UNIVERSITY OF CALIFORNIA

SANTA CRUZ

EFFICIENT CROWDSOURCING ALGORITHMS FOR DISCOVERING THE TOP ITEMS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

TECHNOLOGY & INFORMATION MANAGEMENT

by

Shenshen Liang

June 2020

The Dissertation of Shenshen Liang
is approved:

Professor Luca de Alfaro, Chair

Professor John Musacchio

Professor Yi Zhang

Quentin Williams
Acting Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Efficient crowdsourcing algorithms for discovering the top items

by

Shenshen Liang

Crowdsourcing provides a convenient way to collect information from humans. It is proved to be an effective approach to solve large scale problems with relatively low cost. Discovering top items from large datasets is a fundamental problem, which finds its applications in various areas, such as players ranking, candidates recruiting, students admission, and so on. Crowdsourcing can provide cost-efficient solutions for these tasks, which usually require a large number of inputs.

We start the research by exploring crowdsourcing data for ranking. Two common data types, numerical grades and pairwise comparisons, are examined. We design grading and comparison tasks with the same budgets and run them in crowdsourcing platform. After collecting crowd workers' responses, we analyze the influence factors and compare the precisions of two types of data on ranking tasks. We choose to use pairwise comparisons in the rest of this dissertation based on the analysis results.

Generally, there are two scenarios in the problem of top item discovery. In one scenario, the number of target top items is unknown and unidentifiable, meanwhile the qualities of top items are more important than the bottom ones, i.e., the errors in top items incur larger losses. In this circumstance, the problem of identifying top items is essentially a top-heavy ranking problem. In another scenario, the number of target top items can be predefined or known. Such scenario occurs frequently when there is a constraint on the amount of resources available. The problem is equivalent to a top-K selection problem in this circumstance. In this dissertation, we propose novel algorithms to solve problems of both scenarios.

For top-heavy ranking problems, as there is extensive research on aggregating pairwise comparisons into rankings, we focus on studying strategies for selecting pairs of items to be sent to the crowd, in order to make rankings converge as quickly as possible to the correct results. We define a “top-heavy” version of ranking distance, and propose two algorithms to select the next pairs of items for the crowd to compare. In particular, we define the loss involved in each pair of items, and design strategies which select pairs that have maximum loss, or that lead to maximum ranking change. We demonstrate that the rankings converge significantly faster with our algorithms in the experiments. We further extend our algorithm to an efficient batched version, which delivers the comparable ranking results, while dramatically reduces the computation time and complexity.

For top-K selection problems, we propose an online top-K selection algorithm, as existing approaches are either inefficient as a result of having to learn a full ranking which incurs unnecessary crowdsourcing costs, or require a fixed amount of work for task completion and can not provide useful partial results if less work is performed. Our algorithm dynamically and repeatedly selects items into acceptance (in the top-K) and rejection (not in top-K) with an error tolerance $\epsilon > 0$. Partial results can be obtained at any point of time in the algorithm. We prove that the complexity of the algorithm is $O(N \log N)$, given an error bound and average probability of correctness of crowd. We further propose several optimization techniques to reduce the number of comparisons needed. Experimental results show that the algorithm can achieve comparable results with significantly fewer comparisons.

To my parents and husband.

Acknowledgments

Without the kind support of many people, I would not be where I am today. I am grateful to everyone who played a vital role in the completion of this dissertation.

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Luca de Alfaro. He provides insightful, inspiring, irreplaceable and invaluable guidance and support throughout my Ph.D life. I truly appreciate his patience and encouragement, his wisdom, enthusiasm and experience in research and life. His highly motivated and devoted attitude towards research sets an example of excellent researcher. He also sets a role model for me with his courtesy and patience with the students. The lessons I learned from him will have a lasting influence on my career and my life. It is my great honor and an unforgettable experience working with him.

I am grateful to my committee members, Professor John Musacchio and Professor Yi Zhang, for their insightful comments, constructive suggestions and valuable discussions on my research. Thanks to Professor Patrick Mantey for providing fruitful comments in my qualification exam. Especially, I would like to thank Professor Yi Zhang, who helped and encouraged me through my difficult times in pursuing my Ph.D. I am sincerely thankful to her.

I would like to thank my master advisor, Professor Ying Liu, for her guidance and encouragement. She introduced and led me into the world of science and research. Her positive, optimistic attitude in research and life has sustained me throughout the years.

My appreciation also goes to the collaborators and friends in my graduate career. Thanks to Michael Shavlovsky, Vassilis Polychronopoulos, Maria Daltayanni, Rakshit Agrawal, Chunye Wang, Zhongpeng Lin, Liang Dai, Jing Du, Joel Barajas, Karla Caballero, and other visitors and students that I have met. Their companionship and encouragement helped me through the many trials and tribulations of the doctoral journey.

Finally, I am much indebted to my parents, Zhongning and Zhuyuan, for loving, tolerating, supporting, comforting, and trusting me in my entire life. Their commitment and dedication to educating me has shaped my life and character. I also thank my husband, Liang, who has been through all the ups and downs with me. Without his support, patience and encouragement, I cannot make this journey. This dissertation is dedicated to them.

Chapter 1

Introduction

1.1 Crowdsourcing

With the development of advanced computation and artificial intelligence, computers can process large volume of data, perform complex calculations, and so on. However, there are many tasks that are difficult, or impossible, to solve by the most sophisticated computers and programs alone. For example, judging the appropriateness of an image, reviewing the quality of a product, or understanding the sentiment of a dialogue. In contrast, humans can provide answers to these problems with their unique capabilities. Humans are able to convert their perceptions and judgments into knowledge and concepts, and transfer them to different domains. As a result, some challenging tasks for computers can be easy, or even trivial for humans.

Crowdsourcing is one of the most common ways to obtain human-computational input. It decomposes large scale problems into pieces of micro-tasks, distributes them to a massive number of ordinary workers on the web, and obtains small pieces of information from the workers. It has been widely and extensively used in a variety of applications. In industry, companies have been using it to solve core business challenges. For example, Wikipedia, the largest online encyclopedia, utilizes crowdsourcing to gather information and data for a wide ranges of topics. Another example is Yelp, which collects and publishes crowdsourced reviews about businesses, and helps users to make informed decisions. In academia, universities and researchers use crowdsourcing to solve a variety of tasks: object annotation, image labeling, peer-graded homework, translation, proofreading, and so on. Compared to the traditional way of obtaining opinions from domain experts and professionals, crowdsourcing has been proven to be an effective way to retrieve human-powered information and solve large-scale problems, with relatively low cost and latency. In typical crowdsourcing systems, such as Amazon Mechanical Turk and Figure Eight, there are two types of roles: requester and worker. The requesters post a collection of tasks called Human Intelligence Tasks (HITs) to crowdsourcing system. The system broadcasts the tasks to the public. The workers can choose to accept any subset of tasks, and usually get paid by requesters upon completion of tasks. Figure 1.1 illustrates a crowdsourcing system with requester and worker.

1.2 Discover Top Items via Crowdsourcing

Discovering top items from large datasets is a fundamental problem, which finds its usage in various areas, such as student admission, selection of scholarship recipients, candidate recruiting, landing page design, and so on. These tasks usually require large number of inputs, and crowdsourcing provides a cost-efficient way to solve the problems.

Generally, there are two types of tasks in identifying top items. In one circumstance, the number of top items in demand is unidentifiable or unknown, while the top items are more important than the items below. For example, when a social media content

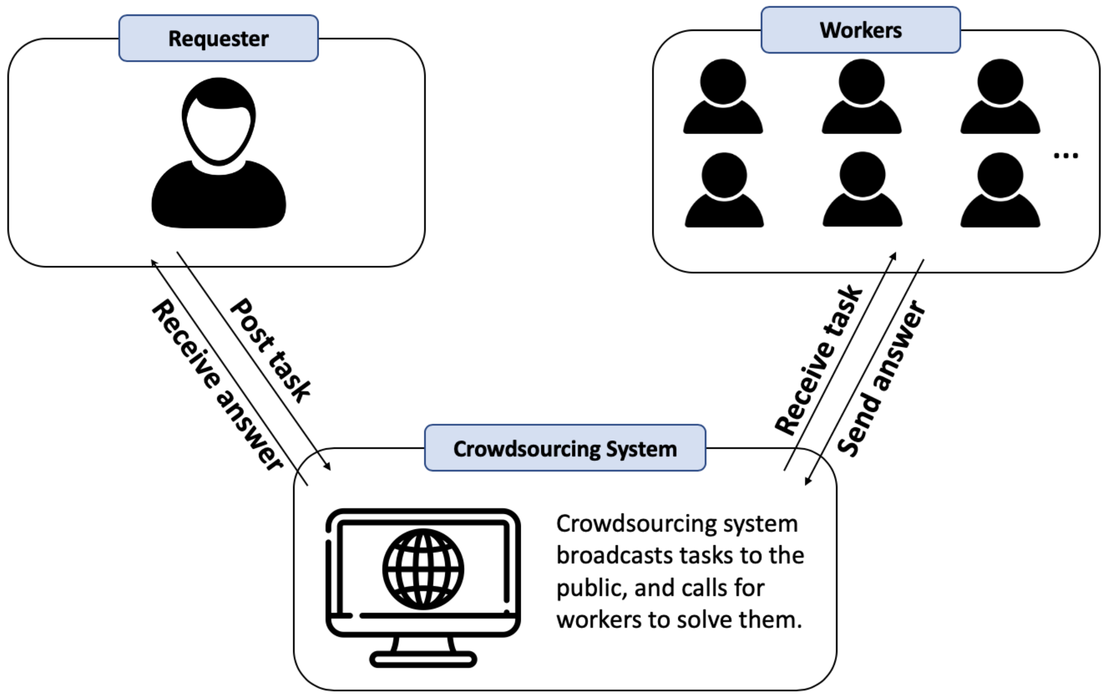
Requester
Workers
...
Receive answer
Post task
Receive task
Send answer
Crowdsourcing System
Crowdsourcing system broadcasts tasks to the public, and calls for workers to solve them.

Figure 1.1: Crowdsourcing system

creator designs their homepage, they usually places the more attractive contents at the top, so that they can receive more clicks on their contents. In other words, the number of clicks is related to the position where a content is placed in the item list. Failing to display the contents with proper order can result in reduced traffic and interests from online visitors. Apparently, the problem of identifying unknown number of top items is essentially a global, top-heavy ranking problem, where higher ranked items command geometrically higher visibility, revenue, or importance, and incur larger loss if they are mis-placed. In another circumstance, the number of target top items are predefined or known. Such scenario occurs frequently, when there is a resource restraint, where only a fix number of resources are available. For example, a department of university can afford to admit up to 30 prospective students in the next academic year. This type of problem can be formulated as an item selection problem, the goal of which is to find the top-K items from an itemset.

1.3 Dissertation Overview

Research of crowdsourcing has emerged as a new field in computer science, and the challenges require diverse efforts based on the specific problems and applications. In this dissertation, we focus on methods of finding top items, when there is a hidden ground truth in items' order. Specifically, we are interested in the problem where the hidden true order is based on the intrinsic 'quality' of items, as opposed to the orders that are based on personal preferences, item relevance, and so on. For example, in conference talks selection, the choice of talks depends on the quality ordering of talks. Moreover, we do not focus on the problems where there is historical information that can serve as objection function to learn a ranking. For instance, our methods do not aim at learning a ranking on products sales, given customers' purchase histories. We concentrate on the finding the top items from the itemset, where there is no prior information that can be used as objective function, and the order assessment is usually obtained by human inputs. For example, in student admission, while there are facts such as SAT scores, GPAs, and so on, none of them shall be used as objective functions to rank the students' qualification. Human judgments, such as committee members' assessments, are generally used to figure out the ordering.

In this dissertation, we analyze the crowdsourcing data for ranking first, and then propose crowdsourcing algorithms for top-heaving ranking and top-K selection problems.

In Chapter 2, we explore the two most common data types for crowdsourcing ranking problems. The two data types are ratings by numerical grades and pairwise comparisons. Each type has its advantages and disadvantages. Numerical grades provide fine grain information about the items, and they can represent humans' opinions precisely. It is also straight-forward to aggregate the numerical grades into a ranking. However, humans are found to be more prone to bias and errors when assigning grades. Pairwise comparison is a simple expression of preference between two items. It usually requires less cognitive workload, and is less prone to errors because of its simplicity. Nonetheless, the number of pairs is polynomial in the number of items in a dataset, and more comparisons are needed to finish a ranking task.

We investigate the human factors in the ranking process and compare the two data types, so that we can understand which data is better for learning the order of items, and can design the crowdsourcing data collection mechanism effectively. We propose different tasks and sent them to the crowd workers on Amazon Mechanical Turk. The costs of tasks are designed to the identical so that the results are comparable. We then estimate rankings for each task from crowd workers' responses, and compare the ranking precisions. The experimental results suggest that pairwise comparisons are more precise and efficient for crowdsourced ranking. Consequently, we use pairwise comparisons in the rest of this dissertation for top item discovery.

In Chapter 3, we study the crowdsourcing top-heaving ranking problem, where an ordinal rank k is associated with a benefit proportional to its position, and the errors in the head of ranking are weighted more heavily than the errors in the tail. Though extensive research has been done on how to aggregate pairwise comparisons into accurate rankings [41, 53, 77, 83, 91, 101], it is still very challenging to efficiently obtain data with minimal cost. Therefore, we concentrate on studying strategies for selecting pairs of items to be sent to the crowd for comparison, so that rankings will converge as quickly as possible to the correct result.

In setting the problem, we assume that each item in a ranking has an intrinsic "quality", and we define a "top-heavy" version of ranking distance where the mis-placement of items is weighed according to $1/k^{\lambda}$ for rank k with $\lambda > 0$; this will be used to judge the speed of convergence of the proposed methods. As ranking aggregation is not our primary focus, we perform ranking aggregation by applying the Glicko [44] and TrueSkill [51], which are well-established on-line aggregation algorithms. We then propose two algorithms to select the next pairs of items for the crowd to compare. In particular, we define the loss involved in each pair of items, and design strategies that select pairs for comparison that have maximum loss, or that lead to maximum ranking change. We demonstrated the ranking converge significantly faster with our algorithms in the experiments. Furthermore, the pair selection strategies are computationally expensive, as they require calculation of many alternative pairs to select the best next one. To address this problem, we further propose efficient bathed version of the strategies, which deliver the the same ranking precision essentially, while dramatically reduce the computation time and complexity.

In Chapter 4, we explore the crowdsourcing top-K selection problem. Generally, there are two types of approaches for this problem. One type consists in determining a global ranking, and then returning the top-K elements of such a ranking. This type of method has been extensively studied in crowdsourcing communities, and usually has the advantage of being able to offer partial results before the completion of ranking. However, it can be inefficient as it learns a full ranking, even among the items that are all selected, and all rejected. It costs unnecessary crowdsourcing budgets to determine and refine the global ranking. Another type of approach focuses on selecting the top-K items rather than building a global ranking. It can be far more efficient than global-ranking algorithms, as the crowdsourcing work is aimed at separating the top-K elements from the rest, rather than obtaining a full ranking. However, these specialized top-K algorithms require the completion of a fixed amount of work, and do not provide partial results if less work is performed. Yet, partial results are usually useful. For example, in candidate selection, one may wish to extend offers to the candidates as soon as they are determined to be in the top-K set. If the crowdsourcing work takes longer than expected, if a deadline makes it impossible to carry it to the end, or if the funds for it are exhausted, having a usable partial result is the difference between failure and (partial) success.

we propose a top-K selection algorithm that dynamically refines a classification of the items into accepted (in the top-K) and rejected (not in top-K). Given an error tolerance $\epsilon > 0$, the algorithm creates a batch of comparisons, and as the responses from the crowd come in, the algorithm uses them to refine the classification. The process is repeated until the top-K elements are identified. Partial results can be obtained at any time by returning the accepted elements, along with the unclassified elements that have the largest estimated quality. We show that given the average probability of crowd correctness q, for a fixed error tolerance $\epsilon > 0$, and for any top K/N percent, the expected number of required comparisons is $O(N \log N)$, where N is the total number of elements. The worst-case occurs when $K = N/2$. We further propose several optimization techniques to improve our algorithm. The algorithms are evaluated analytically and experimentally, and the results demonstrate that our algorithm can achieve more accurate results with same amount of data.

Last but not least, we summarize our contributions and conclude our research in Chapter 5.

Chapter 2

Crowdsourcing Rankings via Ratings and Direct Comparisons

2.1 Introduction

In top item discovery problem, one of the essential tasks is to learn the order among items, i.e., ranking items. In crowdsourcing systems, there are two common data collection schemes for ranking problems. Requesters can either ask workers to provide ratings to individual items with numerical grades, or to perform direct comparisons between pairs of items. Each type has its advantages and limitations.

Numerical grade is commonly used in ranking problems. For example, Yelp asks their users to rate businesses with grades between 1 to 5, and uses the aggregated grades to rank the merchants. Numerical grades provide fine grain information about the target objects, and are capable of representing worker's opinion precisely. Meanwhile, it is relatively simple and straight-forward to aggregate grades to rankings, for example, by averaging and sorting the grades. However, the process of grading can be inconsistent and prone to errors. For example, after interviewing overqualified candidates frequently, the interviewers may raise their expectations unconsciously, and rate subsequent candidates lower than normal.

Pairwise comparisons, on the other hand, have become a popular data source for ranking problems. In a single pairwise comparison, two items are presented to a user together. The worker is asked to compare the pair and make a binary decision on one over another. Pairwise comparisons are relatively easy to obtain from crowds, due to the low cognitive workloads required in comparison. However, it requires more data to learn a ranking, since the number of pairs is polynomial in the number of total items. Also, it is less straight-forward to aggregate the comparisons to rankings.

The goal of this chapter is to understand the human factors that influence the accuracy of crowdsourcing data for ranking, and determine which data type shall be used to obtain more precise information. We propose three research questions, design experiments and compare the accuracy of rankings that are achieved with various crowdsourcing settings. In particular, we keep the budgets of all tasks identical, so that the results are valid for comparison.

The rest of the chapter is organized as follows: after a review of background in Section 2.2, we describe our methodology in Section 2.3, where we define the research questions and design experiments for the answers. In Section 2.4, we present our experimental outcomes and discuss the results. Finally, we conclude this chapter in Section 2.5.

2.2 Related Work

Crowdsourcing obtains small pieces of information from ordinary crowds. It has been proven effective to get human-powered information with relatively low cost [100]. Crowdsourcing has been used extensively in many areas, such as opinion collection, ranking, knowledge sharing, entity recognition, object annotation, and so on.

Crowdsourcing platforms, such as Amazon Mechanical Turk [97], Figure Eight (formerly known as CrowFlower) [36] and ChinaCrowd [21], provide matketplaces where users can request mass number of workers to perform human intelligence tasks (HITs). Generally, there are two types of users in these platforms, namely, requester and worker. The requesters design and post HITs to the platform to collect inputs from human workers, and usually pay a small amount of money for each HIT. The workers browse and accept the HITs that they would like to work on, and receive the compensation upon completing and submitting their answers.

Ranking is a popular problem in crowdsourcing platforms. Usually, there are two major schemes to collect inputs for rankings: ratings via grades, or pairwise comparisons. In grading tasks, requesters define numerical ranges, which are usually non-negative, and ask workers to choose a value within the range to evaluate the target object. Grades allow workers to express their judgments with finer grained information, and thus can more precisely describe worker's opinion over the items. Moreover, the methods of aggregating grades to rankings are straight-forward, including calculation of average, median, and so on. However, the grades can be inconsistent between workers, or even inconsistent between the same worker with different contexts. As a result, grades can be prone to errors.

In pairwise comparison tasks, workers express their relative preferences between two objects, without mapping their judgments to grades and assigning the grades explicitly. Such judgments are relatively easier to make, since they require less cognitive workloads than mapping judgments to grades. However, it usually requires a high number of comparisons to achieve good ranking results, and the ranking aggregation methods are more complicated.

In [17], Checco and Demartini analyzed and compared three types of data, i.e., grades, stars and pairwise comparisons, for ranking in crowdsourcing environments. However, their experiments were conducted by collecting workers' preferences to the estimated rankings. As a result, the evaluation of rankings can be subjective and biased. Yang et al. explored three approaches of pairwise comparison, absolute relevance and relevance ratio for relevance assessment, but their research focuses on understanding and comparing the performance and cost differences between the inputs from crowdsourcing and conventional experts [104]. Kuhberger and Gradl evaluated three response modes of choice, rating and ranking when the crowd is presented with risky options, but focused on different mode's impact on risky option selection rather than ranking [64].

A classical, tournament-based algorithm, Glicko, is used to aggregate pairwise comparisons to rankings. The algorithm takes the outcomes of head-to-head games as comparisons, and assumes that a player's skill follows a Gaussian distribution, with two parameters as player's average skill and uncertainty. Glicko extends Bradley-Terry model and updates players' skills with comparisons [44].

2.3 Experimental Methodology

In this section, we propose research questions to understand the factors that influence human's precision on grading, and to compare the ranking outcomes from grading and pairwise comparisons. We then describe our methodology to investigate the problems, and present four metrics for evaluating ranking precision.

We examine and compare two common data schemes that ranking tasks adopt in crowdsourcing platforms:

- Numerical grade. Workers evaluate a single item with one grade in given range.

- Pairwise comparison. Workers select one item from an item pair. In particular, we do not allow tie in comparisons to simplify decision-making process.

The experiments are conducted by posting human intelligence tasks (HITs) to Amazon Mechanical Turk. We collect crowd's answers and aggregate them to rankings. The estimated rankings are assessed with four metrics described in Section 2.4.

We believe that when human brains process grades for object assessment, an intrinsic step is to compare and make references and utilize the existing knowledge of the similar objects. This is a critical step in human's evaluation process, which occurs frequently, sometimes even unconsciously, regardless of whether the evaluation task asks for an explicit comparison. For example, when a Yelp user rates a restaurant, they may compare the taste of food, service, environment, etc., to similar restaurants they has visited, and assign a grade that is in consistent with their prior experience, or the expectation that is developed through the prior knowledge. Another example exists in real estate property valuation, where appraisers refer to properties with comparable size, years of construction, location, and so on, and leverage the values of those properties to assess the target property. Apparently, the information that human use as reference in assessment plays a significant role in evaluation process. As a result, we focus on exploring the impacts of reference information to ranking. In the rest of this chapter, we refer such information as 'context'. Apparently, a context is self-contained in pairwise comparisons: a worker only needs to compare the two items in the pair, and does not require extra reference to make a decision.

Our research investigate the following research questions (RQ):

RQ1: Does context of grades impact ranking precision?

RQ2: How does biased order in context influence ranking precision?

RQ3: How do grades versus pairwise comparisons perform in crowdsourcing ranking problems?

In the rest of this section, we will first introduce our designed datasets and experimental setups, then describe the methods we design to investigate the research questions.

2.3.1 Datasets and Experiment Settings

There are various types of data that can be used for crowdsourcing analysis, such as area of polygons, facial expressions, fuzziness of images, and so on [5,31,70,102,106]. In this investigation, we choose to use polygons, because while there is a clear ground truth in the order of polygons areas, evaluating them requires low cognitive cost, and we can easily generate the desired statistical properties and adjust the difficulty level of tasks.

We generate polygons of various shapes and areas as follows: For any polygon, the number of angles is determined by uniformly sampling an integer from the range of $[3, 20]$. The area of a polygon is a float value, which is uniformly sampled from the range of $(0, 100]$. We then generate a convex polygon that has the corresponding number of angles and area. After the polygon is generated, each polygon is plotted inside a fix-size rectangular. An example of a polygon is provided in Figure 2.1.

We generate 20 datasets for the research, with 40 polygons in each dataset. Each dataset is sent to workers on Amazon Mechanical Turk via Human Intelligence Tasks (HITs). In the analysis, we design two types of HITs: grading HITs and comparison HITs. For grading HITs, the workers are asked to evaluate the area of a polygon with a number within $(0, 100]$. For comparison HITs, worker are required to compare the area of two polygons and pick the larger one. In particular, we purposely do not provide

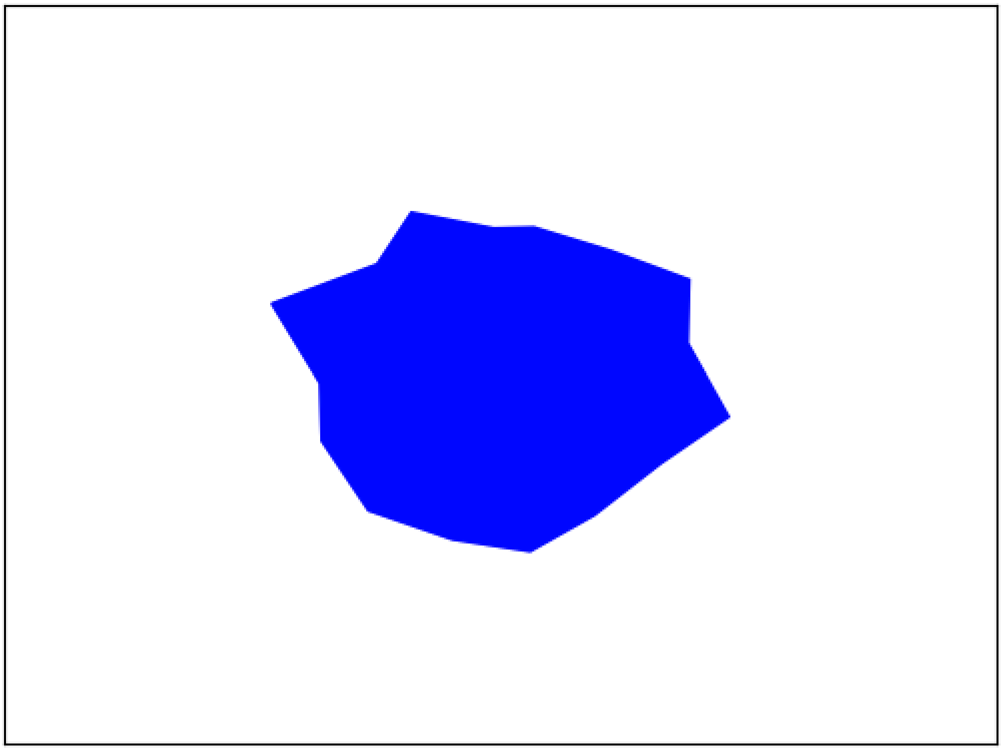

Figure 2.1: Example of a polygon

any guidance or baseline for grading tasks, because we want to simulate the real-world situations where there is no guideline for making an assessment, for example, evaluating students for admission, grading student essays, and so on. In order to make the outcomes of tasks comparable, we keep the budgets identical for any experiment on each dataset.

2.3.2 Experiment Design

In this section, we describe the research questions in detail, and illustrate our methodology and experiments for the investigation.

Impact of Grading Context (RQ1)

In real world scenarios, people usually get multiple elements in grading tasks. For example, an professor needs to grade the homework of multiple students; a company assesses multiple candidates; and so on. In RQ1, we want to understand a fundamental question from psychological perspective: do humans grade each item in isolation, or do they first grade a few items without prior information, then use the previous items and grades as reference points to assign future grades? In other words, does a context impact human's precision in grading?

We design grading experiments to answer this question. For each dataset that we generate (which is described in Section 2.3.1), we send grading tasks to Amazon Mechanical Turk and ask the workers to grade the area of each polygon. The tasks are designed as follows:

Task 1: Present all 40 items in sequence in one HIT, where items are ordered randomly.

Task 2: Present every item in a single HIT, where HITs are ordered randomly.

In task 1, workers receive one item at a time, and the next item is only shown after the previous item is graded. Workers are not allowed to go back and revise the grades they have assigned. In this way, workers can make references from previous items and calibrate the future grades, i.e., a context is provided to workers. In task 2, the items are presented in an isolated fashion, and there is not an available context.

Each task has an identical cost of $\$0.8$. We send out each task 5 times for every dataset, so the cost of a task is $\$4$ for each dataset.

How we process the HIT responses

After collecting the responses of all tasks, we calculate the averages grades of each item for the two tasks. Then, we estimate two rankings for each dataset, by sorting the average grades of each task. We will demonstrate the ranking results in Section 2.4.1 of this chapter.

Impact of Biased Order in Context (RQ2)

Biased order in context, where top items appear early in sequence, is a phenomenon that happens frequently in real world. For example, when companies and universities hire employees, they usually incline to evaluate the candidates that appear to be more qualified and have stronger background first. Apparently, the bias in order can impact the first impressions of humans. Psychological research has suggested that primacy effect, also known as first-impression bias, can influence human judgments significantly, even in the presence of contradictory evidences afterwards [3, 45, 89]. Such bias can last long [47] and is consistently found to be resistant to change [46, 73].

We believe that when people make references in grading, essentially, they memorize the characteristics of some prior items, and use them as 'signposts' for future grades. Due to the first impression bias, the initial items may become the major referencing points. Workers may develop a profound memory in those items, and leverage them heavily for grading.

In this research question, we are interested in the impacts of biased order in context, and how people are influenced by different levels of bias in the order. We design six tasks to investigate this problem. For every dataset described in 2.3.1, we send six grading tasks to the crowd respectively. Each task organize all 40 items in a particular order in a single HIT. Similar to task 1 of RQ1, the items are displayed to workers one by one in sequence, and the grades can not be modified once assigned. We create various levels of bias in context in the tasks, by placing different number of top items at the beginning of sequence. The task are designed as follows:

Task 1: All items are ordered randomly.

Task 2: Uniformly sample two top items and place them at the beginning of sequence, then randomly order the other items.

Task 3: Uniformly sample five top items and place them at the beginning of sequence, then randomly order the other items.

Task 4: Uniformly sample ten top items and place them at the beginning of sequence, then randomly order the other items.

Task 5: Uniformly sample fifteen top items and place them at the beginning of sequence, then randomly order the other items.

Task 6: Uniformly sample twenty top items and place them at the beginning of sequence, then randomly order the other items.

We define the 'top items' as the half of items with larger areas in the dataset. Note that the task 1 in this research question is the same as task 1 in RQ1, so we can re-use part of the results from RQ1. The cost of each task is again $\$0.8$, and each task is sent out 5 times for each dataset. Consequently, the cost of a task is still 4 for each dataset.

How we process the HIT responses

Similar to RQ1, we calculate the average grades and estimate six rankings for each dataset. The results will be discussed in Section 2.4.2.

Comparisons vs. Grades (RQ3)

We have explored the characteristics of grading tasks. To this point, we want to compare the performance of grades and pairwise comparisons in ranking tasks, and understand which type can yield more precise results.

To answer this question, we design pairwise comparison tasks, and compare them with the best-performed grading task. In the comparison tasks, we generate a pair by randomly matching two items from a dataset. In this way, a total of 20 random pairs are generated for each dataset. Each pair is sent to the crowd in a single HIT, and tie is not allowed. In summary, for each dataset, we perform and compare the following three tasks:

Task 1: The grading task with best performance from RQ1 and RQ2.

Task 2: Send 20 random pairs for comparison, with each pair in a single HIT. The cost of one pairwise comparison is identical to the cost of one grade.

Task 3: Send 20 random pairs for comparison, with each pair in a single HIT. The cost of one pairwise comparison is two times as the cost of one grade.

We design the different costs of two comparison tasks to capture the different efforts that a pairwise comparison task requires. If the item for evaluation is relatively simple and straight-forward, comparing two items can be considered as an atomic, unit task which requires the similar effort as grading an item. In another case, if the item is relatively complex to evaluate, comparing two items requires two units of efforts of evaluating the items, while grading an item only requires one. As a result, the cost for comparison doubles that of grading.

In task 1, each dataset costs 0.8 and is sent 5 times for grading. In order to make the results of task 2 and 3 comparable, we need to keep their costs the same as task 1. Consequently, each item in a dataset receives 10 comparisons in task 2, and 5 comparisons in task 3. The costs of task 2 and task 3 are both $\$4$ for each dataset.

set receives 10 comparisons from crowd workers, and we pay $\$0.02$ for each comparison. As a result, a dataset costs $\$4$. In task 3, each item in a dataset receives 5 comparisons and we pay $\$0.04$ for each. Consequently, a dataset still costs $\$4$.

How we process the HIT responses

For grading tasks, we estimate the rankings in the same way of RQ1 and RQ2. For comparison tasks, we use Glicko [44] to aggregate the comparisons into a ranking for each dataset. The estimated rankings from grades and pairwise comparisons are illustrated and compared in Section 2.4.3.

2.3.3 Ranking Quality Evaluation

We use four metrics to evaluate the aggregated rankings, namely, bubble count distance, bubble quality distance, absolute ranking difference, and absolute quality difference. In this section, we introduce each metric respectively.

We consider an itemset $S = \{I_1, I_2, \ldots, I_n\}$ with n items, where each item $I_i \in S$ has an intrinsic quality q_i. In the metric definition below, we denote the true ranking of items as r, and the true ranking of item I_i as $r(i)$. Similarly, we denote the estimated ranking as r' and estimated ranking of item I_i as $r'(i)$. The four metrics are defined as follows.

Bubble Count Distance

When r' is different from r, we can correct it by performing the bubble sort algorithm and swapping the orders of items. The number of swaps that takes to transform r' to r represents the dissimilarity of two rankings. It is also referred as Kendall tau distance [63]. We normalize the count of swaps with n to make the distances of different sized datasets comparable. So the bubble count distance can be expressed as:

$$\mathcal{L}(r, r') = \frac{\sum_{(i,j)} c(i, j)}{n} \tag{2.1}$$

where

$$c(i,j) = \begin{cases} 1 & \text{if } \Big(r(i) > r(j) \text{ and } r'(i) < r'(j)\Big) \text{ or } \Big(r(i) < r(j) \text{ and } r'(i) > r'(j)\Big) \\ 0 & \text{otherwise} \end{cases}$$

Bubble Quality Distance

When we interpret the swap in bubble sort algorithm from cost perspective, we consider each step to correct r', i.e., each swap, is associated with a cost. To account for the cost of correcting r', we use the absolute quality difference between two items I_i and I_j, i.e., $|q_i - q_j|$, to represent the cost of swapping a mis-ordered item pair. We denote the normalized cumulative sum of the quality differences between all swapped pairs as bubble quality distance:

$$\mathcal{L}(r, r') = \frac{\sum_{(i,j)} v(i,j)}{n} \tag{2.2}$$

where

$$v(i,j) = \begin{cases} |q(i) - q(j)| & \text{if } \Big(r(i) > r(j) \text{ and } r'(i) < r'(j)\Big) \text{ or } \Big(r(i) < r(j) \text{ and } r'(i) > r'(j)\Big) \\ 0 & \text{otherwise} \end{cases}$$

Absolute Ranking Difference

In this metric, we compare the pairwise difference between true ranking $r(i)$ and estimated ranking $r'(i)$ of each item I_i. The absolute ranking difference is defined as the normalized difference between the two ranking, $r(i)$ and $r'(i)$, for all items in itemset, i.e.,

$$\mathcal{L}(r, r') = \frac{\sum_{i=1}^{n} |r(i) - r'(i)|}{n} \tag{2.3}$$

Absolute Quality Difference

We also compare the quality difference between two items with the same ranking in r and r'. Such difference depicts the distance between the quality distributions of r' and r. Denote the item that ranks at p position of true ranking as I_i, and its quality as $q_{r(i)=p}$. Similarly, denote the item that ranks at p position of estimated ranking as I_j, and its quality as $q_{r'(i)=p}$. The absolute quality difference is calculated the normalized sum of quality differences, i.e.,

$$\mathcal{L}(r, r') = \frac{\sum_{p=1}^{n} |q_{r(i)=p} - q_{r'(j)=p}|}{n} \tag{2.4}$$

2.4 Experiment Results and Discussions

In this section, we evaluate the ranking results with the four metrics, compare the evaluation outcomes, and discuss our findings.

As described in section 2.3.2, every task has a cost of $\$4$ on each dataset, and we estimate a ranking for each task on each dataset. For each estimated ranking, we evaluate it with the four metrics as described in 2.3.3. Subsequently, for each task, we calculate the average and the 95% confidence interval of each metric among all the 20 datasets. In summary, we calculate an average ranking loss, and a 95% confidence interval of loss, on every metric for each task. The results are demonstrated in the following sections.

2.4.1 RQ1: Grading in Isolation versus in Context

We obtain two rankings results from HITs in isolation and in context respectively. The comparison of the ranking precision is summarized in Table 2.1. In each cell of the table, the first number is the average loss of 20 datasets, and the second number is the range of 95% confidence interval. For example, 10 ± 2 refers to an average loss of 10, with the 95% confidence interval of $[8, 12]$.

Table 2.1: Ranking losses from grading tasks, with and without context

Metric	**Bubble Count Distance**	**Bubble Quality Distance**	**Absolute Ranking Difference**	**Absolute Quality Difference**
Task 1: with context	2.60 ± 0.39	12.44 ± 3.6	4.02 ± 0.54	3.58 ± 0.48
Task 2: without context	4.30 ± 0.18	31.66 ± 3.1	6.27 ± 0.3	5.76 ± 0.46

We also plot the data in Figure 2.2 for each metric respectively.

From Figure 2.2, the rankings from the grades that are obtained with context constantly outperform the ones from the grades that are obtained in isolation. The average loss from contextual grades is significantly lower than those from isolated grades, and the 95% confidence intervals does not overlap, on every metric that we use.

The results of experiments demonstrate that when grading tasks are performed with a context, they can produce more precise rankings than the ones that are executed without a context. Actually, such experimental results align with our expectations. We believe that when humans are assigned with a set of items for grading, they assign a few grades without reference, memorize them, and form a statistical opinion of the items' distribution with the memory, consciously or unconsciously. Such opinion on distribution makes people more consistent in assigning grades afterwards. In contrast, when items are presented in an isolated fashion, it is more challenging for humans to form such statistical ideas, thus the grades assigned by workers are more subject to various noise. As a result, the grades from isolated tasks are more inconsistent, and the rankings are less precise.

From the results of RQ1, we believe that in crowdsourcing environments, workers are more precise when they grade items with context. Grading tasks are preferably assigned with context.

2.4.2 RQ2: Biased Order in Context

We propose 6 tasks in this research question. All grading tasks have their own contexts available, while the levels of bias in order are different from each other. The average loss and the 95% confidence interval of each task is illustrated in Table 2.2. Additionally, we visualize the results in Figure 2.3.

Table 2.2: Ranking losses with different biases in contexts

Metric	**Bubble Count Distance**	**Bubble Quality Distance**	**Absolute Ranking Difference**	**Absolute Quality Difference**
Task 1: no bias	2.6 ± 0.39	12.44 ± 3.6	4.02 ± 0.54	3.58 ± 0.48
Task 2: 2-item bias	2.71 ± 0.51	14.54 ± 4.9	4.12 ± 0.73	3.6 ± 0.7
Task 3: 5-item bias	2.8 ± 0.50	15.47 ± 5.46	4.18 ± 0.69	3.77 ± 0.72
Task 4: 10-item bias	3.33 ± 0.72	22.37 ± 9.03	4.94 ± 1.01	4.36 ± 0.99
Task 5: 15-item bias	3.91 ± 0.68	27.43 ± 7.85	5.72 ± 0.94	5.1 ± 0.85
Task 6: 20-item bias	4.13 ± 1.01	29.36 ± 10.30	5.99 ± 1.2	5.31 ± 1.34

From Figure 2.3 it is clear that the average ranking losses increase monotonically, when there is more bias existing in context. The differences become more evident when more top items are placed at the beginning of sequence. Meanwhile, the variances generally increase with more bias in the context.

We believe these results reflect how human brains process information, and how it can be influenced. As discussed in RQ1, when humans perform grading tasks with context, they memorize certain characteristics of a subset of items, and form statistical ideas of item distributions. The information of those items are used as 'reference' to compare with other items. From the psychological research in [47, 73, 89], it is apparent that due to the 'first impression effect' and its nature of long-lasting and change-resistance, the more bias exists at the beginning of item sequence, the stronger first impression is, so that the statistical idea of distribution is influenced more heavily, and more uncertainties are introduced to the grading process. As a result, with more bias at the beginning of item sequence, we observe larger average losses and variances in ranking results.

From the results of RQ2, it turns out that an unbiased context is preferred when assigning crowdsourcing grading tasks.

2.4.3 RQ3: Grades versus Pairwise Comparisons

We compare the two pairwise comparison tasks with the grading task with unbiased context, which produce the best ranking precision among all grading tasks.

Table 2.3 shows the average ranking losses and 95% confidence interval of the rankings from pairwise comparison tasks and the best-performed grading task. The corresponding graph is provided in Figure 2.4

Table 2.3: Ranking losses of pairwise comparison tasks and grading task without biased context

Metric	**Bubble Count Distance**	**Bubble Quality Distance**	**Absolute Ranking Difference**	**Absolute Quality Difference**
Task 1: grading with unbiased context	2.60 ± 0.39	12.44 ± 3.6	4.02 ± 0.54	3.58 ± 0.48
Task 2: pairwise comparison with unit cost	2.49 ± 0.16	10.91 ± 1.34	3.83 ± 0.22	3.53 ± 0.32
Task 3: pairwise comparison with 2X cost	2.56 ± 0.24	11.9 ± 1.92	3.94 ± 0.25	3.55 ± 0.34

From Figure 2.4, it is obvious that rankings estimated from pairwise comparisons achieve better precision compared to the more precise ranking that is estimated from grades. Meanwhile, the variances of loss from pairwise comparisons are much smaller.

We believe there are two major factors that lead to such results. On one hand, with two items paired together, a context is self-contained in a pairwise comparison. This context provides a reference and makes the judgment easier. On the other hand, workers only need to figure out the relative order of two items to perform a pairwise comparison. In contrast, to grade an item, from psychological perspective, workers not only need to figure out the relative orders among the items, but also to quantify the differences and represent them with different values. The grading task requires more cognitive workload and thus is prone to errors. As a result, it is natural that pairwise comparisons can yield more precise rankings than grades.

Results of RQ3 illustrate that pairwise comparisons shall be adopted over grades for crowdsourcing ranking tasks.

2.5 Conclusion

In this chapter, we explore two types of data, i.e., numerical grades and pairwise comparisons, for crowdsourcing ranking tasks. We aim at understanding the human factors in grading, and comparing the precision of ranking from grades and pairwise comparisons. We propose three research questions, and investigate the questions by designing different tasks and sending them to Amazon Mechanical Turk, with all tasks having the same costs. We estimate a ranking from each task, and compare the ranking losses with four metrics.

The experiment results show that humans are more precise in grading, when they grade items with context. Also, humans are sensitive to the order of context, and a biased order in context hinders people from grading precisely. Compare to grades, pairwise comparisons are less prone to error and can yield more precise rankings, due to its nature of simplicity and self-contained context. In addition to the precision advantage, comparing a single pair of items requires less efforts than grading a sequence of items, hence a comparison task has a shorter turnaround time and is more widely accepted by crowd workers. Consequently, pairwise comparisons are adopted in the remaining part of this dissertation.

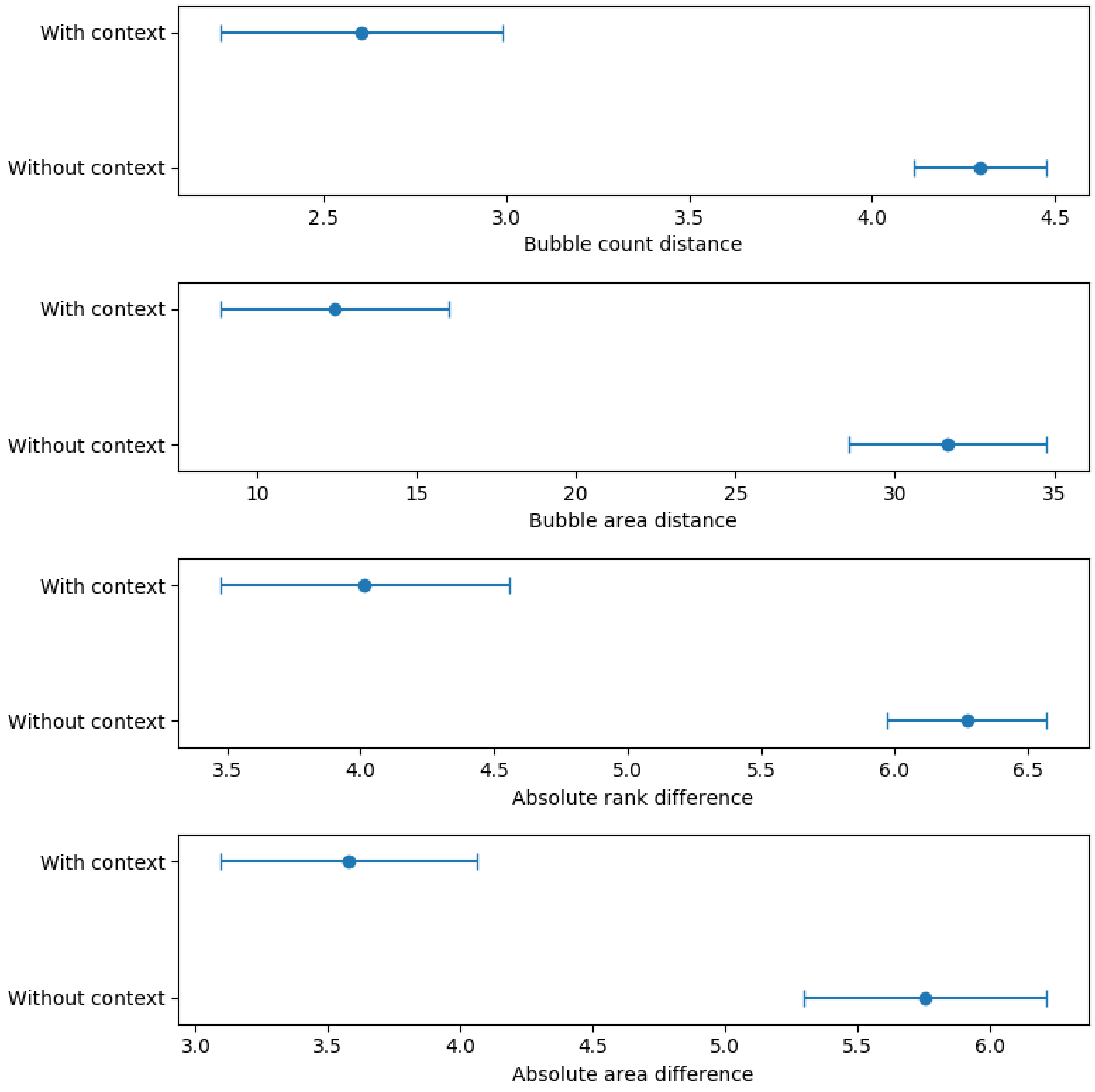

Figure 2.2: Average ranking loss and 95% confidence interval, from grading tasks with and without context

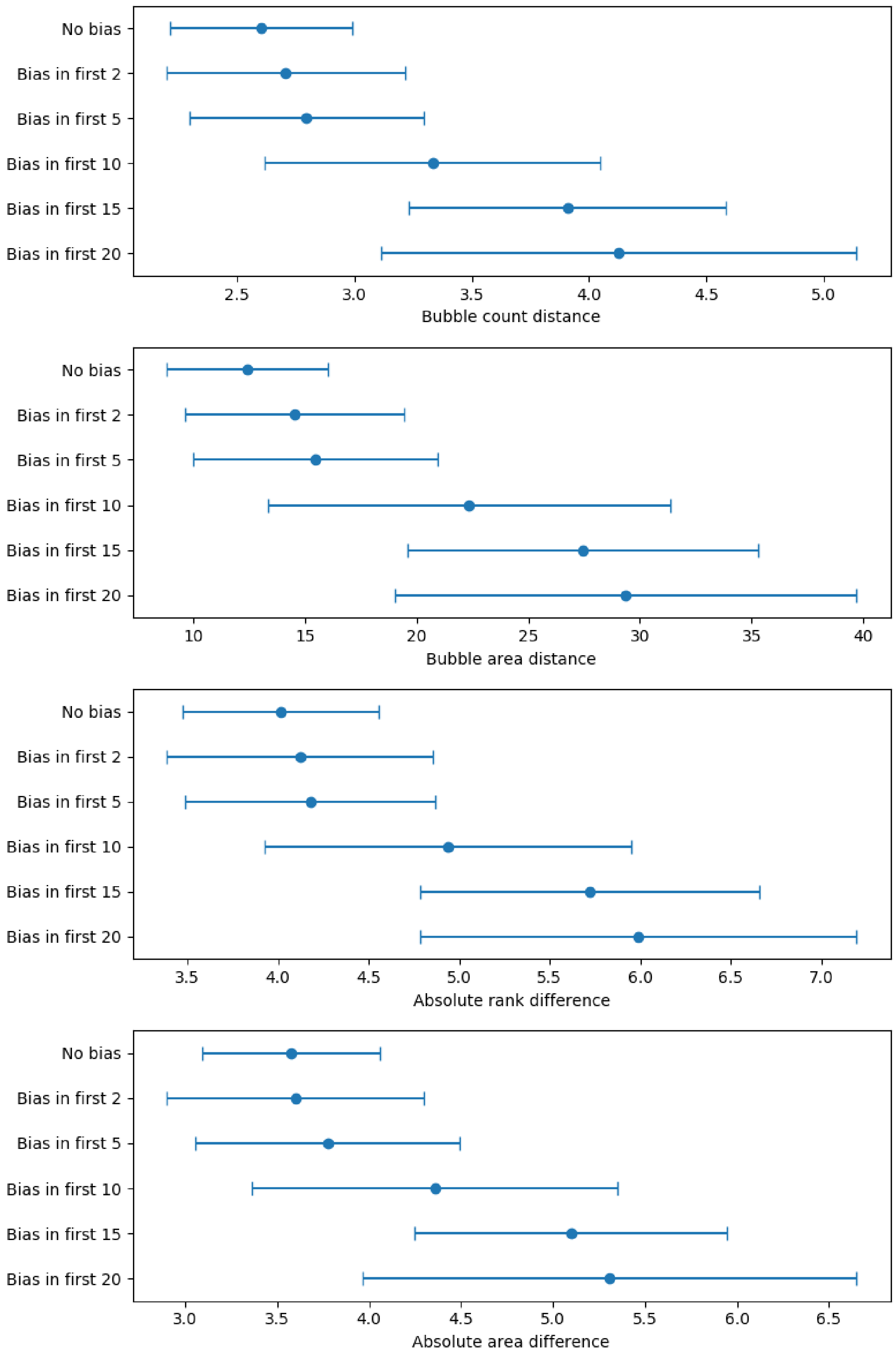

Figure 2.3: Average ranking loss and 95% confidence interval, from grading tasks with different biases in context

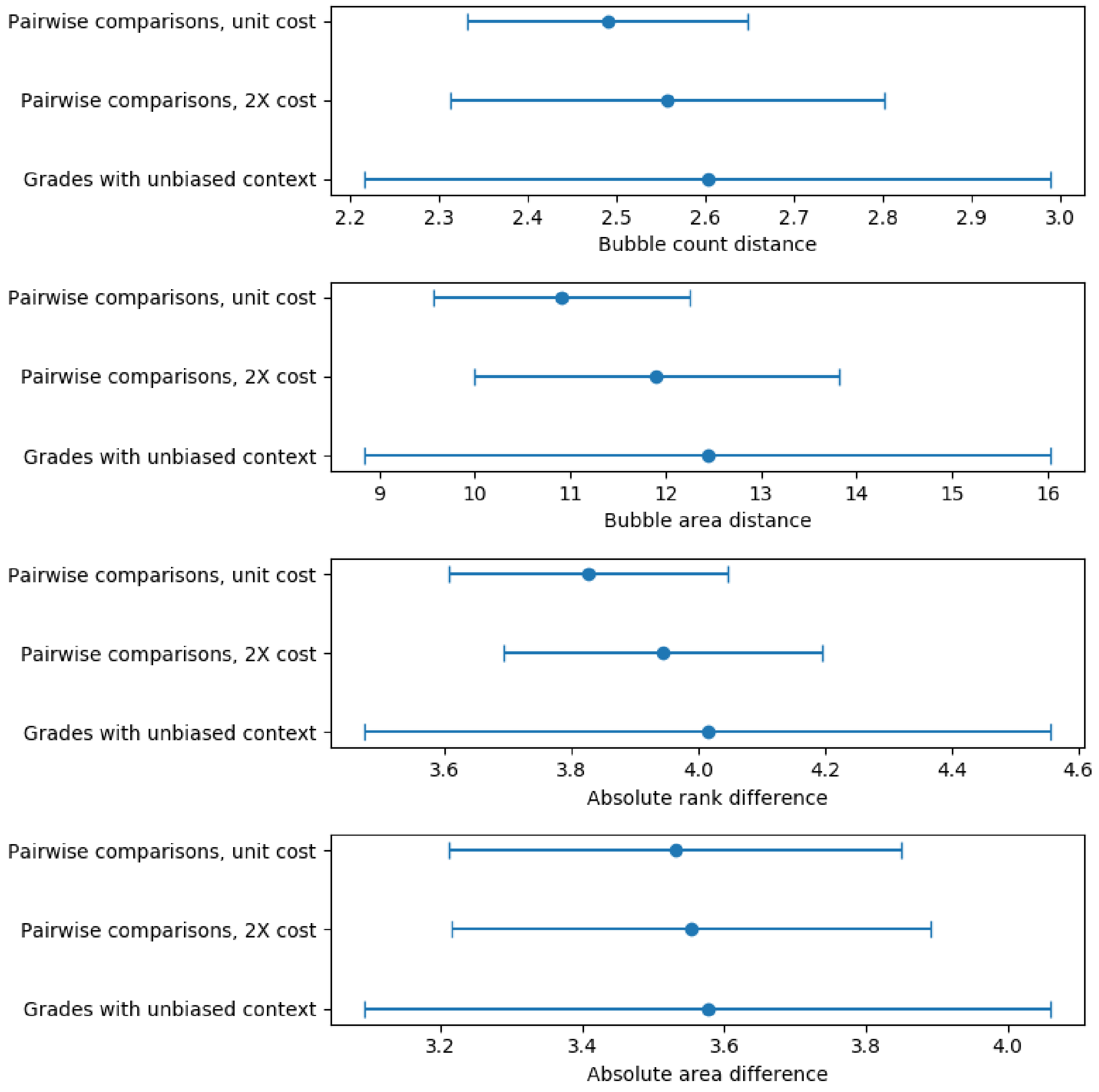

Figure 2.4: Average ranking loss and 95% confidence interval, from pairwise comparisons tasks and grading tasks without biased context

Chapter 3

Pairwise Comparison Selection for Top-heavy Rankings

3.1 Introduction

Identifying top items from large dataset is one of the key problems in real world. This problem widely exists in various applications, such a student admissions, selecting the recipients of scholarships, online contests, and so on. Crowdsourcing has been proven effective to get large scale of information with relatively low cost, and thus is suitable for this problem.

One common circumstance in top item discovery problem is that, the number of target top items is unknown or unidentifiable, while the influences of top items are larger than others. Such a scenario prevails, especially when the result is provided to diversified groups of users. For example, when a social media content creator, such as a video blogger, designs their homepage, they does not know how many videos the users will view, and the numbers vary among different users. Meanwhile, the high quality videos are usually placed at the top of homepage, so that they can attract more user clicks and views. Failing to place the high quality videos can result in disinterest from users and reduced traffic. The nature of top item discovery, when the number of target items is unknown, is essentially a ranking problem, where the errors at the top of ranking incur larger loss to the system. Based on our findings in Chapter 2, we will use pairwise comparisons for ranking.

Extensive research has been done on how to aggregate pairwise comparisons into accurate rankings, such as Mallows [72], nuclear norm minimization [43], Bradley-Terry [8], Glicko [44], TrueSkill [51] and so on [62][12]. However, it is still very challenging to efficiently obtain data with minimal computational cost. Therefore, we concentrate on studying active learning strategies for selecting pairs of items to be sent to the crowd for comparison, so that rankings will converge as quickly as possible to the correct rank. Specifically, in this dissertation we focus on 'top-heavy' rankings where an ordinal rank k is associated with a benefit proportional to its position. These type of geometric benefit distributions are typical in online settings, where higher rank commands geometrically higher visibility, and revenue [85][16][25].

In setting the problem, we assume that each item in a ranking has an intrinsic "quality", and we define a "top-heavy" version of ranking distance where the mis-placement of items is weighed according to $1/k^{\lambda}$ for rank k with $\lambda > 0$; this will be used to judge the speed of convergence of the proposed methods. Thus, errors in the head of the ranking will be weighed more heavily than errors in the tail. As ranking aggregation is not our primary focus, we perform ranking aggregation by applying the Glicko [44] and TrueSkill [51], which are well-established on-line aggregation algorithms.

We then formulate and compare two active learning strategies for selecting the next pairs of items to compare. In particular, we define the *loss* (or error) involved in each pair of items, and we consider and justify strategies that select pairs for comparison that have maximum loss, or that lead to maximum ranking change. We identify one such strategy, *maximum loss,* as the one that leads to overall best results, as demonstrated by our simulations.

These pair selection strategies are computationally expensive, as they require at each round the computation of many alternatives in order to select the best. To address this problem, we propose and develop efficient batched versions of the pair selection strategies, which deliver essentially the same performance while exhibiting much less computation time and complexity, with the dominant step being a sorting step.

In this chapter, our contributions can be summarized as follows:

- We define a distance to measure "top-heavy" rankings.
- We propose two active learning strategies for selecting pairs effectively which reduce ranking loss rapidly, in crowdsourcing environments.
- We propose an efficient batch algorithm with low computational resources.

After a review of related work in Section 3.2, we define the problem precisely, and describe the Glicko and TrueSkill methods for ranking aggregation (Section 3.3). In Section 3.4 we present our active learning methods for selecting pairs, and in Section 3.5 we present and analyze the experimental results.

3.2 Related Work

Obtaining data from crowdsourcing is widely applied in many fields, such as ranking, knowledge sharing, elections, opinion collection, and so on [61][27]. Pairwise comparison is one of the most commonly used crowdsourced input for ranking problems.

Much research has been done on how to aggregate comparison outcomes into a ranking [41, 53, 77, 83, 91, 101]. Generally, ranking aggregation methods can be categorized into three types: permutation-based, such as researches in Mallows [72] and CPS [83] models; matrix factorization, such as work in [43, 60]; and score-based probabilistic method, such as Plackett-Luce [69][80], Bradley-Terry [8], Thurstone [95], etc. Permutation methods are generally computational expensive, while matrix factorization methods do not have sufficient probabilistic interpretations. As a result, we use score-based methods for ranking aggregation in this chapter.

Score-based methods assign a score to each item, and use all scores to generate rankings. Among score-based methods learning from pairwise data, the Elo ranking method is perhaps the first Bayesian pairwise ranking algorithm [37], and it is widely used in ranking sports and estimating the underlying skills of players. A player's skill is assumed to follow a Gaussian distribution with two parameters as average skill level and player's uncertainty. Glickman extended Bradley-Terry model and updated player skills based on designed length of period, assuming same Gaussian distribution of player skills [44]. Trueskill by Microsoft is another Bayesian ranking system with Gaussian distribution assumption [51]. It extendes a Thurstone-Mosteller model which adds a latent variable as player performance.

Ranking aggregation via pairwise comparisons aims at computing a ranking for items that can represent all the comparison outcomes with minimum data disagreement. The problem that concerns us in this dissertation is how to optimally select pairs for comparison, so that a "good" ranking can be obtained with as few comparisons as possible, and thus, as efficiently as possible.

Active learning is an effective way to improve efficiency and promote performance. It has been recognized that by properly selecting items, learning tasks achieve better accuracy and require less data for training [24, 35, 66, 84]. There are mainly three types of active learning methods. The first one is uncertainty sampling, which targets at finding items that the system is most uncertain about [96][71]. Another one is minimizing expected loss, which focuses on searching for items that can reduce highest expected error [88]. Lastly, query by committee method looks for items that a set of learners (refers as committee) having largest disagreement with [76][42]. While extensive research has been performed on active learning, most of them are for binary or graded based problems [15,26,34,39,65,79,103].

Some research was done on active learning for ranking from pairwise comparisons. Donmez et al. applied their document selection algorithm to RankSVM and RankBoost [33]. Also using RankSVM, Yu proposed to add most ambiguous document pairs to training dataset [105]. Chen et al. proposed a framework to find reliable annotators and informative pairs jointly, which requires annotator quality information [18]. A maximum likelihood based algorithm was proposed by Guo et al. to locate the topmost item in a ranking [48]. Other research based on pairwise comparisons includes work done by Chu et al. [22], Ailon [2], Jamieson et al. [57] and so on. Notably, a majority of them focus on selecting annotators rather than items.

3.3 Top-Heavy Ranking

A ranking problem consists in sorting a set of items $S = \{I_1, I_2, \dots, I_n\}$ according to their quality. Ranking problems are solved via crowdsourcing when assessing the quality of an item requires human judgement, as is the case, for instance, when assessing the quality or appeal of videos, images or merchandise. We consider here ranking problems that are *top-heavy* in their reward: being ranked in position k has value proportional to $1/k$.

These top-heavy problems find application whenever the ranking is used to display the items to users. In such cases, a higher rank commands a larger amount of user engagement, which can be measured in item views, page visits, user votes and so forth, according to the nature of the items being ranked. As user attention is valuable (and can be monetized), we assume that the *value* of being ranked in position k is proportional to $1/k^\lambda$, for some $\lambda > 0$. We call this a λ-top heavy ranking. This is equivalent to assuming that user attention follows a Zipf distribution, an assumption that has been validated on the Web [54].

3.3.1 Ranking Quality

To measure the quality of a ranking, we introduce a measure of *distance* between top-heavy rankings. Our distance will give more weight to differences among top positions than to differences among positions in the tail of rankings. This reflects the intuition that errors in top rankings matter more than errors in the tails of rankings, as there is much more value in the top than in the tail. For instance, in a sport competition where athlete sponsorship is proportional to the inverse of the rank, it would obviously be worse to get the order wrong between the first and second positions than between the 101st and the 102nd.

Precisely, for our set of items $S = \{I_1, I_2, \ldots, I_n\}$, consider two rankings r and r' so that $r(i)$ is the position (the ordinal) of item I_i according to ranking r, for $1 \leq i \leq n$, and similarly for r'. We define the distance $d(r, r')$ between r and r' by:

$$d(r, r') = \frac{\sum_{i=1}^{n} \left| \frac{1}{r(i)^\lambda} - \frac{1}{r'(i)^\lambda} \right|}{n} , \tag{3.1}$$

where λ is the coefficient of the λ-top heavy ranking, and n is the number of items in S. Equation (3.1) can be understood as follows. If r is the correct ranking, and r' is another ranking, then $|\frac{1}{r(i)^\lambda} - \frac{1}{r'(i)^\lambda}|$ is the amount of value that item i receives in error, either in positive or negative. Thus, the quantity (3.1) represents the total value mis-allocation of ranking r', measured with respect to ranking r.

In particular, if r^* is the correct ranking, we denote by

$$\mathcal{L}(r) = d(r, r^*) \tag{3.2}$$

the *loss* of r, measured as its distance from optimality.

3.3.2 Learning Top-Heavy Rankings

Our goal consists in developing algorithms for learning top-heavy rankings via crowdsourcing. The algorithms we develop follow the following scheme:

1. We start with a random ranking.
2. At each round:
   (a) We select two items, and we ask a user to compare them.
   (b) We use the result of the comparison to update the rankings.

We rely on binary comparisons because they are the most elementary of comparisons, and they require less cognitive load on the user than multi-way comparisons. The goal of the above process is to converge as quickly as possible to the optimal ranking according to distance (3.1), that is, to reduce the loss of the rank as quickly as possible. As our distance is top-weighed, this means identifying the top items early.

In this dissertation, we focus on step 2a: the selection of the items to be compared. Once two items are compared, there are several classical methods for updating a ranking according to the comparison outcome; we describe two such alternative methods, Glicko and Trueskill, in the following. Our focus here is on how to choose the elements whose comparison will reduce ranking loss in the fastest possible way, and with low computational cost. Intuitively, choosing the elements to compare entails estimating which elements might be incorrectly ranked, keeping into account that errors at the top matter more than errors in the tail of the ranking. As the choice of the pairs to be compared uses information from the ranking update step, we first describe the ranking update step, and subsequently our proposed methods for item selection.

3.3.3 Ranking Update Methods

We describe here two ranking update methods: Glicko [44], and TrueSkill [51].

Glicko

The Glicko [44] method for ranking update models each item as having a score that has a Gaussian distribution. Thus, for each item I_i, Glicko stores the median μ_i and the standard deviation σ_i of the score. The model further assumes that if two items I_i, I_j have scores X_i and X_j (sampled from their respective distributions), then the probability that a user prefers I_i to I_j is proportional to

$$\frac{e^{\kappa(X_i - X_j)}}{1 + e^{\kappa(X_i - X_j)}} \, ,$$

where $\kappa > 0$ is an arbitrary scaling constant. This is known as the Bradley-Terry model of match outcomes [8]. In [44], the constant is $\kappa = (\log 10)/400$, and was chosen to scale the resulting scores so that they would approximate the scores of the Elo ranking for chess players [37]. The value of the constant is immaterial to the ranking being produced (it is simply a scaling for the scores), and we choose $\kappa = 1$ in our implementation.

With these choices, once a comparison is done, the Glicko model parameters are updated as follows. Denote with s_{ij} the outcome of comparison between item I_i and item I_j:

$$s_{ij} = \begin{cases} 1 & \text{if } i \text{ wins } j \\ 0 & \text{if } j \text{ wins } i \end{cases}$$

Let also for $i = 1, 2$,

$$g(\sigma_i^2) = \frac{1}{\sqrt{1 + 3q^2\sigma^2/\pi^2}}$$

The update formulas for the mean and standard deviations are:

$$\mu_i' \leftarrow \mu_i + \frac{q}{\frac{1}{\sigma_i^2} + \frac{1}{\delta_i^2}} g(\sigma_j^2)(s_{ij} - z_j)$$

$$\sigma_i'^2 \leftarrow \left(\frac{1}{\sigma_i^2} + \frac{1}{\delta_i^2} \right)^{-1}$$

where

$$z_j = \frac{1}{1 + e^{-g(\sigma_j^2)(\mu - \mu_j)}}$$

$$\delta_i^2 = \left[q^2 \left(g(\sigma_j^2) \right)^2 z_j (1 - z_j) \right]^{-1} .$$

The above update formulas are obtained from [44] as the special case in which time decay of the scores does not occur. Glicko models time decay of scores, so that as players remain inactive, their score median decreases, and their score standard deviation increases, modeling increased uncertainty about their abilities. Such time dependence is appropriate in modeling tennis and chess scores, but not in modeling the quality of items in crowdsourcing batches of short temporal duration.

TrueSkill

The TrueSkill rating system [51] also assumes a Gaussian belief on online game players' skills. It estimates skills by constructing a factor graph, connecting players that have had a match together, and using approximate message passing. Trueskill was developed for players belonging to teams; we present here a simplified version without teams (or, more precisely, where all teams have one player only), which corresponds to the problem at hand.

The skill of player i, denoted as s_i, is again assumed to be Gaussian-distributed with mean μ_i and standard deviation σ_i. The performance of player i, denoted by p_i, is also assumed to be Gaussian distributed, with mean value equal to s_i and standard deviation β, with β constant for all players. Thus, s_i models the intrinsic skills of player i, whereas p_i models the actual performance of player i in a specific match. Translated into our setting, s_i models the intrinsic quality of item i, and p_i models how the quality of an item is perceived by the worker performing the comparison. A tie between I_i and I_j occurs when $|p_i - p_j| \leq \varepsilon$ for a chosen ε, while I_i is preferred to I_j when $Y_i - Y_j > \varepsilon$.

If denote $\boldsymbol{Z}$ as whole set of match outcomes, the skills of all players as $\boldsymbol{S}$, the performances of all players as $\boldsymbol{P}$, and all differences between the performances of two players as $\boldsymbol{D}$, the general Bayesian inference problem is:

$$\begin{aligned} p(\boldsymbol{S}|\boldsymbol{Z}) &\propto \iint p(\boldsymbol{Z}, \boldsymbol{D}, \boldsymbol{P}, \boldsymbol{S})\, d\boldsymbol{D}\, d\boldsymbol{P} \\ &= \iint p(\boldsymbol{Z}|\boldsymbol{D}) p(\boldsymbol{D}|\boldsymbol{P}) p(\boldsymbol{P}|\boldsymbol{S}) p(S)\, d\boldsymbol{D}\, d\boldsymbol{P} \end{aligned}$$

where the joint density of the entire system is presented as a product of distributions. So, the problem consists in computing a marginal probability. To solve it, Trueskill implements an approximate method with a factor graph structure, where messages pass between nodes that correspond to the parameters of the problem. The message passing iterations stop when convergence is reached, yielding the player's skills, which correspond for us to the item qualities. An example of a factor graph is illustrated in Figure 3.1.

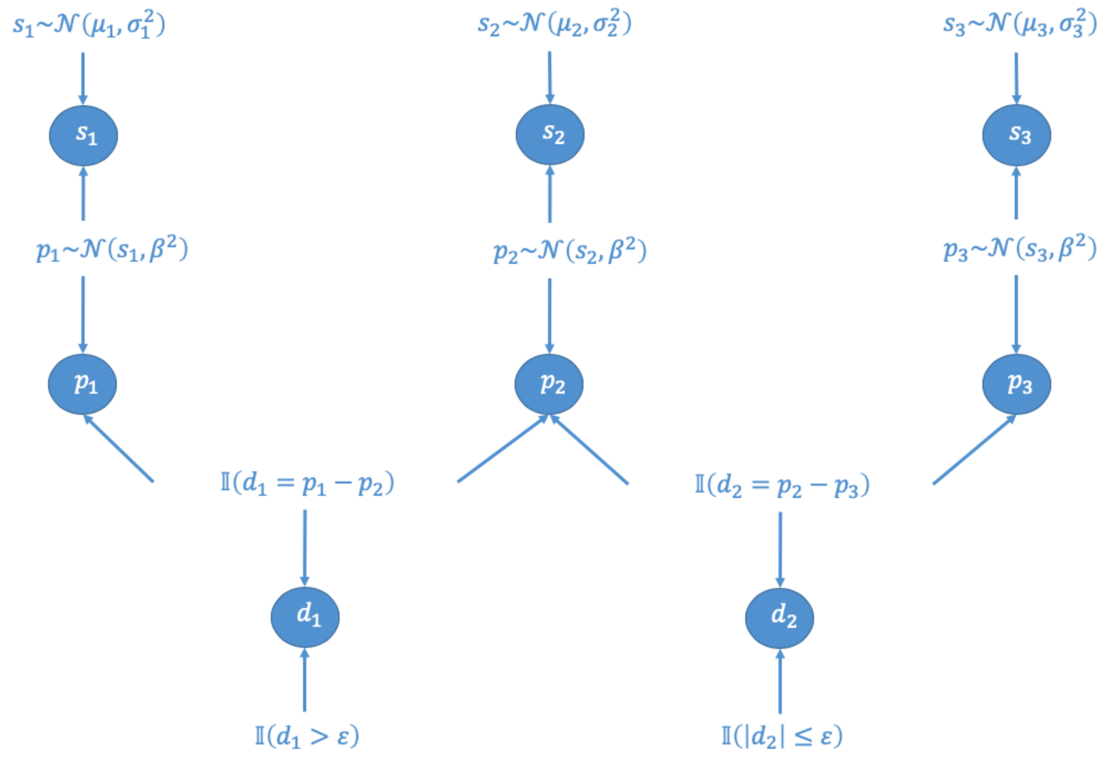
$s_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$
$s_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$
$s_3 \sim \mathcal{N}(\mu_3, \sigma_3^2)$
s_1
s_2
s_3
$p_1 \sim \mathcal{N}(s_1, \beta^2)$
$p_2 \sim \mathcal{N}(s_2, \beta^2)$
$p_3 \sim \mathcal{N}(s_3, \beta^2)$
p_1
p_2
p_3
$\mathbb{I}(d_1 = p_1 - p_2)$
$\mathbb{I}(d_2 = p_2 - p_3)$
d_1
d_2
$\mathbb{I}(d_1 > \varepsilon)$
$\mathbb{I}(|d_2| \leq \varepsilon)$

Figure 3.1: An example of TrueSkill factor graph

We apply TrueSkill after each comparison, updating the scores of the two items that took part in the comparison itself.

3.4 Active Learning for Pair Selection

Our active learning strategies aim at selecting the pair whose comparison will reduce ranking loss most effectively. In designing pair selection strategies, we focus on strategies that select items which might be incorrectly ranked, as comparing such items is likely to be more beneficial. We propose two strategies: maximum loss and maximum ranking changes.

3.4.1 Maximum Loss

The maximum loss ranking strategy selects for comparison at each step the pair of items that have the largest expected mis-allocation of reward. To make this expected value mis-allocation precise, we define the expected loss of a pair of items as the product of the probability of the two items having incorrect relative orders by the amount of error resulting from this situation. If we denote the item with higher rank as I_i and the one with lower rank as I_j, the probability of a pair having incorrect relative orders is essentially the probability of item I_j having a larger sampled value than that of item I_i from their distributions respectively. The amount of value mis-allocation resulting from incorrect relative orders in a top-heavy ranking is $\left|\frac{1}{r(i)} - \frac{1}{r(j)}\right|$. So, our strategy of selecting maximum expected loss selects the items i, j given by:

$$\arg\max_{(i,j)} \left\{ Prob.(\tilde{I}_i < \tilde{I}_j) * \left| \frac{1}{r(i)} - \frac{1}{r(j)} \right| \right\} \tag{3.3}$$

where $\tilde{I}_i, \tilde{I}_j$ are sampled values from distributions of item I_i, I_j, $Prob.(\tilde{I}_i < \tilde{I}_j)$ is the probability of I_i, I_j having incorrect relative orders, and $r(i)$, $r(j)$ are the ranking positions of the items. By the properties of Gaussian distributions, the probability of $r(i)$, $r(j)$ having incorrect relative orders can be calculated as:

$$Prob.(\tilde{I}_i < \tilde{I}_j) = Prob.(\tilde{I}_i - \tilde{I}_j < 0) = \Phi\left(\frac{-|\mu_i - \mu_j|}{\sqrt{\sigma_i^2 + \sigma_j^2}}\right)$$

where μ_i, μ_j are the means and σ_i, σ_j are the standard deviations of the distributions of I_i, I_i respectively.

3.4.2 Maximum Ranking Change

The maximum ranking change strategy selects the pair that will have the greatest impact on the current ranking. Intuitively, if two items change their rankings dramatically after a comparison that is inverse of their current relative order, it implies that a big problem exists potentially and the previous ranking is unstable or unreliable. In consideration of this, we propose a strategy, which selects the pair that will result in the largest expected ranking change after comparison, if the comparison shows an inverse order of the current order.

The expected ranking change for items receiving inverse comparisons is the product of the probability of two items receiving an inverse comparison and the expected amount of change to rankings after the comparison. Mathematically, the probability of two items receiving an inverse comparison is equal to the probability of two items having incorrect current order. With the same notations as first strategy, and denoting the expected amount of change for items receiving an inverse comparison as $g(I_i \prec I_j)$, we would like to select a pair of items i, j as follows:

$$\arg\max_{(i,j)} \left\{ Prob.(\tilde{I}_i < \tilde{I}_j) * g(I_i \prec I_j) \right\} \tag{3.4}$$

where $Prob.(\tilde{I}_i < \tilde{I}_j)$ is computed as before. The expected amount of change for items receiving an inverse comparison can be calculated by:

$$g(I_i \prec I_j) = \left| \frac{1}{r(i)} - \frac{1}{r(i)^{I_i \prec I_j}} \right| + \left| \frac{1}{r(j)} - \frac{1}{r(j)^{I_i \prec I_j}} \right|$$

where $r(i)^{I_i \prec I_j}$ and $r(j)^{I_i \prec I_j}$ are the updated rankings of item I_i and I_i, if item I_i loses to I_j in comparison, i.e., if item I_i and I_j receive an inverse comparison.

The problem with the selection (3.4) is that it requires computing the outcome of all possible pair comparisons. This is very expensive computationally: in order to get the future ranking positions of the items in a pair, the algorithm has to perform quality updates for both items, and sort all items for a new ranking. To address this problem, we analyze the equation and propose an approximate version.

Equation (3.4) can also be expressed as:

$$\begin{aligned} &Prob.(\tilde{I}_i < \tilde{I}_j) * g(I_i \prec I_j) \\ &= Prob.(\tilde{I}_i < \tilde{I}_j) * \left(\frac{1}{r(i)} - \frac{1}{r(i)^{I_i \prec I_j}} - \frac{1}{r(j)} + \frac{1}{r(j)^{I_i \prec I_j}} \right) \end{aligned} \tag{3.5}$$

Equation (3.5) holds because with a comparison of I_j winning I_i, the ranking update algorithms will update the ranking so that $r(j)^{I_i \prec I_j}$ ranks higher than or equivalent to $r(j)$, while $r(i)^{I_i \prec I_j}$ ranks lower than or equivalent to $r(i)$. This result is consistent with the intuition that one item shall get a lower ranking than its current ranking if loses, while the other shall rank higher if wins.

Assuming the ranking changes for both items are in same percentage of $\alpha > 0$, i.e.,

$$\begin{cases} \frac{1}{r(i)^{I_i \prec I_j}} = \frac{1}{r(i)} * (1+\alpha) \\ \frac{1}{r(j)^{I_i \prec I_j}} = \frac{1}{r(j)} * (1-\alpha) \end{cases}$$

then (3.5) can be further expressed as:

$$\begin{aligned} &Prob.(\tilde{I}_i < \tilde{I}_j) * \left(\frac{1}{r(i)} - \frac{1}{r(i)^{I_i \prec I_j}} - \frac{1}{r(j)} + \frac{1}{r(j)^{I_i \prec I_j}} \right) \\ &= Prob.(\tilde{I}_i < \tilde{I}_j) * \left(\frac{1}{r(i)} \cdot \frac{\alpha}{(1+\alpha)} + \frac{1}{r(j)} \cdot \frac{\alpha}{(1-\alpha)} \right) \end{aligned} \tag{3.6}$$

By first order Tyler Series, while $\alpha \to 0$, $\frac{\alpha}{(1+\alpha)} \to \alpha$, $\frac{\alpha}{(1-\alpha)} \to \alpha$. Assuming $\alpha \to 0$, (3.6) can be approximated by:

$$\begin{aligned} & Prob.(I_i \prec I_j) * \left(\frac{1}{r(i)} + \frac{1}{r(j)} \right) \cdot \alpha \qquad (3.7) \\ \propto\ & Prob.(I_i \prec I_j) * \left(\frac{1}{r(i)} + \frac{1}{r(j)} \right) \end{aligned}$$

We assume α approaches 0 because we believe that in a ranking system with sufficient comparisons, one single comparison shall not change any item's ranking dramatically. As an example, a tennis player will not get a huge ranking drop just because he loses one game.

In light of the above approximations, the maximum ranking change strategy selects the items i, j as follows:

$$\arg\max_{(i,j)} \left\{ Prob.(I_i \prec I_j) * \left(\frac{1}{r(i)} + \frac{1}{r(j)} \right) \right\} . \qquad (3.8)$$

Thus, the maximum ranking change pair selection strategy only relies on the current positions of items, and the computational complexity is significantly reduced.

3.4.3 Randomized Pair Drawing

Deterministically selecting for comparison the pair with the highest value, as done in (3.3) and (3.8), carries the risk of trapping the ranking in a local optimum. To deal with this problem, we propose to use a *randomized* version of our pair selection strategies. In the randomized version, we select each pair with probability proportional to the arguments of (3.3) and (3.8), respectively. The algorithms thus still focus on the most promising pairs for comparison, but their randomized nature makes them more robust. In experiments we performed, the randomized version of the selection algorithms always outperformed the deterministic ones, so that for this dissertation we opted for presenting the results only for the stochastic versions, for the sake of conciseness.

3.4.4 Batch Algorithm for Efficient Selection

In many online applications, the volume of items in a ranking is huge. With such size, it is very computational expensive, or even impossible to evaluate all candidate pairs. Also, updating the ranking after every user comparison sequentially is impractical and will impose vast burden on the system. As a result, in practice, instead of sequential algorithms, batch active learning methods are widely used. Therefore, we design a batch algorithm to reduce the computational cost and make our algorithms more efficient.

In the above pair selection algorithms, we observe that items close in rank are more likely to be selected, because they have a higher probability of being incorrectly ranked. We make use of this observation in our batch algorithm to narrow the candidate pair space. For any item I_i with position $r(i)$, when selecting its candidate pairing set, we do not consider all other items: rather, the batch algorithm only looks for a subset of items immediately below I_i and evaluates the corresponding pairs. In this way, we are able to cut down evaluation pairs dramatically. Since the ranking can be incorrect, we also include in the evaluation pairs items that are sampled randomly. This randomness improves the stability of the algorithm.

Once the batch algorithm determines candidate pairing sets for all items, it uses the same active strategies to calculate values for all pairs. Then, it selects a batch of pairs stochastically, where each pair is selected with probability proportional to its value. All selected pairs are presented to users concurrently for their comparisons. After the comparison outcomes are collected, the batch algorithm updates the ranking with all outcomes at once and a new batch will be selected. In this way, the expensive process of computing pair evaluations is performed only once for every batch, rather than once for every pair presented to users.

3.5 Experiments

3.5.1 Experiment Settings

We conduct experiments with simulated data as it can provide precise "true" ranking for evaluation. We assume there are 100 items, each of them has an underlying score with a Gaussian distribution, and sample its mean and variance respectively. Specifically, we sample all items' means from one Gaussian distribution as opposed to any heavy-tailed distribution, because in this dissertation, we focus on the intrinsic "qualities" of items that represent their strength or greatness, such as competitiveness of sports players, skills of online gamers, ratings of merchandise, reviews of restaurants; and a ranking is generated based on items' qualities. In contrast, a heavy-tailed distribution is usually assumed in relevance ranking problems, which sort items by their degree of relevance.

All pair selection strategies start with the same non-informative score estimation. Every time an algorithm requests a user comparison, we simulate it by sampling from true distributions of both items. The item with a larger sampled value is considered the winner. We conduct 20 experiments for each strategy, with 2,000 pair selections per experiment. All algorithms are evaluated with respect to the loss (3.2). In the experiments, we choose λ values at $0.5, 1, 2$ respectively.

3.5.2 Algorithm Loss Comparison

Figure 3.2 shows the loss decrease resulting from the proposed algorithms, for different λ coefficients. Along the x-axis is the number of selected pairs and along the

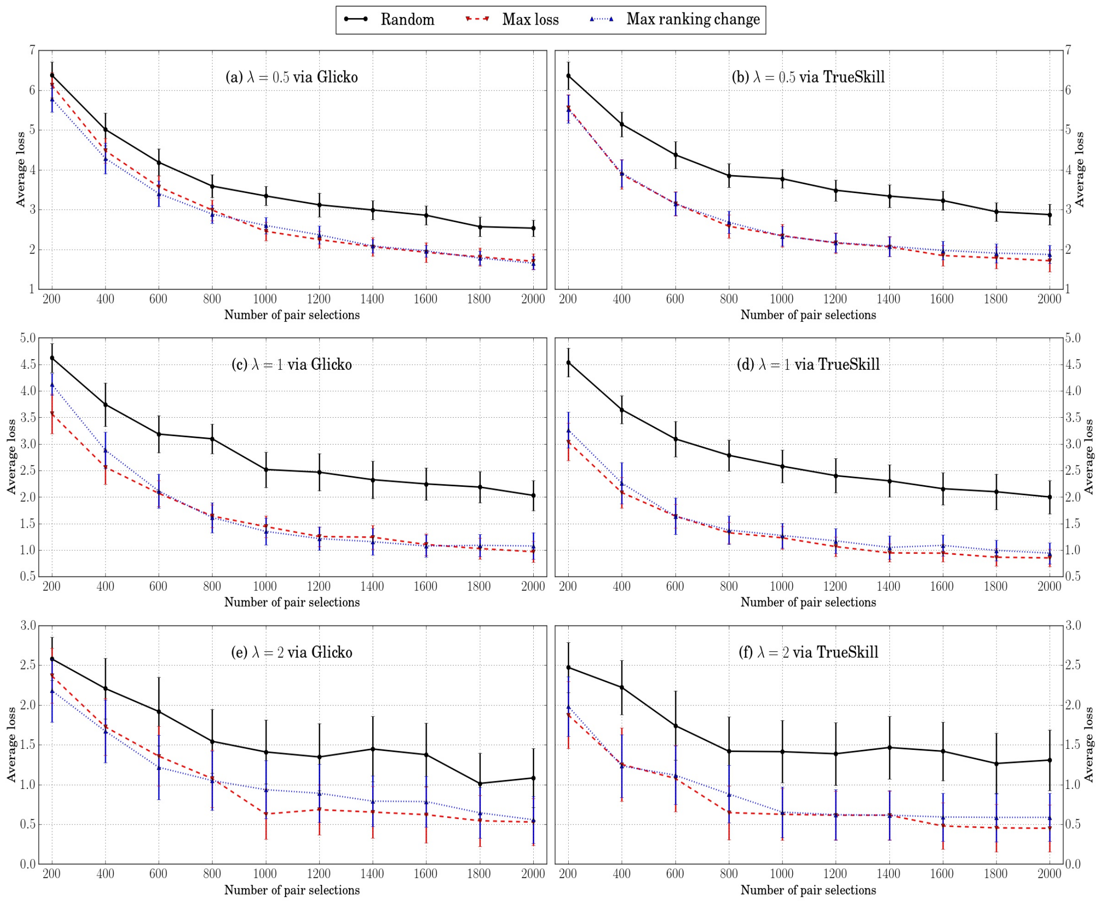

Figure 3.2: Loss comparison: active learning vs. random strategy.

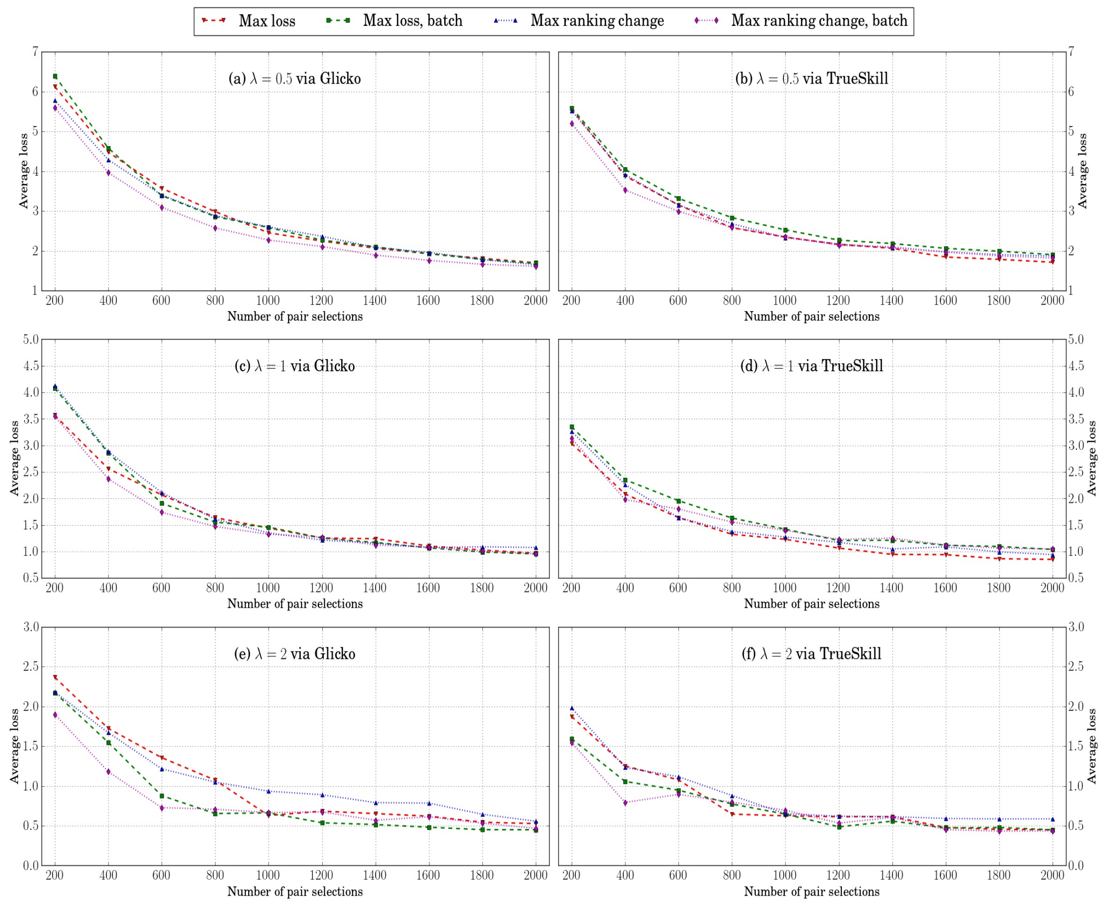

Figure 3.3: Loss comparison: sequential vs. batch.
The loss differences between sequential and batch versions are not significant.

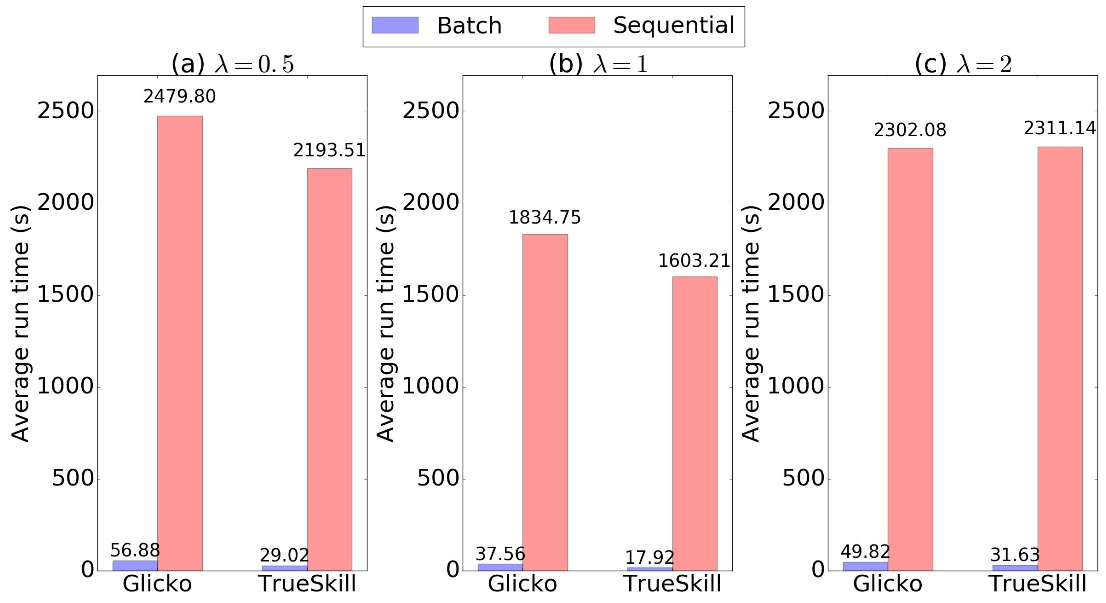

Figure 3.4: Average run time comparison: sequential vs. batch.

y-axis is the average loss per experiment. The error bars indicate 95% confidence intervals, and the random strategy is used as baseline. After each pairwise comparison, the item scores are updated via Glicko and TrueSkill respectively, and a new pair to compare is selected.

The results demonstrate that our two proposed algorithms perform significantly better than the baseline. With more pairwise comparisons, it is expected to see the ranking losses of all the algorithm decreasing, but the proposed algorithms converge to the true ranking and reduce loss much faster. The 95% confidence intervals between baseline and proposed algorithms rarely overlap, illustrating that the reduction in loss resulting from our algorithms is significant. Maximum loss and maximum ranking change algorithms get comparable results at the end of experiments, with maximum loss performing slightly better. We believe it is because the approximation used in maximum ranking change introduces some loss.

Relatively, the loss decrease is slower for algorithms evaluated with a λ coefficient of 2. We believe that it is due to the top-heavy reduction effect. With $\lambda > 1$, except for the very top of the ranking, the losses from mis-ranked elements become smaller, limiting the scope for loss decrease once the top-ranked items are correctly ranked. The divergence between algorithms and evaluation measure results in the performance decrease.

3.5.3 Sequential and Batch Version Comparison

We have also experimented with the batch version of our algoritms. We start with the same non-informative score estimation. We use 10% of total items as candidate pairing set for each item, with half of them sampled from the immediately lower ranked elements, and half of them sampled randomly. The batch size is set at 20, so the ranking will be updated after collecting 20 pairwise comparisons from users.

Sharing the same x-axis and y-axis as Figure 3.2, Figure 3.3 shows loss decrease of sequential and batch versions of each algorithm. It is evident that in spite of evaluating only 10% of all pairs, the batch versions achieve almost same performances as sequential versions. The results demonstrate that the batch versions of our algorithms are capable of getting comparable performance as the sequential versions, while dramatically reducing the computational cost. Figure 3.4 illustrates that the average run time per experiment of the batch versions are significantly smaller than that of the sequential versions. Collectively, Figures 3.3 and 3.4 prove that our batch algorithms are an effective solution to ranking problems with large dataset.

3.6 Conclusion

In this chapter, we propose two active learning strategies for selecting pairwise comparisons for top-heaving rankings, where top ranking items are considered more critical than items lower in the rankings. To address the computational challenge arising from large data volume, an efficient batch algorithm is proposed and applied. Our experimental results show that both active learning methods are effective at reducing ranking loss; overall, the maximum loss method achieves slightly better performance. We also demonstrate how our batch algorithm can achieve comparable loss decrease results while significantly reducing computational costs.

Chapter 4

Online Top-K Selection

4.1 Introduction

Selecting top-K items from large itemsets is a common task. It occurs when there is a constraint on the amount of resources available, for example, in college admissions, selecting the recipients of scholarships, newspaper highlight design, and so on. Broadly, there are two approaches to the top-K selection problem. One approach consists in determining a global ranking, and then returning the top-K elements of such a ranking. This approach can avail itself of a long line of work on crowdsourced ranking, starting with now classical approaches such as the BTL model [8], ELO ranking [37], and Trueskill [51]. This approach can be inefficient for the task of top-K selection, as it determines a full ranking even among items that are all selected, and all rejected. Nevertheless, the approach often has the advantage of being able to offer partial results: as these algorithms maintain and refine a global ranking (in various ways), at any point during the crowdsourcing work, they can provide a best estimate of the ranking, and thus, of the top-K elements.

Approaches that focus on selecting the top-K elements with crowdsourced inputs, rather than building a global ranking, have been proposed in [23, 29, 31, 82, 98] among others. As expected, these can be far more efficient than global-ranking algorithms, as the crowdsourcing work is aimed at separating the top-K elements from the rest, rather than obtaining a full ranking. However, these specialized top-K algorithms require the completion of a fixed amount of work, and do not provide useful partial results if less work is performed. For instance, if the algorithms of [31, 82] are interrupted, one is left with multiple lists of "best-among-M" elements, for comparison batches of size M. Short of randomly selecting K of these best-among-M elements, there is no obvious way of compiling a global top-K result if only part of the required work for the algorithm is performed. Yet, partial results are often useful. For example, in candiate selection, one may wish to extend offers to candidates as soon as they are determined to be in the top-K set. If the crowdsourcing work takes longer than expected, if a deadline makes it impossible to carry it to the end, or if the funds for it are exhausted, having a usable partial result is the difference between failure and (partial) success.

Here, we propose a crowdsourced top-K selection algorithm that dynamically refines a classification of the items into accepted (in the top-K) and rejected (not in top-K). Initially all items are unclassified; for each item, the algorithm maintains an estimated quality, and an estimated uncertainty on the quality. Given an error tolerance $\epsilon > 0$, the algorithm creates a batch of comparisons and sends them to the crowd. As the results from the crowd come in, the algorithm uses them to refine the classification, promoting to accepting all the items that fall into the top-K with probability at least $1 - \epsilon$, and demoting to rejecting all items that have probability at least $1 - \epsilon$ of *not* belonging to the top-K elements. The process is repeated until the top-K elements are accepted. Partial results can be obtained at any time by returning the accepted elements, along with the unclassified elements that have largest estimated quality. We show that not only can our algorithm provide partial results; it is also more efficient than previous top-K algorithms. The efficiency stems from the fact that items are classified into accepted or rejected as soon as enough evidence is accumulated, rather than after a prescribed amount of comparisons are performed. Hence, the ability of our algorithm to provide partial results, and its superior efficiency, both stem from the same underlying mechanism.

The contributions of this chapter can be summarized as follows:

- We present an online algorithm, named BetaTopK, for top-K item selection based on the dynamic classification of items into top-K, and rejected, items. The algorithm can at any time provide a best-effort selection of the top-K elements.
- We show that, for a fixed error tolerance $\epsilon > 0$ and an average probability of crowd correctness q, and for any top K/N percent, the expected number of required comparisons is $O(N \log N)$, where N is the total number of elements. The worst-case occurs for $K = N/2$.
- We propose several optimization techniques to improve BetaTopK.
- We evaluate the performance of our algorithm analytically and experimentally, from the perspectives of selection loss and number of required comparisons, and we show that it improves on the state of the art. Both global ranking and top-K selection approaches are used as benchmarks.

4.2 Related Work

Identifying top-K items is a critical problem in many areas, such as student admission, candidate recruiting, newspaper photo selection, and so on. Crowdsourcing provides an efficient way to collect input with relatively low costs, and it has been applied extensively in learning tasks, such as ranking, classifying, labeling, and more [7,32,74].

Extensive research has been done on obtaining a global ranking from crowdsourced data. Among the foundational work, we recall here the maximum likelihood estimation (MLE) model [40,55], which is one of the most commonly adopted schemas for global ranking. Another approach is the Bradley-Terry-Luce (BTL) model [8,68], which has provided the foundations of many crowdsourcing works, including the classical Elo rating [37]. For example, RankCentrality by Negahban et. al extended the BTL model and used random walk strategy on pairwise-choice Markov chain [78]. It is one of the most representative works in the family of spectral-based ranking paradigm, along with [75, 77, 86], to name a few. Another notable work stemming from BTL model is TrueSkill by Microsoft [51]. It improved Elo ranking [37] by constructing a factor graph and using approximate message passing, dramatically reducing the amount of input required to converge. Much work followed, including the CrowdBT model with BTL by Chen et al. [18], which took the quality of workers into consideration. Wauthier et. al [101] suggested a weighted counting algorithm on edges to achieve an approximate overall ranking with noisy input. The work in [1,2,9,10,13,50,57] considered the scenario in which input comparisons can be selected adaptively, and proposed different active learning strategies to improve a global ranking.

While comprehensive work exists for the problem of global ranking with crowd input, a direct application of those methods to a top-K problem increases the complexity unnecessarily, since determining a global ordering is more expensive than determining the set of top-K items.

Approaches that are tailored to top-K problem include [31, 82], which generalized the tournament approach into a multi-round ranking and selection process for determining the top-K elements. The algorithm by Davidson et al. [30] also adopted the tournament approach and utilized a heap-based method to get the top-K list. These three works can cope with errors in crowd comparisons, but they require the completion of a fixed amount of work, namely the ranking-and-selection rounds, before the final result becomes available. Eriksson proposed a top-K algorithm [38] by considering comparisons as graph edges. It performed a 'reverse' depth first search and eliminated the items with path length larger than K. However, this algorithm can return the items far from the top-K items if not all comparisons among the items are observed. Chen and Suh proposed a spectral MLE approach [20] that recovered the top-K items with high probability, with the strict assumption that the comparisons follow the BTL model. Jiang et al. [58] and Suh et al. [94] suggested top-K methods by extending the spectral MLE algorithm with different restrictions respectively, with the same BTL assumption. As the BTL model does not account for the possibility of users purposely lying, the BTL model and its variations fail to provide accurate estimations in practice [6, 28]. A Borda-count based algorithm was proposed in [90] to achieve the $O(N)$ bound of input comparisons, with the assumption of a Binomial distribution for the number of comparisons of each pair. Our algorithm is related, except it performs active learning, using a probability model to decide when to truncate the accumulation of Borda counts and discard or promote an item, while offering precise guarantees on the correctness probability of the top-K set. Chen et al. studied bounds for the amount of comparison input that is needed, under the setting in which comparisons can be requested uniformly for all items [19]. In contrast, our algorithm is an on-line one, where comparison requests are dynamically generated for the most needed items. In [23], active learning strategies for top-K problems were discussed. However, it assumed the probabilities of preferences between items are given, which is not realistic in practice.

4.3 Online Top-K Selection Algorithm

4.3.1 Definitions and Problem Settings

Our goal is to discover the top-K items from an itemset via crowdsourcing. We consider an itemset I, where each item $i \in I$ has an intrinsic quality q_i. For $K \leq |I|$, the top-K problem consists in selecting the subset $T_K \subseteq I$ of elements such that $|T_K| = K$, and such that $\sum_{i \in T_K} q_i$ is maximized. For a candidate top-K selection $U \subseteq I$, $|U| = K$, we do not measure the quality of U by its elementwise difference from T_K, as this would provide only a coarse measure of selection quality. Rather, we define the *selection loss* $\mathcal{L}(U)$ as the fraction of quality we fail to capture by selecting U rather than T_k:

$$\mathcal{L}(U) = \frac{\sum_{i \in T_k} q_i - \sum_{i \in U} q_i}{\sum_{i \in T_k} q_i} \tag{4.1}$$

This notion of loss encodes the general goal of top-K selection: with a constraint on the number of items that can be selected, the goal is to amass as large a total quality as possible. Such measurement finds its application in many real-world scenarios. For example, a graduate program plans to admit 10 students, but one of the top-10 applicants is not selected by mistake. In this case, selecting an applicant whose ranking is close to top-10 will be better than a bottom-ranked one. The loss defined in Equation (4.1) captures the degree of loss among different results, while an elementwise measurement fails to do so.

4.3.2 Algorithm foundations

The core idea of our algorithm consists in modeling the quality of an item via a Beta distribution, as the two parameters of Beta distribution account for the number of wins and losses of an item in comparing with others. If comparison outcomes are truthful, that is, if higher-ranked items win when they are compared against lower-ranked ones, a Beta distribution Beta$(\alpha, \beta)(x)$ represents the a-posterior probability that an item ranks at a fraction x of the total rank, given that $\alpha - 1$ comparisons were won, and $\beta - 1$ lost. Thus, a Beta distribution accounts both for the estimated position of an item in the ranking, and for the uncertainty with which the rank is known. The algorithm accumulates evidence on items until the items can be accepted in the top-K set, or discarded, with a specified amount of confidence.

If there are no comparison errors, the fractional rank of an item after w wins and l losses has an a-posteriori distribution Beta$(w+1, l+1)(x)$. If crowd comparisons are only correct with an average probability of q, then the a-posteriori fractional rank distribution is Beta$(w+1, l+1)\Big(xq + (1-x)(1-q)\Big)$. Thus, let

$$c(w, l; q)(x) = CDF_{\text{Beta}}\Big(xq + (1-x)(1-q); w+1, l+1\Big) \; ,$$

where $CDF_{\text{Beta}}(x; \alpha, \beta)$ is the cumulative distribution function of Beta distribution, with parameters of α, β at x. Given a probability bound $\epsilon > 0$ of making a wrong decision, we can promote the item to the top-K set when

$$c(w, l; q)\left(\frac{N-K}{N}\right) < \epsilon, \tag{4.2}$$

and we can eliminate the item when

$$1 - c(w, l; q)\left(\frac{N-K}{N}\right) < \epsilon. \tag{4.3}$$

Equation (4.2) and (4.3) suggest that a non top-K item can be selected into top-K list with a probability of at most ϵ, and likewise a top-K item can be eliminated with the same probability bound. Also, with Equation (4.2), the probability of an item in top-K

can be calculated any time before BetaTopK algorithm completes. As a result, the top-K list can be generated anytime by selecting the K items with highest probabilities. In this way, the online property on our algorithm is achieved.

We propose an initial algorithm, which determines the top-K list by eliminating items until only K items remain. Later, we will propose an optimized algorithm that performs item promotion and elimination concurrently.

4.3.3 The BetaTopK Algorithm

The BetaTopK algorithm is detailed in Algorithm 1. In this algorithm, the top-K list is achieved by eliminating unlikely candidates in iterations, and returning the remaining itemset. The notations of the algorithm are:

- I: the itemset, in which the number of remaining items reduces with the progress of algorithm.
- K: the number of top items to be selected.
- h: the number of comparisons an item receives in one iteration.
- ϵ: the probability of eliminating a top-K item erroneously.
- E_K: the top-K result set.
- T: an itemset that holds the candidates for elimination.

At the beginning of an iteration, each item starts with zero wins and losses. It is compared with h random opponents, and the number of wins and losses are recorded, as illustrated in the PerformComparisons procedure. The algorithm then eliminates the items with probabilities of being top-K smaller than ϵ. Such iteration repeats until K items remain. The remaining items are returned as the output of the algorithm.

Algorithm 1 BetaTopK Algorithm

```
Input: $I$, $K$, $h$, $\epsilon$
Output: $E_K$

 1: while $|I| > K$ do
 2:     PerformComparisons($I, h$)
 3:     $e_i \leftarrow c(w_i, l_i; q)\left(\frac{|I|-K}{|I|}\right)$
 4:     // Demotion: identify items to be eliminated.
 5:     $T \leftarrow \{i \in I \mid 1 - e_i < \epsilon\}$ // See Equation (4.3).
 6:     if $|T| > |I| - K$ then
 7:         $T \leftarrow |I| - K$ items in $T$ with largest $e_i$
 8:     $I \leftarrow I \setminus T$
 9: $E_K \leftarrow I$
10:
11: procedure PERFORMCOMPARISONS($I, h$):
12:     for all $i \in I$ do
13:         $w_i \leftarrow 0, l_i \leftarrow 0$
14:     $c \leftarrow 0$
15:     while $c < \frac{h}{2}$ do
16:         Shuffle $I$
17:         for $i = 1$ to $|I|$ do
18:             Get items $u, v$ at positions $i$ and $(i+1)\%|I|$
19:             Ask the crowd to compare $u, v$
20:             Update $w_u, l_u, w_v, l_v$
21:         $c \leftarrow c + 1$
```

At any time before the BetaTopK algorithm completes, a best-effort top-K list can be obtained via Algorithm 2; this algorithm calculates the probability of being top-K of each item in the remaining itemset, and returns the K items with the largest probabilities.

Algorithm 2 Online retrieval of top-K

```
Input: $I$, $K$
Output: $E_K$
1: $e_i \leftarrow c(w_i, l_i; q)\left(\frac{|I|-K}{|I|}\right)$
2: $E_K \leftarrow K$ items with smallest $e_i$
```

Theorem 1. *Given an error tolerance of ϵ and an average probability of q for crowd correctness, for selection of any K/N percentage of top items, the expected number of comparisons required in BetaTopK for an itemset of N items is $O(N \log N)$.*

Proof. We use the following notations in the proof:

- q: the average probability of correct comparisons by crowd.
- i: the true rank of an item.
- t: the number of wins an item has to receive minimally, in order to not be eliminated, i.e., to survive.
- P_i^d: the probability of item i being eliminated in one iteration.
- p_i : the probability of item i winning a randomly selected opponent.
- x_i: the total number of wins item i receives in one iteration.

In particular, the relationship between p_i and q is:

$$p_i = \frac{(N-i)q + (i-1)(1-q)}{N-1} \tag{4.4}$$

because for an item ranks at position i, $i-1$ items are superior to it and $N-i$ items are inferior to it. With crowd's average probability of comparing correctly being q, an item wins with an expectation of $(i-1)(1-q)$ times when being compared with the $i-1$ items that are superior to it, and wins an expectation of $(N-i)q$ times when comparing to the $N-i$ items that are inferior to it.

Based on Equation (4.4), we can derive the relationship between p_i and $p_{(N+1-i)}$ as follows:

$p_i + p_{(N+1-i)} = \frac{(N-i)q+(i-1)(1-q)}{N-1} + \frac{(N-(N+1-i))q+(N+1-i-1)(1-q)}{N-1} = \frac{(N-1)q+(N-1)(1-q)}{N-1} = q+1-q = 1$

To sum up:

$$p_i + p_{(N+1-i)} = 1 \tag{4.5}$$

An item ranking at position i is eliminated in an iteration with a probability of P_i^d. We can express this probability as a Bernoulli variable since the outcome of elimination is binary. By characteristics of Bernoulli distribution, the expected number of items being eliminated in one iteration is:

$$\sum_{i=1}^{N} P_i^d \tag{4.6}$$

We can prove Theorem 1 if the lower bound of $\frac{\sum_{i=1}^{N} P_i^d}{N}$ is independent of N. The rationale behind is that, the lower bound of $\frac{\sum_{i=1}^{N} P_i^d}{N}$ being independent of N means a fraction of N items is eliminated in one iteration, regardless of the value of N. Equivalently, it means to reduce the remaining number of items to a constant value, $\log N$ iterations are required. Note that in one iteration, all items receive h comparisons respectively, making the complexity of the number of comparisons in one iteration $O(N)$. Consequently, the expected number of comparisons required in BetaTopK is $O(N \log N)$.

We consider $K \leq \frac{N}{2}$ in the following proof. However, the proof remains valid when $K > \frac{N}{2}$, because in that situation, the top-K problem is equivalent to the problem of selecting the bottom-$(N-K)$ items, so the same proof applies.

The proof focuses on showing that the lower bound of $\frac{\sum_{i=1}^{N} P_i^d}{N}$ is independent of N herein. For an item that ranks at posision i, P_i^d can be calculated as: $P_i^d = Pr(x_i < t)$. Note that any comparison that item i receives is a Bernoulli trial with a winning probability of p_i. As a result, the total number of wins that item i receives in one iteration with h comparisons, i.e., x_i, is a random variable that follows a Binomial distribution with parameters h and p_i. With x_i expressed as a Binomial random variable, $\sum_{i=1}^{N} P_i^d$ can be calculated by:

$$\begin{aligned}\sum_{i=1}^{N} P_i^d = \sum_{i=1}^{N} Pr(x_i < t) &= \sum_{i=1}^{N}\sum_{j=0}^{t-1} Pr(x_i = j) \\ &= \sum_{i=1}^{N}\sum_{j=0}^{t-1} Binomial(j; h, p_i)\end{aligned} \tag{4.7}$$

where $Binomial(k; n, p)$ is the probability mass function of Binomial distribution: $Binomial(k; n, p) = Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$.

By the nature of Binomial distribution, since $p_i + p_{(N+1-i)} = 1$, we have:

$$Binomial(j; h, p_{(N+1-i)}) = Binomial(h-j; h, p_i) \tag{4.8}$$

With Equation (4.8), Equation (4.7) can be calculated with the top half of all items:

$$\begin{aligned}&\sum_{i=1}^{N}\sum_{j=0}^{t-1} Binomial(j; h, p_i) = \\ &\sum_{i=1}^{\frac{N}{2}}\sum_{j=0}^{t-1} \Big\{ Binomial(j; h, p_i) + Binomial(h-j; h, p_i) \Big\}\end{aligned} \tag{4.9}$$

By characteristics of Bernoulli distribution, $\sum_{j=0}^{t-1}\Big\{Binomial(j;h,p_i)+Binomial(h-j;h,p_i)\Big\}$ is the total area of two sides of a Binomial probability mass function, and its value ranges from 0 to 1, i.e.:

$$0 \le \sum_{j=0}^{t-1}\Big\{Binomial(j;h,p_i)+Binomial(h-j;h,p_i)\Big\} \le 1 \tag{4.10}$$

Based on information theory, $\sum_{j=0}^{t-1}\Big\{Binomial(j;h,p_i)+Binomial(h-j;h,p_i)\Big\}$ is minimized when $p_i = 0.5$. Consequently, Equation (4.9) has a lower bound as follows:

$$\begin{gathered}\sum_{i=1}^{\frac{N}{2}}\sum_{j=0}^{t-1}\Big\{Binomial(j;h,p_i)+Binomial(h-j;h,p_i)\Big\} \ge \\ \frac{N}{2}\cdot\sum_{j=0}^{t-1}\Big\{Binomial(j;h,0.5)+Binomial(h-j;h,0.5)\Big\}\end{gathered} \tag{4.11}$$

With Equation (4.10) and (4.11), we can derive the lower bound of the expected number of items being eliminated in one iteration, i.e. $\sum_{i=1}^{N} P_i^d$ below:

$$\sum_{i=1}^{N} P_i^d \ge N\cdot\frac{1}{2}\cdot\sum_{j=0}^{t-1}\Big\{Binomial(j;h,0.5)+Binomial(h-j;h,0.5)\Big\} \tag{4.12}$$

where $\frac{1}{2}\cdot\sum_{j=0}^{t-1}\Big\{Binomial(j;h,0.5)+Binomial(h-j;h,0.5)\Big\}$ is a value within the range of $[0,\frac{1}{2}]$.

If we can show that the variable t is independent of N, by Equation (4.12), we can assert that the lower bound of $\frac{\sum_{i=1}^{N} P_i^d}{N}$ is independent of N.

As aforementioned, t is the number of minimal wins an item has to receive to survive, and an item will be eliminated when the area of Beta distribution to the right of threshold, i.e., $\Big(1-\frac{K}{N}\Big)q+\frac{K}{N}(1-q)$, is smaller than ϵ. In other words, an item will be eliminated if the area of Beta distribution to the left of the threshold is greater than

$1 - \epsilon$. With h comparisons, t can be calculated as:

$$t = \min_t \left\{ CDF_{Beta}\left(\left(1 - \frac{K}{N}\right)q + \frac{K}{N}(1-q); t+1, h-t+1\right) \leq 1 - \epsilon \right\} \quad (4.13)$$

and $t >= 0$.

From above equation, it is apparent that the value of t depends on the percentage of K out of N, i.e., $\frac{K}{N}$, but is independent of N.

With Equation (4.13) showing that t is independent of N, we can conclude that the lower bound of $\frac{\sum_{i=1}^{N} P_i^d}{N}$ is independent of N. As a result, Theorem 1 is proved.

■

4.3.4 Optimized BetaTopK Algorithm

We present here an optimized version of the BetaTopK Algorithm 1, which improves the initial BetaTopK algorithm in two aspects. First, it adds a promotion stage to each iteration, which selects items with probabilities of being in top-K larger than $1 - \epsilon$. As a result, the promotion stage can identify top-K items with a bounded error of ϵ. Second, instead of resetting all wins and losses at the beginning of each iteration, the optimized BetaTopK keeps the comparisons from previous iterations as long as the comparisons are performed among the remaining items. In this way, the algorithm can converge faster due to better utilization of historical information.

The optimized algorithm is illustrated in Algorithm 3. In the algorithm, we use the previous notations with the following additions:

- K_{gap}: the number of top-K items yet to be discovered.
- T: an itemset that holds temporary results. It is used for different purposes at selection and elimination stages.

The reason for establishing a temporary itemset T is that, if the number of top-K candidates retrieved in an iteration exceeds what the algorithm expects, or if the number of elimination candidates exceeds what the algorithm targets at, the algorithm should only select or eliminate a subset of the items in T. Keeping the temporary itemset allows the algorithm to further calculate the target subset.

4.4 Evaluation

We compare our algorithm with two algorithms. One is TrueSkill, which is a crowdsourcing, global ranking algorithm [51]. Another one is the algorithm proposed in [31], which is a top-K selection algorithm.

We conduct experiments with simulated crowdsourcing data as it can provide a precise evaluation of the loss, since we know the items' true qualities. The items' true qualities, or scores, are sampled from a Gaussian distribution. The crowd's perception on each item is expressed as a Gaussian distribution as well. The mean of the latter Gaussian distribution represents the crowd's average perception on the item, and it is set to the item's true quality. The crowd's uncertainty about the item is expressed as the variance of this Gaussian distribution, and is sampled uniformly.

Every time an algorithm requests a comparison from crowd, we simulate it via a two-step process. First, we determine the true winner, that is, the winner as perceived by a honest (or truthful) crowd worker. To this end, we sample the perception distributions of the two items; the item with the larger sample value is the true winner. Second, we return the true winner as comparison winner with probability q, and the true loser as comparison winner with probability $1 - q$; this models the crowd accuracy q. In real-world applications, q can be determined by the average rate of correct answers from crowd empirically.

Algorithm 3 Optimized BetaTopK Algorithm

```
Input: $I$, $K$, $h$, $\epsilon$
Output: $E_K$

1: $E_K \leftarrow \emptyset$, $K_{gap} \leftarrow K$
2: while $|I| > K_{gap}$ and $|E_K| < K$ do
3:     PerformComparisonsCumulative($I, h$)
4:     $e_i \leftarrow c(w_i, l_i; q)\left(\frac{|I|-K_{gap}}{|I|}\right)$
5:     // Promotion: identify top-K items.
6:     $T \leftarrow \{i \in I \mid e_i < \epsilon\}$ // See Equation (4.2).
7:     if $|T| > K_{gap}$ then
8:         $T \leftarrow K_{gap}$ items in $T$ with smallest $e_i$
9:     $E_K \leftarrow E_K \cup T$
10:    // Demotion: identify items to be eliminated.
11:    if $|E_K| < K$ then
12:        $T \leftarrow \{i \in I \mid 1 - e_i < \epsilon\}$ // See Equation (4.3).
13:        if $|T| > |I| - K_{gap}$ then
14:            $T \leftarrow |I| - K_{gap}$ items in $T$ with largest $e_i$
15:    $I \leftarrow I \setminus E_K$, $I \leftarrow I \setminus T$, $K_{gap} \leftarrow K - |E_K|$
16: if $|E_K| < K$ then
17:     $E_K \leftarrow E_K \cup I$
18:
19: procedure PERFORMCOMPARISONSCUMULATIVE($I, h$):
20:     for all $i \in I$ do
21:         $w_i \leftarrow$ number of wins item $i$ obtains, from the comparisons with other
            remaining items in $I$
22:         $l_i \leftarrow$ number of losses item $i$ receives, from the comparisons with other
            remaining items in $I$
23:     $c \leftarrow 0$
24:     while $c < \frac{h}{2}$ do
25:         Shuffle $I$
26:         for $i = 1$ to $|I|$ do
27:             Get items $u, v$ at positions $i$ and $(i+1)\%|I|$
28:             Ask the crowd to compare $u, v$
29:             Update $w_u, l_u, w_v, l_v$
30:         $c \leftarrow c + 1$
```

In the evaluations, we repeat every experiment 20 times, and we report the average and confidence interval of the results.

4.4.1 Loss Evaluation

In this section, we evaluate our algorithm by comparing its loss with TrueSkill, when they receive the same amount of comparisons. The loss defined in (4.1) is used for the comparison.

A experimental evaluation of top-K ranking and selection algorithms is available in [106]. It showed that TrueSkill, a global ranking algorithm, performs excellently in top-K problems among the existing algorithms. As a result, it is used to compare with our algorithm. TrueSkill estimates the qualities of items with any incoming comparisons, and uses the qualities to update the ranking constantly. It is an online algorithm and does not have a definite number of comparisons for termination. To compare our algorithms to TrueSkill, we feed to the algorithms itemsets with the same size and characteristics, and we record the loss at regular intervals along the working of the (online) algorithms. In our experiments, 100 items are compared against each other, with the goal of selecting the top 10 items. Each algorithm takes a total of 2000 comparisons, and the losses are recorded in every 200 comparisons. The hyper-parameter h of BetaTopK is set to 4 in the experiments.

Figure 4.1 shows the losses of algorithms in two separate settings. The left side of the figure shows the losses with the assumption of a perfectly truthful crowd, i.e., $q = 1.0$. The right side shows the losses with a more realistic assumption, where the crowd can make mistakes and judge incorrectly, with the average correctness q being 0.8. In Figure 4.1, the x-axis is the number of pairwise comparisons performed, and the y-axis is the average loss. The error bars indicate the 95$\%$ confidence intervals.

The results in Figure 4.1 demonstrate that BetaTopK achieves much lower selection

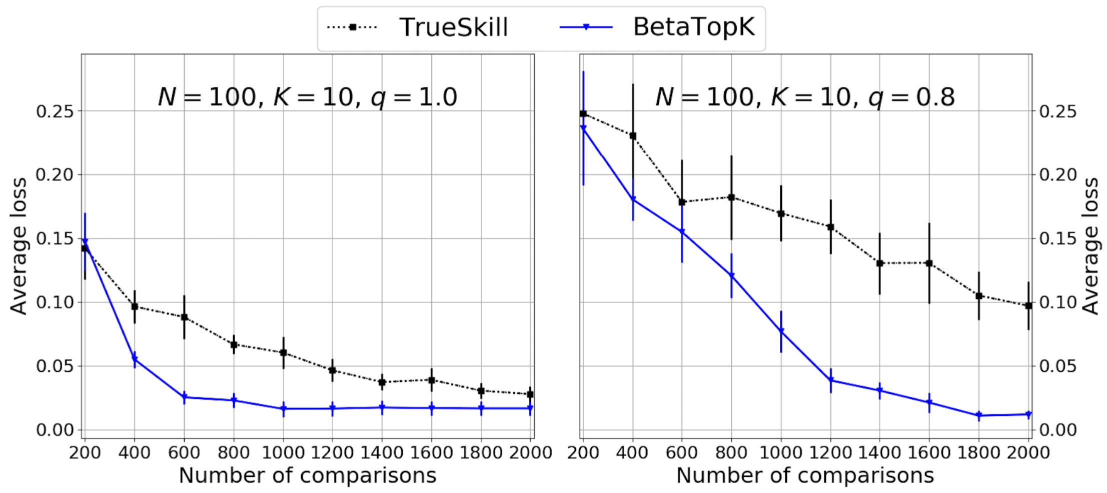

Figure 4.1: Loss comparisons

loss than TrueSkill algorithm, when they consume the same number of comparisons. In both settings, the differences between two algorithms become evident after about 400 comparisons, revealing the fast converge characteristic of BetaTopK. It can be observed that the 95% confidence intervals between TrueSkill and BetaTopK rarely overlap, illustrating that our algorithm is able to consistently obtain substantial improvement. Overall, the results demonstrate that the proposed algorithm can achieve significant loss reduction and return high quality top-K results comparing to one of the best-performed top-K algorithms.

4.4.2 Evaluation: Number of Comparisons

We compare the number of comparisons required for algorithm termination by BetaTopK and the algorithm proposed in [31], when they achieve the same level of error in results. Essentially, comparing the numbers of inputs needed when two algorithms obtain results with same error, is equivalent to comparing the losses when two algorithms receive the same amount of inputs. However, as many top-K selection algorithms can not produce an intermediate result before algorithm completes, it is impossible to report the loss of them in progress. As a result, we choose to compare the number of inputs required by the two algorithms, given the same level of error.

The algorithm in [31] is a recursive offline algorithm that selects the top-K items by dividing the ranking task into reduction and endgame phases. In the reduction phase, the itemset is partitioned into small sets that can be sent out for ranking by crowd workers; the items that end up in top position in such rankings are used to construct a reduced itemset. The process continues, recursively reducing the itemset size, until the complete set of items can be sorted. In the endgame phase, the complete sorting of the reduced itemset is used to reconstruct a top-K selection of the original itemset, following the recursion backwards. We refer this algorithm as RecurTopK due to its recursive structure. RecurTopK needs a pre-established number of comparisons before the top-K set is constructed.

Complexity Analysis

The number of comparisons required by RecurTopK can be derived analytically. We summarize the conclusions here, and present the detailed derivation in Appendix A;

In the following, let:

- s: size of a ranking task in the reduction phase, i.e., the number of elements that RecurTopK can send to each crowd worker for comparison.
- η: probability of an item being incorrectly ranked in a recursive call.
- $CDF_{Binomial}(k; n, p)$: the cumulative distribution function of Binomial distribution. $CDF_{Binomial}(k; n, p) = Pr(X \leq k) = \sum_{i=0}^{\lfloor k \rfloor} \binom{n}{i} p^i (1-p)^{n-i}$, where X is a random variable.

In a perfect scenario where the crowd is perfectly truthful, i.e., $q = 1.0$, we have:

Reduction phase: $\frac{N}{s-1} \cdot s \log s$.

Endgame phase, where K $<$ s : $\left\lceil \log_s(\frac{N}{K}) \right\rceil \cdot \left(\frac{(K+1)\cdot K}{2}\right) \log\left(\frac{(K+1)\cdot K}{2}\right)$.

Endgame phase, where K $\geq$ s : $\left\lceil \log_s(\frac{N}{K}) \right\rceil \cdot sK \log(sK)$.

In scenarios where the crowd can be inaccurate, i.e. $q < 1.0$, we get:

Reduction phase: $R \cdot \frac{N}{s-1} \cdot s \log s$, where

$$R = \min_R \left\{ CDF_{Binomial}\left(\left\lfloor \frac{R}{2} \right\rfloor; R, q \right) \leq \frac{\eta}{\log s} \right\}.$$

Endgame phase, where K $<$ s :

$$U \cdot \left\lceil \log_s(\frac{N}{K}) \right\rceil \cdot \left(\frac{(K+1)\cdot K}{2}\right) \log\left(\frac{(K+1)\cdot K}{2}\right),$$

$$\text{where } U = \min_U \left\{ CDF_{Binomial}\left(\left\lfloor \frac{U}{2} \right\rfloor; U, q \right) \leq \frac{\eta}{\log\left(\frac{(K+1)\cdot K}{2}\right)} \right\}.$$

Endgame phase, where K $\geq$ s : $U \cdot \left\lceil \log_s(\frac{N}{K}) \right\rceil \cdot sK \log(sK)$,

$$\text{where } U = \min_U \left\{ CDF_{Binomial}\left(\left\lfloor \frac{U}{2} \right\rfloor; U, q \right) \leq \frac{\eta}{\log(sK)} \right\}.$$

In the equations above, we relate the probability η of error in a recursive call to the error bound ϵ of BetaTopK via:

$$\eta = \frac{E\epsilon}{2(D+1)} \tag{4.14}$$

where $D = \left\lceil \log_s(\frac{N}{K}) \right\rceil - 1$ is the number of recursive calls required by the RecurTopK algorithm.

For BetaTopK, we perform the evaluation both analytically and experimentally. In the analytical evaluation, we calculate the expected number of comparisons required by

BetaTopK for completion. In the experimental evaluation, we demonstrate the average number of comparisons required by BetaTopK.

Analytical Comparison

Table 4.1 shows the number of crowdsourced comparisons required by RecurTopK (denoted as Recur) and BetaTopK(denoted as Beta) in Algorithm 1 analytically, with various K, N and q settings. The analytical results of Algorithm 1 use the lower bound of expected number of eliminated items to calculate the number of comparisons needed. The value of ϵ is set at 0.01 for BetaTopK, and the same level of error is set for RecurTopK via (4.14).

We can see that in most settings, BetaTopK requires less comparisons than RecurTopK, and as K grows, the advantage of BetaTopK becomes more significant. This observation is in line with our expectation. The BetaTopK Algorithm 1 generates the top-K list by dynamically reducing candidates, promoting or discarding them. When K is large, many candidates can be easily promoted, or many easily discarded. In contrast, the RecurTopK performs a more thorough analysis of the top-K elements; in particular, the number of recursive calls is independent of K. As a consequence, the algorithm does not exploit the ease with which some elements can be determined to be in the top-K set. As a result, the differences between BetaTopK and RecurTopK get larger, the larger the value of K.

Furthermore, Table 4.1 reveals that with the same setting of K, N and s, BetaTopK gains a larger advantage when the crowd is more prone to mistakes, i.e., when q is smaller. This demonstrates that the dynamic nature of BetaTopK enables the more efficient handling of crowd errors.

Table 4.1: Number of comparisons required by RecurTopK and BetaTopK analytically

Parameters		$q = 0.6$		$q = 0.7$		$q = 0.8$		$q = 0.9$		$q = 1.0$	
		Recur	**Beta**	**Recur**	**Beta**	**Recur**	**Beta**	**Recur**	**Beta**	**Recur**	**Beta**
$K < s$	$N = 100$, $K = 10$, $s = 15$	21,392	20,533	6,427	13,759	3,682	8,496	2,209	5,073	736	4,231
	$N = 100$, $K = 50$, $s = 60$	1,430,950	11,632	367,752	10,371	150,092	8,406	68,769	7,554	13,753	6,780
	$N{=}10,000$, $K = 50$, $s = 60$	3,376,557	2,264,560	1,400,874	596,339	657,249	179,726	431,882	146,932	86,376	75,750
$K \geq s$	$N = 100$, $K = 10$, $s = 5$	42,844	20,533	12,206	13,759	6,530	8,496	3,692	5,073	854	4,231
	$N{=}10,000$, $K = 50$, $s = 30$	3,048,114	2,264,560	1,261,284	596,339	640,196	179,726	310,544	146,932	82,413	75,750
	$N{=}10,000$, $K = 500$, $s = 30$	13,444,818	2,162,972	3,994,380	1,003,450	2,126,615	604,319	1,192,733	337,860	258,851	219,106
	$N{=}10,000$, $K{=}5,000$, $s = 30$	331,364,256	1,163,242	80,919,361	1,037,168	33,986,324	840,666	18,308,137	755,406	2,629,951	370,000

Experimental Comparison

In Table 4.2 we compare the analytical evaluation of the BetaTopK Algorithm 1 (denoted as A), with the experimental evaluation of the optimized BetaTopK Algorithm 3 (denoted as E). The comparison uses the same $\epsilon = 0.01$ as in the analytical results. This comparison shows the value of the optimizations, which in many cases reduce the number of crowdsourced comparisons by a factor of two.

In Figure 4.2, we plot the number of crowdsourced comparisons required by RecurTopK, BetaTopK Algorithm 1 with analytical result (termed BetaTopK Analytical), and the BetaTopK Algorithm 3 with experimental result (termed BetaTopK Experimental) when $N = 10,000$. The x-axis is the average probability of correctness of the crowd, and y-axis is the number of comparisons. The results in Table 4.2 demonstrate the dramatic reduction in number of crowdsourced comparisons required by BetaTopK compared to RecurTopK. The results also show that the number of comparisons required by

Table 4.2: Comparisons: number of comparisons required by BetaTopK, analytically and experimentally

Parameters	$q = 0.6$		$q = 0.7$		$q = 0.8$		$q = 0.9$		$q = 1.0$	
	A	**E**	**A**	**E**	**A**	**E**	**A**	**E**	**A**	**E**
$N = 100$, $K = 10$	20,533	18,500	13,759	6,220	8,496	3,440	5,073	1,840	4,231	1,520
$N = 100$, $K = 50$	11,632	10,460	10,371	9,420	8,406	8,200	7,554	5,760	6,780	3,600
$N = 10,000$, $K = 50$	2,264, 560	580, 000	596, 339	204, 000	179, 726	118, 000	146, 932	100, 000	75,750	74,000
$N = 10,000$, $K = 500$	2,162, 972	2,130, 000	1,003, 450	634, 000	604, 319	320, 000	337, 860	186, 000	219, 106	120, 000
$N = 10,000$, $K = 5,000$	1,163, 242	1,158, 000	1,037, 168	968, 000	840, 666	832, 000	755, 406	748, 000	678, 031	458, 000

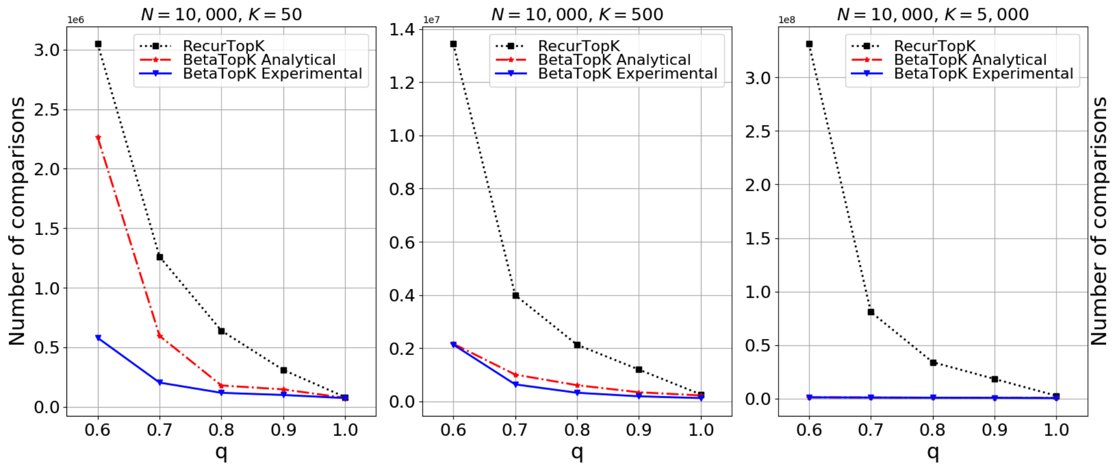

Figure 4.2: Number of comparisons required by each algorithm

experiments is significantly smaller than that of analytical results.

4.5 Conclusion

In this chapter, we propose an online top-K algorithm in crowdsourcing environments, which focuses on maximizing the total qualities of the selected top-K items, while is indifferent to their ordering. We prove that the expected number of comparisons required by this algorithm is $O(N \log N)$ given error tolerance and average probability of crowd correctness. Our analysis shows that the algorithm can achieve comparable selection outcome while requires fewer comparisons than existing algorithms; the advantage is especially marked when the value K of selected items is a non-negligible proportion of the total number of items. We also show how the algorithm can be optimized, further improving its performance.

Chapter 5

Conclusion and Future Work

In this dissertation, we explore the crowdsourcing data for ranking, and propose novel algorithms to discover top items with crowdsourcing inputs. Our work and contributions can be summarized as follows.

We explore two major data types in crowdsourcing ranking problems, numerical grades and pairwise comparisons. We investigate the human factors in grading tasks, and compare the precisions of two data types for crowdsourcing ranking problems. Experiment results show that humans tend to grade more precisely when there is context for grading, and a context without biased order helps humans to grade more accurately. Experiment results also suggest that pairwise comparisons can achieve better ranking results compared to grades, and thus are preferred for crowdsourcing ranking problems.

When the number of target top items is unknown or identifiable, the task of top item discovery is equivalent to a top-heaving ranking problem, where top ranked items are more critical than the lower ranked items. In this dissertation, we focus on selecting the next pair to be sent to the crowd to compare, so that the ranking can converge to the correct result as fast as possible. We propose two strategies, i.e., maximum loss algorithm and maximum ranking change algorithm. Our experiment results show that while both algorithms are effective in reducing ranking loss with the same amount of inputs, the maximum loss method achieves slightly better performance. We further develop an efficient batch algorithm to address the computational challenge arising from large volume of pair candidate calculation. We demonstrate that our batch algorithm can achieve comparable loss decreasement, while reduce computational time and complexity significantly.

When the number of target top items is known, the top item discovery task turns into a top-K selection problem. We propose an online top-K algorithm, which focuses on maximizing the total qualities of the selected top-K items, while is indifferent to their internal ordering. Our algorithm is able to return partial results at any point of time before algorithm completes. We prove that given error bound and crowd's average probability of correctness, the expected number of comparisons required by the algorithm is $O(N \log N)$. We further show several optimization techniques for our algorithm. Experiment results demonstrate that our algorithm can achieve comparable selection outcome, while it requires fewer comparisons than existing algorithms. The advantage is especially marked when the value K is a non-negligible proportion of the total number of items N.

We see several directions to extend our work. We can further develop advanced pair selection algorithms, and examine how our top-K algorithm will be influenced when the pair selection algorithms are applied. We are also interested in understanding the crowd worker's trustworthiness, and modeling it with the progress of crowd's responses. Last but not least, inspired by the results of chapter 2, we believe it is of vital importance to develop algorithms to model the factors that impact human correctness in crowdsourcing environments. Such research will tremendously improve our understanding and utilization of crowdsourcing systems.

Appendix A

Complexity Analysis of RecurTopK

In this section, we show the detailed complexity analysis of RecurTopK algorithm. We continue to use the same notations listed in Section 4.3.3. In addition, we summarize the notations for RecurTopK as follows:

- s: the size of a ranking task in reduction phase, i.e., the number of elements that RecurTopK can send to each crowd worker for comparison.
- η: probability of an item being incorrectly ranked in one recursive call.
- $\bar{q}$:
- D: total number of recursive calls required by RecurTopK algorithm.
- E: total number of iterations required by BetaTopK algorithm.
- R: total number of times one ranking task is sent to the crowd in reduction phase.
- U: total number of times one ranking task is sent to the crowd in endgame phase.

As illustrated in [31], $D = \left\lceil \log_s(\frac{N}{K}) \right\rceil - 1$, and the total number of ranking tasks required by RecurTopK is:

$$R \cdot \frac{N}{s-1} + U \cdot \left\lceil \log_s(\frac{N}{K}) \right\rceil \tag{A.1}$$

where $R \cdot \frac{N}{s-1}$ is the total number of ranking tasks required in the reduction phase, and $U \cdot \left\lceil \log_s(\frac{N}{K}) \right\rceil$ is the total number of ranking tasks required in the endgame phase.

Moreover, BetaTopK and RecurTopK achieve the same level of correctness when:

$$(1-\epsilon)^E = (1-\eta)^{2(D+1)} \iff E\log(1-\epsilon) = 2(D+1)\log(1-\eta) \tag{A.2}$$

Note that for any value x that is smaller than but approximately equal to one, i.e., $x < 1$ and $x \approx 1$, we have $\log x \approx x - 1$. Since both ϵ and η are positive values close to 0, we can get $\log(1-\epsilon) \approx -\epsilon$ and $\log(1-\eta) \approx -\eta$. So Equation A.2 can be approximated as:

$$E(-\epsilon) = 2(D+1)(-\eta) \iff \eta = \frac{E\epsilon}{2(D+1)} \tag{A.3}$$

In this analysis, we consider a comparison-based sorting algorithm with a complexity of $O(N\log N)$ for any ranking task, since $O(N\log N)$ is the optimal complexity for any comparison-based sorting algorithms. Two scenarios are analyzed: a perfect scenario where the crowd is perfectly truthful, i.e., $q = 1.0$; and a more realistic, imperfect scenario where the crowd can be inaccurate, i.e., $q < 1.0$.

A.1 Reduction Phase

In this phase, every ranking task consists of s items. The total number of comparisons required in the reduction phase is the product of the total number of ranking tasks in this phase and the expected number of comparisons per task. Based on the first item of Equation A.1, the total number of comparisons required in reduction phase can be expressed as:

$$R \cdot \frac{N}{s-1} \cdot s \log s \tag{A.4}$$

Perfect scenario where $q = 1$

In a perfect scenario, a ranking task becomes a single sorting task, i.e., $R = 1$. With Equation (A.4), the total number of comparisons required in reduction phase, under perfect scenario, is:

$$1 \cdot \frac{N}{s-1} \cdot s \log s \tag{A.5}$$

Imperfect scenario where $q < 1$

In an imperfect scenario, due to the incorrect comparisons, each ranking task has to be performed multiple times to reach the same level of correctness as BetaTopK, i.e., $R > 1$. We demonstrate the way to calculate the value of R in the following paragraphs.

For a ranking task of size s, with the complexity of $O(N \log N)$, each item is expected to compare with $\log s$ other items. As defined at the beginning of Appendix A, the probability of the aggregated comparison outcome of a pair being correct is $\bar{q}$. Consequently, after comparing with $\log s$ items, the probability of an item being ranked correctly is $\bar{q}^{\log s}$. Given the lower bound of $1 - \eta$ for the probability of an item being correctly ranked in a recursive call, the following formula is established:

$$\bar{q}^{\log s} \geq 1 - \eta \iff \log s \cdot \log \bar{q} \geq \log(1 - \eta) \tag{A.6}$$

When $\bar{q}$ is smaller but approximately equal to 1, $\log \bar{q} \approx \bar{q} - 1$. Similarly, since η is a positive but approximately equal to 0, $\log(1 - \eta) \approx -\eta$. As a result, Equation A.6

can be approximated as:

$$\log s \cdot (\bar{q} - 1) \geq -\eta \iff \bar{q} \geq 1 - \frac{\eta}{\log s} \tag{A.7}$$

As described earlier, in an imperfect scenario, each pair is compared R times. The comparisons are aggregated to a final comparison outcome. If more than half of the comparisons are correct, the aggregated comparison outcome will be correct. For example, at least 3 out of 5 comparisons towards a pair need to be correct, in order to obtain a correct aggregated outcome for the pair.

Meanwhile, each of the R comparisons is correct with a probability of q. Mathematically, $\bar{q}$ a binomial variable, which includes R Bernoulli trials with a probability of q for each trial, i.e., a binomial variable with parameters of q and R. Therefore, $\bar{q}$ can be calculated by:

$$\bar{q} = \sum_{i=\left\lceil \frac{R}{2} \right\rceil}^{R} Binomial(i; R, q) = 1 - CDF_{Binomial}\left(\left\lfloor \frac{R}{2} \right\rfloor; R, q\right) \tag{A.8}$$

Plug $\bar{q}$ into Equation (A.7), we get:

$$\begin{aligned} 1 - CDF_{Binomial}\left(\left\lfloor \frac{R}{2} \right\rfloor; R, q\right) &\geq 1 - \frac{\eta}{\log s} \\ \iff CDF_{Binomial}\left(\left\lfloor \frac{R}{2} \right\rfloor; R, q\right) &\leq \frac{\eta}{\log s} \end{aligned} \tag{A.9}$$

We can compute the minimum value of R that keeps BetaTopK and RecurTopK at the same level of correctness with Equation (A.3) and (A.9). After getting the value of R, we can plug it into Equation (A.4), and get the total number of comparisons required in reduction phase of RecurTopK, for the imperfect scenario where $q < 1$.

A.2 Endgame Phase

In the endgame phase, there are two circumstances: $K < s$ and $K \geq s$. The complexities of the two circumstances are different, and we will investigate them respectively.

A.2.1 Endgame Phase: $K < s$.

In the maxima top-K list of any recursive call, the top item and its $K-1$ subsequent items in the same partition may be the true top$K$ items, so for the partition where the top item resides, K items are placed into the candidate set. Similarly, the second-top item and its $K-2$ subsequent items in the same partition may be the true top$K$, so that $K-1$ items of the partition that includes the second-top item are placed into the candidate set; so on and so forth. Consequently, the number of items in the candidate set in a single recursive call is $K + (K-1) + \cdots + 1 = \frac{(K+1)\cdot K}{2}$.

The total number of comparisons required in the endgame phase is the product of the total number of ranking tasks in this phase and the expected number of comparisons per task. Based on the second item of Equation A.1, with a sorting complexity of $O(N \log N)$, the total number of comparisons required in endgame phase, when $K < s$, is:

$$U \cdot \left\lceil \log_s(\frac{N}{K}) \right\rceil \cdot \left(\frac{(K+1)\cdot K}{2}\right) \log \left(\frac{(K+1)\cdot K}{2}\right) \tag{A.10}$$

Perfect scenario where $q = 1$

In a perfect scenario, each ranking task is simply a sorting task, i.e., $U = 1$. So the number of comparisons in endgame phase, under perfect scenario when $K < s$, is:

$$1 \cdot \left\lceil \log_s(\frac{N}{K}) \right\rceil \cdot \left(\frac{(K+1)\cdot K}{2}\right) \log \left(\frac{(K+1)\cdot K}{2}\right) \tag{A.11}$$

Imperfect scenario where $q < 1$

In an imperfect scenario, for the similar reason as in reduction phase, each pair is compared multiple times, i.e., $U > 1$. We will show the method to calculate U in the following.

In any recursive call, for a ranking task of size $\frac{(K+1)\cdot K}{2}$, with the ranking complexity of $O(N \log N)$, each item is expected to compare with $\log \frac{(K+1)\cdot K}{2}$ other items. For the same reason of Equation A.6, given a lower bound of $1 - \eta$ for probability of an item ranked correctly, we have:

$$\bar{q}^{\log \frac{(K+1)\cdot K}{2}} \geq 1 - \eta \tag{A.12}$$

With the same approximation method used in Equation A.7, we can approximate Equation A.12 with:

$$\bar{q} \geq 1 - \frac{\eta}{\log\left(\frac{(K+1)\cdot K}{2}\right)} \tag{A.13}$$

Using the same reasoning of Equation A.8, $\bar{q}$ is a binomial variable and can be expressed as:

$$\bar{q} = 1 - CDF_{Binomial}\left(\left\lfloor \frac{U}{2} \right\rfloor; U, q\right)$$

Replace $\bar{q}$ in Equation A.13 with above equation, we obtain:

$$\begin{aligned} 1 - CDF_{Binomial}\left(\left\lfloor \frac{U}{2} \right\rfloor; U, q\right) &\geq 1 - \frac{\eta}{\log\left(\frac{(K+1)\cdot K}{2}\right)} \\ \iff CDF_{Binomial}\left(\left\lfloor \frac{U}{2} \right\rfloor; U, q\right) &\leq \frac{\eta}{\log\left(\frac{(K+1)\cdot K}{2}\right)} \end{aligned} \tag{A.14}$$

With the calculated value of U, we can plug it into Equation (A.10), and get the total number of comparisons required in endgame phase of RecurTopK, for the imperfect scenario when $K < s$.

A.2.2 Endgame Phase: $K \geq s$.

In the maxima top-K list of any recursive call, for any top $K - s + 1$ item, all its subsequent items in the same partition may be in true top K list. Since $K \geq s$ and the dominant parameter is K, we can use sK as the approximate size of ranking task in each recursive call. As a result, the number of comparisons required in endgame phase, when $K \geq s$, is:

$$U \cdot \left\lceil \log_s(\frac{N}{K}) \right\rceil \cdot sK \log(sK) \tag{A.15}$$

Perfect scenario where $q = 1$

In a perfect scenario, apparently U is still equal to 1. So the number of comparisons in endgame phase, under perfect scenario when $K \geq s$, is:

$$1 \cdot \left\lceil \log_s(\frac{N}{K}) \right\rceil \cdot sK \log(sK)$$

Imperfect scenario where $q < 1$

In an imperfect scenario, the analysis is very similar to the endgame phase when $K < s$. The only difference is that, the size of a ranking task in a recursive call is sK, as opposed to $\frac{(K+1)\cdot K}{2}$. For the conciseness of this dissertation, we do not repeat the similar process, but list the key results here.

The analysis results are:

$$\bar{q} \geq 1 - \frac{\eta}{\log(sK)}$$

$$\begin{aligned} \bar{q} = 1 - CDF_{Binomial}\left(\left\lfloor \frac{U}{2} \right\rfloor; U, q\right) \geq 1 - \frac{\eta}{\log(sK)} \\ \iff CDF_{Binomial}\left(\left\lfloor \frac{U}{2} \right\rfloor; U, q\right) \leq \frac{\eta}{\log(sK)} \end{aligned} \tag{A.16}$$

The value of U can thus be calculated. Plug in the value of U into Equation A.15, we can get the total number of comparisons required in endgame phase of RecurTopK, for the imperfect scenario when $K \geq s$.

Bibliography

[1] Arpit Agarwal, Shivani Agarwal, Sepehr Assadi, and Sanjeev Khanna. Learning with Limited Rounds of Adaptivity: Coin Tossing, Multi-Armed Bandits, and Ranking from Pairwise Comparisons. In *Conference on Learning Theory*, pages 39–75, June 2017.

[2] Nir Ailon. An Active Learning Algorithm for Ranking from Pairwise Preferences with an Almost Optimal Query Complexity. *J. Mach. Learn. Res.*, 13(1):137–164, January 2012.

[3] Nalini Ambady and Robert Rosenthal. Half a minute: Predicting teacher evaluations from thin slices of nonverbal behavior and physical attractiveness. *Journal of Personality and Social Psychology*, 64(3):431–441, 1993. Place: US Publisher: American Psychological Association.

[4] Ammar Ammar and Devavrat Shah. Efficient Rank Aggregation Using Partial Data. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 355–366, New York, NY, USA, 2012. ACM.

[5] Yukino Baba and Hisashi Kashima. Statistical quality estimation for general crowdsourcing tasks. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '13, pages 554–562, Chicago, Illinois, USA, August 2013. Association for Computing Machinery.

[6] T. Ballinger and Nathaniel Wilcox. Decisions, Error and Heterogeneity. *Economic Journal*, 107(443):1090–1105, 1997.

[7] Daren C. Brabham. Crowdsourcing. In *The International Encyclopedia of Organizational Communication*, pages 1–6. American Cancer Society, 2017.

[8] Ralph Allan Bradley and Milton E. Terry. Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika*, 39(3/4):324–345, 1952.

[9] Mark Braverman, Jieming Mao, and S. Matthew Weinberg. Parallel Algorithms for Select and Partition with Noisy Comparisons. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 851–862, New York, NY, USA, 2016. ACM.

[10] Mark Braverman and Elchanan Mossel. Noisy Sorting Without Resampling. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 268–276, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[11] Séebastian Bubeck, Tengyao Wang, and Nitin Viswanathan. Multiple Identifications in Multi-Armed Bandits. In *International Conference on Machine Learning*, pages 258–265, February 2013.

[12] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005.

[13] Robert Busa-Fekete, Balazs Szorenyi, Weiwei Cheng, Paul Weng, and Eyke Huellermeier. Top-k Selection based on Adaptive Sampling of Noisy Preferences. In *International Conference on Machine Learning*, pages 1094–1102, February 2013.

[14] Róbert Busa-Fekete, Eyke Hüllermeier, and Balázs Szörényi. Preference-based rank elicitation using statistical models: the case of mallows. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1071–II–1079, Beijing, China, June 2014. JMLR.org.

[15] Colin Campbell, Nello Cristianini, Alex Smola, and others. Query learning with large margin classifiers. In *ICML*, pages 111–118, 2000.

[16] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. Expected Reciprocal Rank for Graded Relevance. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 621–630, New York, NY, USA, 2009. ACM.

[17] Alessandro Checco and Gianluca Demartini. Pairwise, Magnitude, or Stars: What's the Best Way for Crowds to Rate? *arXiv:1609.00683 [cs]*, September 2016. arXiv: 1609.00683.

[18] Xi Chen, Paul N. Bennett, Kevyn Collins-Thompson, and Eric Horvitz. Pairwise ranking aggregation in a crowdsourced setting. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 193–202. ACM, 2013.

[19] Xi Chen, Sivakanth Gopi, Jieming Mao, and Jon Schneider. Competitive analysis of the top-K ranking problem. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, Proceedings, pages 1245–1264. Society for Industrial and Applied Mathematics, January 2017.

[20] Yuxin Chen and Changho Suh. Spectral MLE: Top-K Rank Aggregation from Pairwise Comparisons. *arXiv:1504.07218 [cs, math, stat]*, April 2015. arXiv: 1504.07218.

[21] Chinacrowds. http://www.chinacrowds.com.

[22] Wei Chu and Zoubin Ghahramani. Extensions of gaussian processes for ranking: semisupervised and active learning. In *Proceedings of the NIPS 2005 Workshop on Learning to Rank*, pages 29–34. MIT, 2005.

[23] Eleonora Ciceri, Piero Fraternali, Davide Martinenghi, and Marco Tagliasacchi. Crowdsourcing for top-K query processing over uncertain data. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 1452–1453, May 2016.

[24] David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 1996.

[25] Gordon V. Cormack, Charles L A Clarke, and Stefan Buettcher. Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 758–759, New York, NY, USA, 2009. ACM.

[26] Joana Costa, Catarina Silva, Mário Antunes, and Bernardete Ribeiro. On using crowdsourcing and active learning to improve classification performance. In *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, pages 469–474. IEEE, 2011.

[27] Koby Crammer, Michael Kearns, and Jennifer Wortman. Learning from multiple sources. *The Journal of Machine Learning Research*, 9:1757–1774, 2008.

[28] Donald Davidson. Experimental Tests of a Stochastic Decision Theory (1959). In Jacob Marschak, editor, *Economic Information, Decision, and Prediction: Selected Essays: Volume I Part I Economics of Decision*, Theory and Decision Library, pages 133–171. Springer Netherlands, Dordrecht, 1974.

[29] Susan B Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Using the crowd for top-k and group-by queries. In *Proceedings of the 16th International Conference on Database Theory*, pages 225–236. ACM, 2013.

[30] Susan B. Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Using the Crowd for Top-k and Group-by Queries. In *Proceedings of the 16th International Conference on Database Theory*, ICDT '13, pages 225–236, New York, NY, USA, 2013. ACM.

[31] Luca de Alfaro, Vassilis Polychronopoulos, and Neoklis Polyzotis. Efficient Techniques for Crowdsourced Top-k Lists. In *Fourth AAAI Conference on Human Computation and Crowdsourcing*, September 2016.

[32] Anhai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. Crowdsourcing Systems on the World-Wide Web. *Commun. ACM*, 54(4):86–96, April 2011.

[33] Pinar Donmez and Jaime G. Carbonell. Optimizing Estimated Loss Reduction for Active Sampling in Rank Learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 248–255, New York, NY, USA, 2008. ACM.

[34] Pinar Donmez and Jaime G. Carbonell. Proactive learning: cost-sensitive active learning with multiple imperfect oracles. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 619–628. ACM, 2008.

[35] Pinar Donmez and Jaime G. Carbonell. Active sampling for rank learning via optimizing the area under the ROC curve. In *Advances in Information Retrieval*, pages 78–89. Springer, 2009.

[36] Figure Eight. https://www.figure-eight.com.

[37] Arpad E. Elo. *The rating of chess players, past and present*. Arco Pub., 1978.

[38] Brian Eriksson. Learning to Top-K Search using Pairwise Comparisons. In *Artificial Intelligence and Statistics*, pages 265–273, April 2013.

[39] Meng Fang, Jie Yin, and Dacheng Tao. Active Learning for Crowdsourcing Using Knowledge Transfer. In *AAAI*, pages 1809–1815, 2014.

[40] Lester Randolph Ford. Solution of a Ranking Problem from Binary Comparisons. *The American Mathematical Monthly*, 64(8P2):28–33, October 1957.

[41] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *The Journal of machine learning research*, 4:933–969, 2003.

[42] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Machine learning*, 28(2-3):133–168, 1997.

[43] David F. Gleich and Lek-heng Lim. Rank aggregation via nuclear norm minimization. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 60–68. ACM, 2011.

[44] Mark E. Glickman. Parameter Estimation in Large Dynamic Paired Comparison Experiments. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 48(3):377–394, January 1999.

[45] Samuel D. Gosling, Sei Jin Ko, Thomas Mannarelli, and Margaret E. Morris. A room with a cue: Personality judgments based on offices and bedrooms. *Journal of Personality and Social Psychology*, 82(3):379–398, 2002. Place: US Publisher: American Psychological Association.

[46] Aiden P. Gregg, Beate Seibt, and Mahzarin R. Banaji. Easier done than undone: asymmetry in the malleability of implicit preferences. *Journal of Personality and Social Psychology*, 90(1):1–20, January 2006.

[47] Gul Gunaydin, Emre Selcuk, and Vivian Zayas. Impressions Based on a Portrait Predict, 1-Month Later, Impressions Following a Live Interaction. *Social Psychological and Personality Science*, 8(1):36–44, January 2017. Publisher: SAGE Publications Inc.

[48] Stephen Guo, Aditya Parameswaran, and Hector Garcia-Molina. So Who Won?: Dynamic Max Discovery with the Crowd. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 385–396, New York, NY, USA, 2012. ACM.

[49] Bruce Hajek, Sewoong Oh, and Jiaming Xu. Minimax-optimal Inference from Partial Rankings. In *Advances in Neural Information Processing Systems 27*, pages 1475–1483. Curran Associates, Inc., 2014.

[50] Reinhard Heckel, Nihar B. Shah, Kannan Ramchandran, and Martin J. Wainwright. Active ranking from pairwise comparisons and when parametric assumptions do not help. *The Annals of Statistics*, 47(6):3099–3126, December 2019.

[51] Ralf Herbrich, Tom Minka, and Thore Graepel. TrueSkill™ : A Bayesian Skill Rating System. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 569–576. MIT Press, 2007.

[52] John Joseph Horton and Lydia B. Chilton. The Labor Economics of Paid Crowdsourcing. In *Proceedings of the 11th ACM Conference on Electronic Commerce*, EC '10, pages 209–218, New York, NY, USA, 2010. ACM.

[53] Tzu-Kuo Huang, Ruby C. Weng, and Chih-Jen Lin. Generalized Bradley-Terry models and multi-class probability estimates. *The Journal of Machine Learning Research*, 7:85–115, 2006.

[54] Bernardo A. Huberman, Peter L. T. Pirolli, James E. Pitkow, and Rajan M. Lukose. Strong Regularities in World Wide Web Surfing. *Science*, 280(5360):95–97, April 1998.

[55] David R. Hunter. MM algorithms for generalized Bradley-Terry models. *The Annals of Statistics*, 32(1):384–406, February 2004.

[56] Kevin Jamieson, Matthew Malloy, Robert Nowak, and Sébastien Bubeck. An Optimal Exploration Algorithm for Multi-Armed Bandits. In *Conference on Learning Theory*, pages 423–439, May 2014.

[57] Kevin G. Jamieson and Robert D. Nowak. Active Ranking Using Pairwise Comparisons. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2240–2248, USA, 2011. Curran Associates Inc.

[58] Minje Jang, Sunghyun Kim, Changho Suh, and Sewoong Oh. Top-K Ranking from Pairwise Comparisons: When Spectral Ranking is Optimal. *arXiv e-prints*, 1603:arXiv:1603.04153, March 2016.

[59] Xiaoye Jiang, Lek-Heng Lim, Yuan Yao, and Yinyu Ye. Statistical ranking and combinatorial Hodge theory. *Mathematical Programming*, 1(127):203–244, 2011.

[60] Saikishore Kalloori, Francesco Ricci, and Marko Tkalcic. Pairwise Preferences Based Matrix Factorization and Nearest Neighbor Recommendation Techniques. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 143–146, Boston, Massachusetts, USA, September 2016. Association for Computing Machinery.

[61] Ece Kamar, Severin Hacker, and Eric Horvitz. Combining human and machine intelligence in large-scale crowdsourcing. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 467–474. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[62] David R. Karger, Sewoong Oh, and Devavrat Shah. Iterative learning for reliable crowdsourcing systems. In *Advances in neural information processing systems*, pages 1953–1961, 2011.

[63] M. G. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93, 1938. Publisher: [Oxford University Press, Biometrika Trust].

[64] Anton Kühberger and Patricia Gradl. Choice, Rating, and Ranking: Framing Effects with Different Response Modes. *Journal of Behavioral Decision Making*, 26(2):109–117, 2013. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/bdm.764.

[65] Florian Laws, Christian Scheible, and Hinrich Schütze. Active learning with amazon mechanical turk. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1546–1556. Association for Computational Linguistics, 2011.

[66] Bo Long, Olivier Chapelle, Ya Zhang, Yi Chang, Zhaohui Zheng, and Belle Tseng. Active learning for ranking through expected loss optimization. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 267–274. ACM, 2010.

[67] Tyler Lu and Craig Boutilier. Learning Mallows Models with Pairwise Preferences. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 145–152, USA, 2011. Omnipress.

[68] R. Duncan Luce. *Individual choice behavior*. Individual choice behavior. John Wiley, Oxford, England, 1959.

[69] R. Duncan Luce. *Individual choice behavior: A theoretical analysis*. Individual choice behavior: A theoretical analysis. Dover Publications, Mineola, NY, US, 2005. Pages: xii, 153.

[70] Yarun Luon, Christina Aperjis, and Bernardo A. Huberman. Rankr: A Mobile System for Crowdsourcing Opinions. In Joy Ying Zhang, Jarek Wilkiewicz, and Ani Nahapetian, editors, *Mobile Computing, Applications, and Services*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 20–31, Berlin, Heidelberg, 2012. Springer.

[71] David J. C. MacKay. Information-Based Objective Functions for Active Data Selection. *Neural Computation*, 4(4):590–604, July 1992.

[72] Colin Lingwood Mallows. Non-Null Ranking Models. I. *Biometrika*, 44(1/2):114–130, 1957.

[73] Thomas Mann and Melissa Ferguson. Can We Undo Our First Impressions? The Role of Reinterpretation in Reversing Implicit Evaluations. *Journal of Personality and Social Psychology*, 108(6):823–849, June 2015.

[74] Ke Mao, Licia Capra, Mark Harman, and Yue Jia. A survey of the use of crowdsourcing in software engineering. *Journal of Systems and Software*, 126:57–84, April 2017.

[75] Lucas Maystre and Matthias Grossglauser. Fast and Accurate Inference of Plackett–Luce Models. In *Advances in Neural Information Processing Systems 28*, pages 172–180. Curran Associates, Inc., 2015.

[76] Prem Melville and Raymond J. Mooney. Diverse ensembles for active learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 74. ACM, 2004.

[77] Sahand Negahban, Sewoong Oh, and Devavrat Shah. Iterative Ranking from Pair-wise Comparisons. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'12, pages 2474–2482, USA, 2012. Curran Associates Inc.

[78] Sahand Negahban, Sewoong Oh, and Devavrat Shah. Rank Centrality: Ranking from Pairwise Comparisons. *Operations Research*, 65(1):266–287, 2017.

[79] Ulrich Paquet, Jurgen Van Gael, David Stern, Gjergji Kasneci, Ralf Herbrich, and Thore Graepel. Vuvuzelas & Active Learning for Online Classification. In *NIPS Workshop on Comp. Social Science and the Wisdom of Crowds*, 2010.

[80] Robin L. Plackett. The analysis of permutations. *Applied Statistics*, pages 193–202, 1975.

[81] Marion K. Poetz and Martin Schreier. The Value of Crowdsourcing: Can Users Really Compete with Professionals in Generating New Product Ideas? *Journal of Product Innovation Management*, 29(2):245–256, March 2012.

[82] Vassilis Polychronopoulos, Luca De Alfaro, James Davis, Hector Garcia-Molina, and Neoklis Polyzotis. Human-powered top-k lists. In *WebDB*, pages 25–30, 2013.

[83] Tao Qin, Xiubo Geng, and Tie-Yan Liu. A new probabilistic model for rank aggregation. In *Advances in neural information processing systems*, pages 1948–1956, 2010.

[84] Filip Radlinski and Thorsten Joachims. Active exploration for learning rankings from clickthrough data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 570–579. ACM, 2007.

[85] Filip Radlinski, Madhu Kurup, and Thorsten Joachims. How Does Clickthrough Data Reflect Retrieval Quality? In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 43–52, New York, NY, USA, 2008. ACM.

[86] Arun Rajkumar and Shivani Agarwal. A Statistical Convergence Perspective of Algorithms for Rank Aggregation from Pairwise Data. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages I–118–I–126, Beijing, China, 2014. JMLR.org.

[87] Arun Rajkumar and Shivani Agarwal. When can we rank well from comparisons of O(n\log(n)) non-actively chosen pairs? In *Conference on Learning Theory*, pages 1376–1401, June 2016.

[88] Nicholas Roy and Andrew McCallum. Toward optimal active learning through monte carlo estimation of error reduction. *ICML, Williamstown*, pages 441–448, 2001.

[89] Robert J. Rydell and Allen R. McConnell. Understanding implicit and explicit attitude change: A systems of reasoning analysis. *Journal of Personality and Social Psychology*, 91(6):995–1008, 2006. Place: US Publisher: American Psychological Association.

[90] Nihar B. Shah and Martin J. Wainwright. Simple, Robust and Optimal Ranking from Pairwise Comparisons. *Journal of Machine Learning Research*, 18(199):1–38, 2018.

[91] Padhraic Smyth, Usama Fayyad, Michael Burl, Pietro Perona, and Pierre Baldi. Inferring Ground Truth from Subjective Labelling of Venus Images. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 1085–1092. The MIT Press, Cambridge, MA, 1995.

[92] Hossein Azari Soufiani, William Chen, David C Parkes, and Lirong Xia. Generalized Method-of-Moments for Rank Aggregation. In *Advances in Neural Information Processing Systems 26*, pages 2706–2714. Curran Associates, Inc., 2013.

[93] Hossein Azari Soufiani, David Parkes, and Lirong Xia. Computing Parametric Ranking Models via Rank-Breaking. In *International Conference on Machine Learning*, pages 360–368, January 2014.

[94] Changho Suh, Vincent Y. F. Tan, and Renbo Zhao. Adversarial Top- K Ranking. *IEEE Transactions on Information Theory*, 63(4):2201–2225, April 2017.

[95] Louis L. Thurstone. The method of paired comparisons for social values. *The Journal of Abnormal and Social Psychology*, 21(4):384, 1927.

[96] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *The Journal of Machine Learning Research*, 2:45–66, 2002.

[97] Amazon Mechanical Turk. https://www.mturk.com.

[98] Petros Venetis, Hector Garcia-Molina, Kerui Huang, and Neoklis Polyzotis. Max algorithms in crowdsourcing environments. In *Proceedings of the 21st international conference on World Wide Web*, pages 989–998. ACM, 2012.

[99] Petros Venetis, Hector Garcia-Molina, Kerui Huang, and Neoklis Polyzotis. Max Algorithms in Crowdsourcing Environments. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 989–998, New York, NY, USA, 2012. ACM.

[100] Jing Wang, Panagiotis G. Ipeirotis, and Foster Provost. Managing crowdsourcing workers. In *The 2011 winter conference on business intelligence*, pages 10–12, 2011.

[101] Fabian Wauthier, Michael Jordan, and Nebojsa Jojic. Efficient Ranking from Pairwise Comparisons. In *PMLR*, pages 109–117, February 2013.

[102] Qianqian Xu, Qingming Huang, and Yuan Yao. Online crowdsourcing subjective image quality assessment. In *Proceedings of the 20th ACM international conference on Multimedia*, MM '12, pages 359–368, Nara, Japan, October 2012. Association for Computing Machinery.

[103] Yan Yan, Glenn M. Fung, Rómer Rosales, and Jennifer G. Dy. Active learning from crowds. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1161–1168, 2011.

[104] Ziying Yang, Alistair Moffat, and Andrew Turpin. Pairwise Crowd Judgments: Preference, Absolute, and Ratio. In *Proceedings of the 23rd Australasian Document Computing Symposium*, ADCS '18, pages 1–8, Dunedin, New Zealand, December 2018. Association for Computing Machinery.

[105] Hwanjo Yu. SVM selective sampling for ranking with application to data retrieval. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 354–363. ACM, 2005.

[106] Xiaohang Zhang, Guoliang Li, and Jianhua Feng. Crowdsourced Top-k Algorithms: An Experimental Evaluation. *Proceedings of the VLDB Endowment*, 9(8):612–623, April 2016.

[107] Yuxiang Zhao and Qinghua Zhu. Evaluation on crowdsourcing research: Current status and future direction. *Information Systems Frontiers*, 16(3):417–434, July 2014.

[108] Yuan Zhou, Xi Chen, and Jian Li. Optimal PAC Multiple Arm Identification with Applications to Crowdsourcing. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–217–II–225, Beijing, China, 2014. JMLR.org.