requirement i, and ε is an arbitrarily chosen, low value, variable described more below.

$$a_{w,t} = \frac{\sum_{e \in E_{w,t}} e}{||E_{w,t}||} \tag{4.2}$$

Where $a_{w,t}$ is the rating given to worker w for completing task t and $E_{w,t}$ is the set of ratings for each of the worker's skills according to formula 4.1.

The variable ε was introduced to represent a certain amount of uncertainty in a real world scenario. It can be used as a single value to represent a number of different factors during the work process or the rating process. A worker with skill levels above the requirement levels can still fail, for example because of a bad day or a bad night's sleep. When rating, the task provided can make an incorrect assumption based on the result. Similarly, a worker who is normally not good enough for a given task, can still succeed, or the rating can be incorrect because of the task providers inexperience within the field. The choice of ε will be discussed further in section 5.2.

4.5 Estimation of skills

The same approach used in section 3.4 was tested for estimation of multiple skill levels per worker. For each skill, a set of ratings was saved, and the estimated skill level was the average of all ratings for that particular skill. However, with multiple skills the estimation for each skill level fluctuates a lot, and the results are worse than when used with single skill matching.

To solve that problem, a new method of estimation was introduced, called min-max estimation. Instead of adding every rating to sets and using the average rating as estimation, each worker gets a new set, with the same size as the set of skills, that contains pairs of values. The elements in the pair represents the minimum and maximum values that the algorithm believes the skill level could be. After each performed task, the algorithm uses the result to decide how to update the minimum and maximum estimations and then the estimation for that particular skill level is the average of the minimum and maximum values. If the rating received for skill i is 1, the minimum is updated as seen in formula 4.3 and if the rating received for skill i is 0, the maximum is updated as seen in formula 4.4.

$$s_i^{min} : \begin{cases} t_i, & r = 1 \\ s_i^{min}, & otherwise \end{cases} \tag{4.3}$$

$$s_i^{max} : \begin{cases} t_i, & r = 0 \\ s_i^{max}, & otherwise \end{cases} \tag{4.4}$$

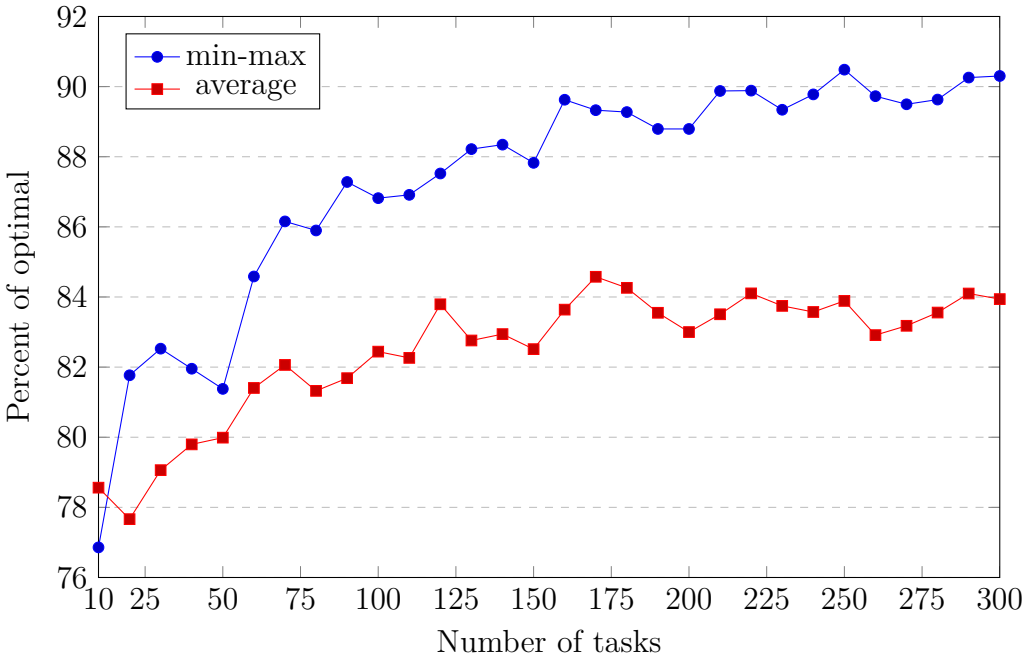
Where s_i^{min} and s_i^{max} are the minimum and maximum estimations for skill i respectively, t_i is requirement i for task t, and r is the rating received for skill i. The estimation for a particular skill is thus continually updated and set according to the following formula:

$$s_i^e : \frac{s_i^{min} + s_i^{max}}{2} \tag{4.5}$$

Where s_i^e is the estimated skill level for skill i.

A comparison of the two rating methods can be seen in the graph below, and when comparing both to the optimal solution it is clear that using min-max estimates the true skill levels both better and faster.

Figure 4.1: Performance of the modified BEF solution, comparing the old skill estimation that uses the average of all ratings, and the new model that uses min-max estimation.



4.6 Selection of workers

Selecting the best worker for a single task is another aspect that gets more complicated with multiple skills and requirements. Also, matching using blocks forces any selection of the best worker to take into account that choosing a worker that is a lot better than needed might waste them, making them unavailable for more difficult tasks later on. To solve this problem, the best worker for a task is defined as the worker with the highest number of skill levels equal to or above the corresponding requirement levels. The worker that maximizes the following formula is first selected:

$$\sum_{i=1}^{k} \begin{cases} 1, & \hat{\mu}_i \geq \lambda_i \\ 0, & otherwise \end{cases} \tag{4.6}$$

Where k is the number of skills for each worker, $\hat{\mu}_i$ is the estimated skill level for skill i, and λ_i is the requirement level for requirement i.

If there are more than one worker who fulfills that criteria, the worker with the lowest total difference, in skill levels against requirement levels, is selected. The worker that minimizes the formula 4.7 is chosen. This means that the worker selected for a task should be *just good enough* to be able to complete it, so that workers a lot better than required can be used later.

$$\sum_{i=1}^{k} |\hat{\mu}_i - \lambda_i| \tag{4.7}$$

This whole process is illustrated, in pseudocode, in Algorithm 2 below.

Algorithm 2: Worker Selection

input : t: the task to be assigned
W: a set containing workers
Output: w_{best}: The best worker for the provided task

```
procedure WorkerSelect(t ,W ):
    let A be w ∈ W with most skill levels above requirement levels of t
    if ||A|| is 1 then
        let w_best be w ∈ A
    else
        let w_best be w ∈ A with lowest total difference in skill levels compared to
          requirement levels
    end
    return w_best
end procedure
```

4.7 Algorithms

All the matching algorithms described in 2.3 were implemented with regards to the model used for multi-skill matching, starting with a modified version of BEF. Any changes made to the implementation of each respective algorithm are described here. The selection process described in 4.6 is used for all greedy approaches, i.e. the modified BEF, Epsilon Greedy, and UCB. Although some algorithms use more information to select the best worker for any given task, the base which all of the use is that particular measurement for *best* worker.

4.7.1 Modified bounded epsilon first

As described in the beginning of chapter 4, the notion of specific worker costs and a total budget were removed for the purpose of this multi-skill matching problem. Therefore an implementation used would not follow the rules of the orgininal BEF, which is why the modified BEF is introduced. The basic structure used by BEF is still used, where there is a set of workers W, and a set of tasks T and the goal is still to maximize the total reward R. Without costs and a budget, ε is instead used to directly decide the number of tasks used for exploration according to:

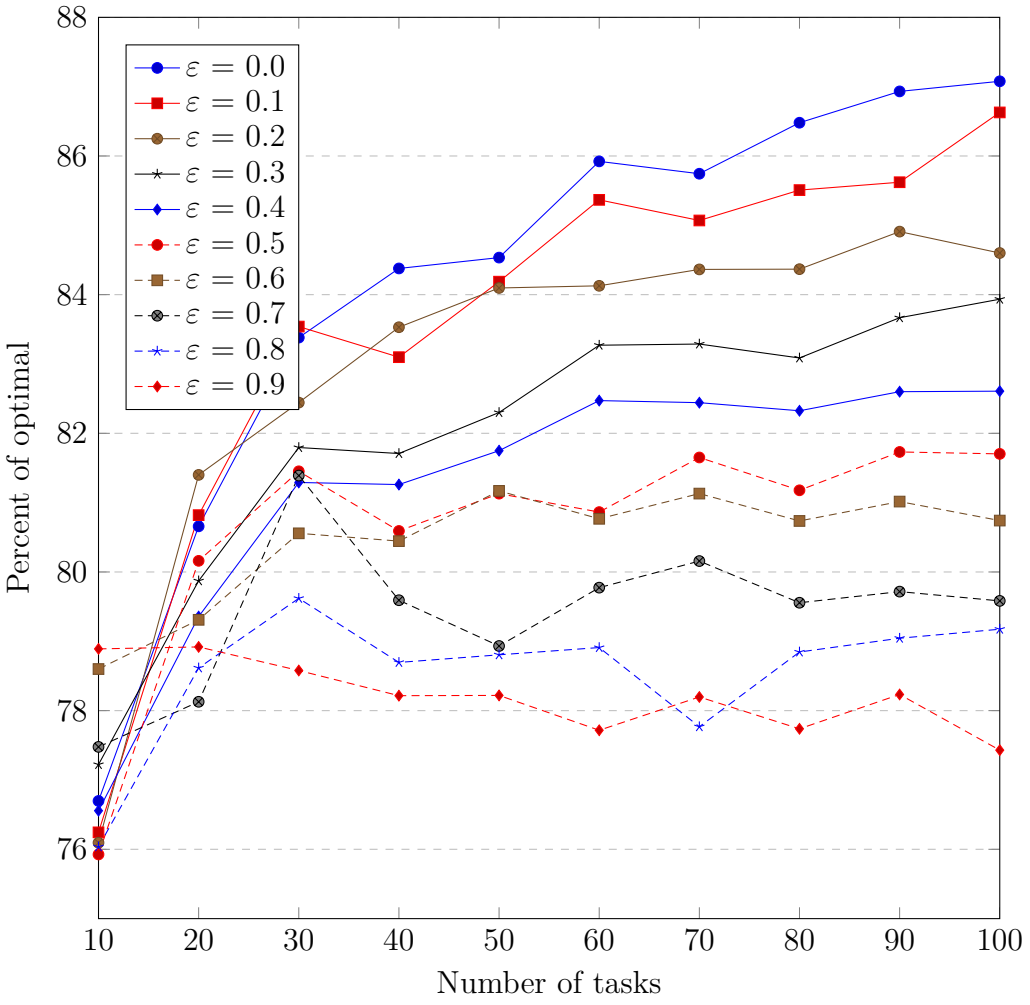
$$T_{explore} \subset T, \quad ||T_{explore}|| = \lfloor \varepsilon \cdot ||T|| \rfloor \tag{4.8}$$

Then, like when using the original BEF, the remaining tasks are used for the exploitation phase. Other than the selection of tasks, this implementation is identical to the one described in Algorithm 1.

4.7.2 Epsilon greedy

Epsilon greedy was implemented with regards to the new model, and the problem of finding the best ε still persists. Therefore several different values for ε were tested and compared, as seen in figure 4.2. Using an ε of 0.0 gives the best result when the number of tasks increases, even when decreasing ε over time. This means that the best results, for this problem, using Epsilon greedy were achieved when never choosing a worker at random.

Figure 4.2: Performance of Epsilon Greedy compared to the optimal solution, with regards to ε. Used to find the optimal ε.



Epsilon greedy tries to choose the best worker for a given task, without regards to any other tasks than might be assigned later. The pseudocode for the implementation can be seen below.

Algorithm 3: Epsilon greedy

input : T: a set containing tasks
W: a set containing workers
ε: probability to select random worker
Output: r: the average reward received by the workers

```
1  procedure EpsilonGreedy(T, W, ε):
2      for task t ∈ T do
3          let r be selected at random ∈ [0, 1]
4          if r < ε then
5              let w be selected at random from W
6          else
7              let w be best worker for task t
8          end
9          assign w to t
10         rate performance of w
11         estimate skills of w
12     end for
13     return average performance rating of all workers
14 end procedure
```

4.7.3 Upper confidence Bound

The only real modification made to this formula was to change what $\bar{x_j}$ represents, as it is supposed to represent the average reward. However, with multiple skills, only using the average reward of a worker gives no indication of the performance on any task. For example, a worker excelling in skill 1 and 2, while being bad at skill 3, might have a higher average rating than someone whose skill levels are exactly the opposite. But if the task in question requires excellent skill levels for skill 3, the second worker would be a better match. Instead of using the average reward, it uses the method described in 4.6. The method does not return only the optimal worker, but instead the score, i.e. the number of skill levels above requirement levels and how close their skill levels are to the requirement levels. This score is then used as $\bar{x_j}$ for UCB.

Algorithm 4: Upper confidence bound (UCB)

input : T: a set of size n, containing tasks
W: a set of size m, containing workers
Output: r: the average reward received by the workers

procedure *UCB(T, W)*:
  let $T_{testOnce}$ be the set where $||T_{testOnce}|| = ||W||$
  let T_{rest} be the rest of the tasks in T
  for $t \in T_{testOnce}$ **do**
    let w be the next worker assign w to t
    rate performance of w
    estimate skills of w
  end for
  for $t \in T_{rest}$ **do**
    let w be the worker that maximizes
    $$WorkerSelect(t, w) + \sqrt{\frac{2\log(\text{Total number of performed tasks})}{\text{Number of ratings for w}}}$$
    (WorkerSelect() is defined in Algorithm 2)
    assign w to t
    rate performance of w
    estimate skills of w
  end for
  return average performance rating of all workers
end procedure

4.7.4 Hungarian min-max estimation

While testing all above methods, it was discovered that all of them favoured very little distinct exploration or in some cases none at all. Modified BEF uses a very small ε for selecting tasks used only for exploration and Epsilon greedy works best with $\varepsilon = 0$. Since all modified methods in this thesis continue to estimate the workers skill levels, even when not using distinct exploration, a new experimental method was conceived. For all greedy methods, a significant drop in performance was observed when moving from single-skill matching to, the more complex, multi-skill matching. This performance drop was also observed when trying to apply the optimal method which used the knapsack algorithm with access to the workers true skills. The Hungarian algorithm is a methods that takes all current workers and tasks into consideration when making the assignments, and it showed much better performance than any greedy method.

Hungarian min-max estimation was created with the idea that always making the optimal assignments with regards to all current workers and tasks will produce the best results, as long as the workers skill levels are continuously estimated. Since the Hungarian algorithm gave good results when used with multiple skills and requirements, while still only needing $O(n^3)$ time to do so, it was used as a base for this

new method. Also, this method was the first one using the new min-max method of estimating skills, hence the name: Hungarian min-max estimation. This method uses the Hungarian algorithm, at each step, with an equal amount of workers and tasks, to make the optimal assignments. However, contrary to the optimal method this does not have access to the workers' true skill levels, but instead uses their estimations. After each assignment, they workers skill levels are estimated again which leads to better assignments during the next set of tasks.

4.7.5 Random

For each block of tasks, a random worker is selected for each task, with no regard to the workers estimated skill levels. The random method is, just as with single skill matching, used for measuring the performance of other algorithms.

4.7.6 Optimal

The optimal method is, like the random method, used primarily for comparison with the other implemented methods. It has access to all workers true skill levels, therefore no estimation is needed and an optimal matching can be done. When matching users with multiple skills and tasks with multiple requirements, a greedy method, like the knapsack algorithm, will not always produce the best result, which is described more in 5.4.1. Therefore an algorithm is needed that takes all current workers and tasks into account at the same time. The Hungarian algorithm is, as described in 4.7.4, a known algorithm for solving the assignment problem when all variables are known and is what will be used as the optimal solution.

5
Results

This chapter will present all results gained from using the different methods for solving the assignment problem with unknown variables.

5.1 Algorithm paramaters

When testing the different algorithms with the multi-skill matching model, a number of different parameters were used. Some of the parameters were the same for every test and algorithm but other were changed.

Table 5.1: Parameters used for testing all implemented methods.

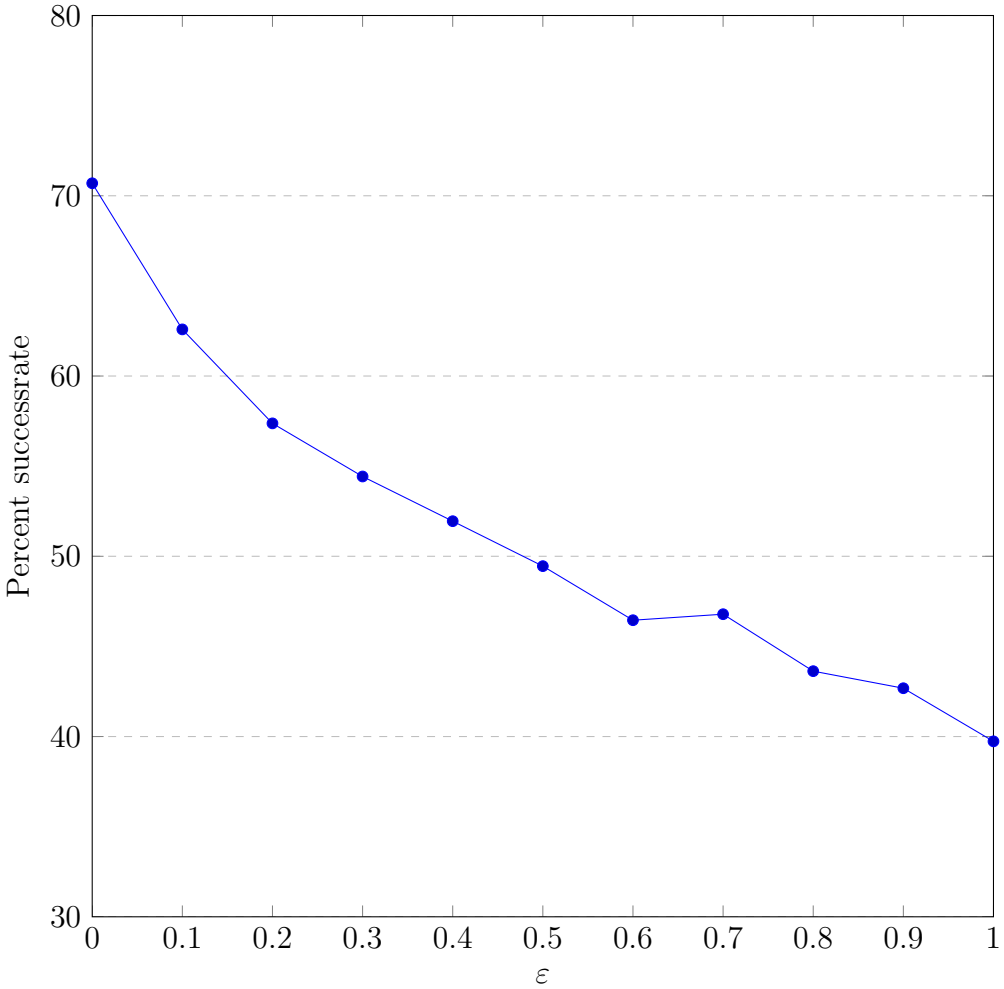
Parameter description	**Value of parameter**
Number of workers	10
Number of tasks	10 - 10000
Number of runs	25

The number of workers was constant, and was set to a relatively small number to speed up running time and to be able to analyze some of the results manually. Different number of tasks were used when testing different aspects. When trying to find the end result of assigning a large number of tasks, many tasks were generated as the running time was still withing acceptable limits. However, when trying to find how the performance changes with the number of tasks provided, many more tests had to be run and therefore fewer tasks were used. A run in this scenario is a single test with a set of tasks and a set of workers, and several runs were used because of two different elements of randomness with each run. First, when rating a worker's skill level, a Bernoulli distribution was used, and as described in section 4.4, it has a certain probability of giving an erroneous rating. Second, since all worker skill levels and task requirement levels are generated uniformly at random, there is always a probability of two sets being generated where very few matches are possible, or none at all. To account for this, every test was run 25 times and the result presented is the average value of all those runs. Also, for each run, the set of tasks is constant while new workers are generated every time.

5.2 Bernoulli rating

All methods described in this thesis uses the same model for rating workers and their skill levels. As described before, this rating model includes a level of randomness in the form of ε, which affects the outcome. This is used to model real life uncertainty, were it might be hard for any task provider to accurately rate a worker's performance in a particular skill. Using the method giving the best overall result, i.e. Hungarian min-max estimation, a number of different values for ε was tested to see how the result was affected, the result of which can be seen below.

Figure 5.1: Success rate of Hungarian min-max estimation with regard to ε, i.e. the probability that the correct skill level rating is given.

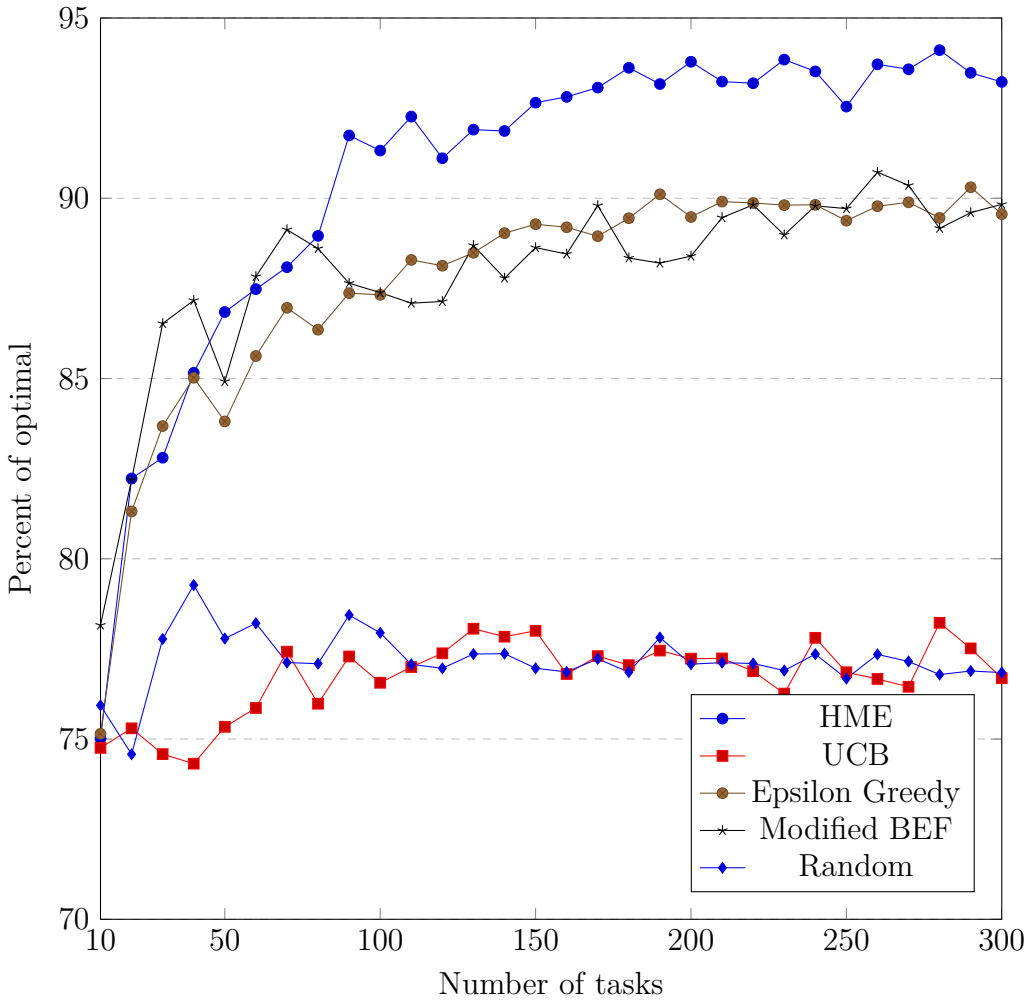


The value 0.015 was selected arbitrarily to be a small number. In this case it means that there is a 1.5% probability that the task provider gives the wrong rating for a particular worker skill.

5.3 Performance compared to optimal

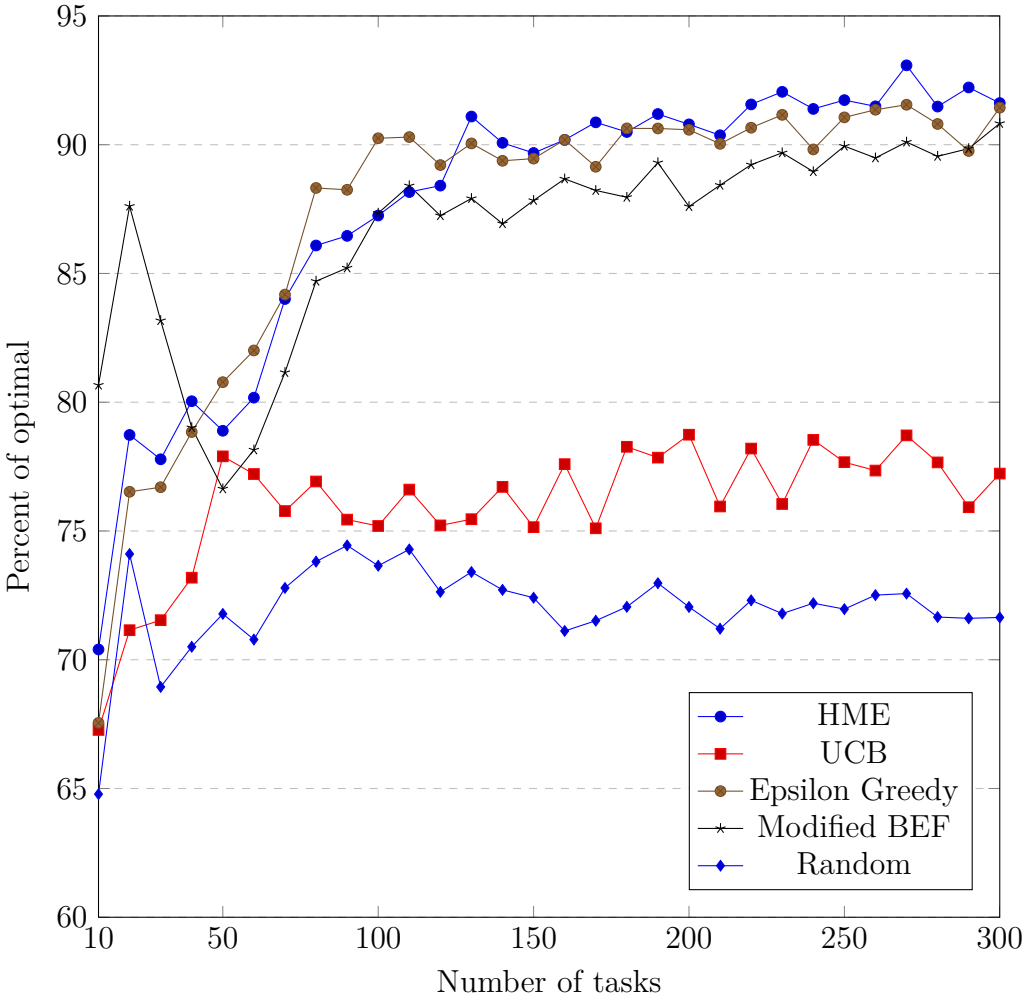
When all methods were implemented, using block matching, they were compared to the optimal solution over time, as an increasing number of tasks were assigned. The goal for any method is to make sure that the estimated skill levels of every worker converges towards the true skill levels, and each method use the same set of workers and tasks. During most of the experimentation, the number of available skills and requirements were limited to 3, to speed up computing time, while still using more than 1 skill. The results of all methods, while using 3 skills and requirements, can be seen in graph 5.2 below.

Figure 5.2: Performance of all algorithms compared to the optimal solution. Used to see if any of them converge towards the optimal solution, given enough tasks. Maximum number of skills per worker and requirements per task is 3.



However, the number of available skills and requirements could vary drastically in a real world scenario, so different number of skills and requirements were tested with the finished implementations. When using just one skill, the problem is basically the same as the Get Another Label problem, where BEF performed well. This is also evident in the plot below, where the three best methods, Hungarian min-max estimation, Modified BEF, and Epsilon greedy, produce very similar results.

Figure 5.3: Performance of all algorithms compared to the optimal solution. Used to see if any of them converge towards the optimal solution, given enough tasks. Maximum number of skills per worker and requirements per task is 1.



Tests made with more skills and requirements, from 3 up to 10, all give similar results as with 3 skills, meaning the big difference occurs when changing from 1 skill and requirement to multiple.

5.4 Method comparison

5.4.1 Hungarian min-max estimation

Regardless of the number of skills and requirements available, this method always produced the best results, as evident by graphs 5.2 and 5.3. The main difference between this method and the other ones is that it is not greedy, it instead takes all current tasks and workers into account when doing the assignment, resulting in a lower risk of wasting workers on too easy tasks. When only one skill is involved it is easier for an algorithm to make the optimal worker selection for a task. For example, consider the two scenarios with two workers and two tasks, table 5.2 using one skill and requirement, and table 5.3 using two skills and requirements.

Available data	
Requirements	
t_1	0.6
t_2	0.7
Skills	
w_1	0.6
w_2	0.7

Table 5.2: Worker and task example data, workers with 1 skill and tasks with 1 requirement.

Available data	
Requirements	
t_1	[0.5, 0.5]
t_2	[0.6, 0.4]
Skills	
w_1	[0.4, 0.6]
w_2	[0.6, 0.4]

Table 5.3: Worker and task example data, workers with 2 skills and tasks with 2 requirements.

When assigning the workers to the tasks, while following the rules, there is only one correct way to do it, resulting in the optimal match every time. Using the Hungarian algorithm would produce the same result in this case.

	w_1	w_2
t_1	X	
t_2		X

Table 5.4: Matching table for 2 workers and 2 tasks, workers with 1 skill and tasks with 1 requirement. The only greedy optimal assignment possible.

However, when using more than 1 skill and requirement, as in table 5.3, there can be two ways for a greedy algorithm to assign the workers, as seen below:

	w_1	w_2
t_1	X	
t_2		X

Table 5.5: Matching table for 2 workers and 2 tasks, workers with 2 skills and tasks with 2 requirements. First example of a greedy attempt at an optimal solution.

	w_1	w_2
t_1		X
t_2	X	

Table 5.6: Matching table for 2 workers and 2 tasks, workers with 2 skills and tasks with 2 requirements. Second example of a greedy attempt at an optimal solution.

Both assignments follow the rules, but they will give different results, were table 5.5 gives the optimal solution. In table 5.6, w_2 is wasted on t_1 and will be unavailable for t_2, which will result in an overall worse result.

5.4.2 Epsilon greedy and modified BEF

No matter the number of skills available, both these methods give similar results, since they use similar approaches, both methods use the concept of exploring and exploiting workers. This can be seen both in figure 5.3 and figure 5.2. When more than 1 skill and requirement is used, their results differ greatly from the one recieved from using Hungarian min-max estimation.

5.4.3 UCB

This solution gives results, not much better than random, no matter what number of skills and requirements are used. The biggest difference from Epsilon greedy and modified BEF is that it adds another layer of weight when choosing the best worker for a task, as described in 2.3.3, which might be unfavourable in this setting.

6
Discussion

All the results, for multi-skill matching, included in this thesis are based on experimentation with synthetic data, no real world data have been used. Therefore, results may vary greatly when any of the methods are applied to real world data. However, since all skill levels and requirement levels were generated uniformly at random, no conclusions could be drawn from the distributions. If a large amount of data from specific groups could be analyzed, patterns could be detected that might be able to add weights when estimating the workers skill levels. Because of this uncertainty we do not know if the solutions from this thesis would produce better or worse results in a real world setting. However, we think the results presented here are good and that there is something to be used. If the methods can not be used as is, they may be able to be used as a base to improve upon.

6.1 Limitations

All current solutions are implemented for the offline task assignment problem. This means that there is a static setup of workers and tasks. If more workers are added as the algorithm is running, they would be initiated with 0.5 in all estimated skill levels and are less likely to be selected for as many tasks as the already existing workers. None of the algorithms account for this at the moment. However, the idea that workers could be able to suggest their own skill levels might help. This can be both positive and negative depending on the honesty of the workers. With only honest workers, the true skill levels could probably be estimated faster, but if many of the workers are dishonest, the estimation might take much longer. One way to solve this might be to add a probability for each task to be assigned to a worker that have never been used before. Another solution might be to add more virtual tasks, described more in section 6.4, for estimating new workers before including them in the algorithm. That way the new workers are only compared to other new workers and none of the groups are given an advantage.

The option for workers to input their own perceived skill levels is actually something that was used from *Just Arrived*'s model, as new workers have the possibility to do so when signing up. This set of skill levels exist besides the set of estimated skill levels.

6.2 Estimation of skills

A lot of experimentation with multi-skill matching, during this thesis, was done with the system of estimating skill levels with rating averages. While results felt good enough, we also felt that there was definitely room for improvement. In a pretty late stage, a new estimating method was introduced, described in section 4.5, which when applied to all implementations improved the results considerably. While this approach works well in our simulated setting, we have no way of knowing how well it would work when the rating of skills is performed by a human being, i.e. a task provider rating a skill performance 1 or 0 depending on if the result is good enough or not. One big weakness with using the min-max approach is that the estimated values might get stuck, outside the real values, meaning the true skill level might never be found. Since it uses failure or success to modify the max and min skill levels respectively, giving an incorrect rating might give minimum value above the true level or a maximum value below it. Consider the following two examples, where the true skill level of the worker is 0.8:

Requirement level	**Min**	**Max**
initial	0	1
0.6	0.6	1

Table 6.1: Example of correct worker skill level estimation.

Requirement level	**Min**	**Max**
initial	0	1
0.6	0	0.6

Table 6.2: Example of incorrect worker skill level estimation.

In both cases, the initial estimation of the skill level is 0.5, but after just one mistake, as shown in table 6.2, the estimated skill level is down to 0.3, and the worker's probability to be selected for a task with a requirement level of 0.8 has decreased considerably. Also, a safeguard when updating the min and max is that the min value is set to the maximum of the current min value and the task requirement level, and the max value is set to the minimum of the current max value and the task requirement level. This decision was made to stop the estimation from fluctuating to much, but as evident by this small example, it also introduces a risk. With correct ratings however, the true skill level might be found quickly, as seen in table 6.1, where the current estimated level would already be 0.8.

One way to solve this problem might be to use the complete history of completed tasks to find where the incorrect rating was made. If something seems off, the problem might be found by running another algorithm besides the best one and comparing both results, one or more new tests could be made to try and find were things went wrong. The tests and results could be checked again, either by a person or automatically, and then update the estimated min and max values accordingly. Another way to solve it can be to always do a number of automated ratings, depending on some testing algorithm, although this might only be possible when it comes to software related tasks. For example, after a task provider has rated the skill performance of a programming test, a number of quality testing runs could be made to complement that rating. Using a human rating as well as an automated one would probably decrease the probability of an incorrect rating.

6.3 Rating of skills

As discussed above, the ε chosen for Bernoulli when rating skills was chosen arbitrarily. Some experimentation was done, but it was found that using a low value, meaning around $1 - 2\%$ probability of failure produced good results while still allowing for some uncertainty. Surely, a lot of research could be done into the area of how people make the wrong decisions, even when they should have the necessary competence to make the correct one. Using data from decisions made during different times of the day, or different days of the week for example, could give an average probability of how often people make those erroneous decisions. The ε could then be updated accordingly and the results would be one step further to representing the real world. Basically, the ε was added since we had no access to real data and wanted to add some level of uncertainty to try and mimic a real world scenario more closely.

6.4 Virtual tasks

The models for multi-skilled matching used in this thesis has some connection to real world data, but they require a larger supply of tasks than workers, which is not always the case. When looking at data like that used for the Get Another Label project, there were certainly many more tasks available than there were workers, about ten times more. But when it comes to the data from Just Arrived, the ratio is reversed, meaning there are about ten times as many workers available as tasks. In both scenarios however, there might be an incentive for task providers to use what we call virtual tasks. These tasks can be anything from written tests, trying to gauge workers abilities, to non important or at least less important tasks, that the task providers are willing to waste to find the workers true potential. A company working with development using different programming languages might need a new project manager. One way to find a good candidate could be to offer programming assignments and other written tests to try and estimate a potential worker's skill levels. All these tests could be made available online for anyone to try, meaning the company would have no need to spend any extra time until a good result appeared. Providing less important tasks could mean offering parts of larger tasks that have no big impact on the overall outcome. For example if a large programming project was underway and a certain smaller module was needed. A request to create that module could be put online and analyzing several candidates results could help with rating each of their skills. If the code-quality rating could be automated even more time could be saved, but that is probably a large project all in itself. Also, if the tasks made available contain no private information, the results from the candidates performances could be shared, meaning a lot more workers skill levels could be evaluated in a shorter amount of time.

7
Conclusion

7.1 Future work

As mentioned in section 6.1, an interesting addition to the methods described in this thesis would be to add a weight to workers depending on what distribution they come from. This could be used to simplify the matching process for particular groups of people in the future, hopefully decreasing the number of tasks needed to estimate the workers' true skill levels. This addition might also improve the result in the case were new workers are introduced in a set where the existing workers already have performed a number of tasks. If any conclusions at all can be drawn without the need to test the new workers, the number of tasks needed for skill level estimation might decrease considerably.

Another way to use this kind of model would be to analyze users of a job focused social network, for example, *Linkedin* [20]. Users can enter their own skill levels and other people can endorse them, this endorsement could work as a kind of starting point for evaluating their real skill levels. This, together with analyzing results from previous jobs via *Linkedin* and how pleased their employers are, might give a lot of interesting information. Somewhat related to this is that *Just Arrived* could try to apply this method as an indicator when the number of registered workers and tasks increases. Although it will need more work before it can be used without human interaction, it might give some hints for which recommendations to make. Testing this could certainly be done with data from many other sources than *Just Arrived*. Probably any source that contains sets of workers and tasks, especially if a history of completed tasks and ratings is accessible as well.

7.2 Problem description

As stated in section 1.4 and section 1.5, the first problem was to find way of matching a set of tasks and a set of workers in a way as close to the optimal solution as possible, while initially nothing is known about the workers' abilities. Also, as a sub-problem, find a quick and efficient way of estimating each workers skill levels.

Regarding the first part, to find a good way of matching workers and, we think that the results are good. If the process can be somewhat automated, anything better than completely random would be useful and the results presented in chapter 5 show that there is a least a possibility to come close to an optimal solution. Since all data

used is generated, there is no way to prove that the results would be similar in a real world setting, but since all data is created uniformly at random, adding more information about the workers might actually improve the results. Also, the results assume that someone is able to accurately rate the performance of each skill, or that a system is in place that can do that automatically.

When looking at the results, most solutions approach the optimal results, given enough time, with some doing it quicker than others. The best solution, Hungarian min-max estimation, reaches over 90% of the optimal results after each worker has been rated less than ten times. If virtual tasks are introduced, those ten performance ratings can be given in a early stage, which might also solve the problem where new workers are introduced dynamically. Overall, we think that results show that using Hungarian min-max estimation with the min-max skill level estimation is a good way to assign workers to tasks in a simulated scenario, and it is probably viable in a real world scenario as well.

Bibliography

[1] A. Begel, J. Herbsleb and M. Storey, "The future of collaborative software development", Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work Companion - CSCW '12, 2012.

[2] "Amazon Mechanical Turk - Welcome", Mturk.com, 2017. [Online]. Available: https://www.mturk.com/mturk/welcome. [Accessed: 25- May- 2017].

[3] A. Ivanovs, "Top 18 Most Popular Freelance Marketplaces 2017 - Colorlib", Colorlib, 2017. [Online]. Available: https://colorlib.com/wp/popular-freelance-marketplaces/. [Accessed: 25- May- 2017].

[4] "The three billion Enterprise crowdsourcing and the growing fragmentation of work", 2017. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/us/Documents/strategy/us-cons-enterprise-crowdsourcing-and-growing-fragmentation-of-work.pdf. [Accessed: 25- May- 2017].

[5] J. Munkres, "Algorithms for the Assignment and Transportation Problems", Journal of the Society for Industrial and Applied Mathematics, vol. 5, no. 1, pp. 32-38, 1957.

[6] "Waze – Crowdsourcing Maps and Traffic Information – Digital Innovation and Transformation", Digit.hbs.org, 2017. [Online]. Available: https://digit.hbs.org/submission/waze-crowdsourcing-maps-and-traffic-information/. [Accessed: 25- May- 2017].

[7] "McDonald's asks public to create products with 'burger builder' crowd-sourcing tool", Campaignlive.co.uk, 2017. [Online]. Available: http://www.campaignlive.co.uk/article/mcdonalds-asks-public-create-products-burger-builder-crowd-sourcing-tool/1294903. [Accessed: 25- May- 2017].

[8] "'Let's Go! Arctic' advertisements", Greenpeace East Asia, 2017. [Online]. Available: http://www.greenpeace.org/eastasia/multimedia/slideshows/climate-energy/lets-go-arctic-advertisements/. [Accessed: 25- May- 2017].

[9] C. Ho and J. Wortman Vaughan, "Online task assignment in crowdsourcing markets", in AAAI, Toronto, Ontario, Canada, 2012.

[10] L. Tran-Thanh, S. Stein, A. Rogers and N. Jennings, "Efficient crowdsourcing of unknown experts using bounded multi-armed bandits", Artificial Intelligence, vol. 214, no. 1, pp. 89-111, 2017.

[11] "Just Arrived - About Us", Just Arrived, 2017. [Online]. Available: https://justarrived.se/en/about-us/. [Accessed: 12- May- 2017].

[12] M. Katehakis and A. Veinott, "The Multi-Armed Bandit Problem: Decomposition and Computation", Mathematics of Operations Research, vol. 12, no. 2, pp. 262–268, 1987.
[13] S. Martello and P. Toth, KNAPSACK PROBLEMS ALGORIHNS AND COMPUTER IMPLEMENTATIONS, 1st ed. WILEY, 1990.
[14] M. Tokic, "Adaptive epsilon-greedy exploration in reinforcement learning based on value differences", in Advances in artificial intelligence, Karlsruhe, Germany, 2010, pp. 203-210.
[15] Peter Auer. "Using confidence bounds for exploitation- exploration trade-offs." Journal of Machine Learning Research, vol. 3, pp. 397–422, 2002.
[16] H. Kuhn, "The Hungarian method for the assignment problem", Naval Research Logistics Quarterly, vol. 2, no. 1-2, pp. 83-97, 1955.
[17] H. Kuhn, "The Hungarian Method for the Assignment Problem", 50 Years of Integer Programming 1958-2008, pp. 29-47, 2009.
[18] "Steps of the Hungarian Algorithm - HungarianAlgorithm.com", Hungarianalgorithm.com, 2013. [Online]. Available: http://www.hungarianalgorithm.com/hungarianalgorithm.php. [Accessed: 12- May- 2017].
[19] "ipeirotis/Get-Another-Label", GitHub, 2017. [Online]. Available: https://github.com/ipeirotis/Get-Another-Label. [Accessed: 25- May- 2017].
[20] "LinkedIN - About us", LinkedIN Corporation 2017. [Online]. Available: https://press.linkedin.com/about-linkedin. [Accessed: 16- May- 2017].