AN EVALUATION OF A STATISTICAL FRAMEWORK AND ALGORITHMS FOR ADAPTIVE AGGREGATION

By

Brandeis Hill

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved by the
Examining Committee:

_____
Sibel Adalı, Thesis Advisor

_____
Malik Magdon-Ismail, Co-Thesis Advisor

_____
Mark Goldberg, Member

_____
Mukkai Krishnamoorthy, Member

_____
Frank Spear, Member

Rensselaer Polytechnic Institute
Troy, New York

April 2007
(For Graduation May 2007)

AN EVALUATION OF A STATISTICAL FRAMEWORK AND ALGORITHMS FOR ADAPTIVE AGGREGATION

By

Brandeis Hill

An Abstract of a Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

The original of the complete thesis is on file
in the Rensselaer Polytechnic Institute Library

Examining Committee:

Sibel Adalı, Thesis Advisor
Malik Magdon-Ismail, Co-Thesis Advisor
Mark Goldberg, Member
Mukkai Krishnamoorthy, Member
Frank Spear, Member

Rensselaer Polytechnic Institute
Troy, New York

April 2007
(For Graduation May 2007)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

During my graduate career at Rensselaer Polytechnic Institute, many people have directly and indirectly influenced my research and personal development. With their guidance, it would be impossible to complete this dissertation.

I would like to first acknowledge my advisor, Dr. Sibel Adalı for providing the opportunity to conduct research under her guidance and supervision. I would like to express my deepest gratitude for her patience and feedback. Her multifaceted contributions to the research is much appreciated.

I would like to also acknowledge my co-advisor, Dr. Malik Magdon-Ismail for his guidance and suggestions to this research. He provided the mathematical fundamentals necessary for the statistical framework. I am very grateful for his technical expertise.

A special thank you is extended to my doctoral committee, Dr. Mark Goldberg, Dr. Mukkai Krishnamoorthy and Dr. Frank Spear, for their comments and suggestions.

Through the years, the following people have assisted in my personal development: Dr. John Alan Bivens II, Kim Bivens, Rasheda Huggins, Lesley Mbogo, Dr. Bettina Schimanski, Dr. Bouchra Bouqata, Dr. Kaoutar El Maghraoui, Benjarath Phoophakdee, Dr. Chi-nan Chiang, Dr. Ken Durgans, Dr. Juan Gilbert, Dr. Raquel Hill, Dr. Damon Woodard, Dr. Jamika Burge, Karen Hare, Jamila Greene, Chris Coonrad, Shannon Carrothers, Pam Paslow, Jacky Carley and countless others.

Lastly, I would like to thank my parents, Carlton and Magnolia Hill, for instilling the value of work and perseverance. Your understanding and support was never-ending. This degree is as much yours as mine. I would also like to thank Gemez Marshall, my husband, for his daily encouragement and assistance.

Thank You All.

Dr. Brandeis Hill Marshall

ABSTRACT

The rank aggregation problem aims to combine several ranked lists to obtain a final "consensus" ranked list that gives better results than any one of the individual ranked lists. With the emergence of the World Wide Web, the meta-search community has studied the rank aggregation problem in order to aggregate search results from multiple search engines and increase the coverage of the Web by accessing more information. The existing aggregation methods address the problem of capturing the user feedback accurately while minimizing the impact of outliers or spam. However, the prior work does not provide guidelines about *when* to use the aggregation method in *which* problem setting as well as *how* to dynamically select an aggregation method for that problem setting. In this thesis, we address these shortcomings in the prior work. Since rank aggregation also appears in fields other than the Web, we need a more general platform to examine an optimal ranking. We define two factors that contribute to the performance of rank aggregation methods including noise, such as spam, and misinformation, such as trustworthiness.

In meta-search, *relevance* of an object is difficult to decide since evaluation depends on subjective expert judgments. To address this concern, we propose a flexible statistical framework to model the possible different relationships between the rankers, such as search engines, and the ground truth. Our model contains a ground truth ranker, which corresponds to the correct ordering of objects, and the input rankers that serve as approximations of the ground truth ranker. We also develop several aggregation methods that capture different aspects of the rank information including precision optimal, iterative best flip and three algorithms that are approximations to the minimum feedback arc set problem. We show that there is a trade off between information and robustness when selecting the best aggregation method and that none of the well-known rankers perform well uniformly in all different noise and misinformation conditions. We develop classification methods and bayesian techniques to dynamically select the optimal aggregator based on classified noise and misinformation.

CHAPTER 1
INTRODUCTION

The rank aggregation problem aims to combine several ranked lists to obtain a final "consensus" ranked list that gives better results than any one of the individual ranked lists. The existing approaches in finding a solution to this problem include devising algorithms to exploit different aspects of the ranked lists and using human evaluators or information from prior requests to test the accuracy of these algorithms. The existing aggregation methods [53, 56, 60, 86, 97, 104] address the problem of capturing the user feedback accurately, capturing the most useful information in the rankings and suppressing outliers or spam. However, the prior work only provides an ad-hoc definition of spam and does not provide any guidelines as to which algorithm to use when. Furthermore, most of the algorithms developed in the prior work are static and do not address the problem of automatically adopting the aggregation methods to the specific problem setting. While the rank aggregation problem has been studied widely in the meta-search community, the problem arises in many different fields where objects are ranked. Therefore, a more general definition of optimal ranking in the presence of noise and also misinformation is needed. This is the problem studied in this thesis. We define two factors related to the performance of rank aggregation methods, namely noise (including spam) and misinformation or trustworthiness. We then develop optimal aggregation methods as a function of these factors. In this chapter, we formalize the problem and discuss the motivation and contributions of this research.

1.1 Problem Setting and Scope

The hunt or search for relevant information is not new; however, the quality and quantity of information has changed with the emergence of the World Wide Web. In the information search domain, each ranked list (ranker) refers to a search engine result of a specific query and the aggregator refers to the results of a meta-search engine for that query. The aggregator only has access to the ranking of objects.

Even though the ranks may be obtained by a scoring system, this information is not available to the aggregator. It is possible that the aggregator may have additional information about each object, but we do not take this information into account. In this thesis, we assume this information can be integrated in many different ways to the optimal ranking algorithm.

One of the main stumbling blocks in search is deciding which information is *relevant*. Relevance is usually considered a subjective notion. Hence, evaluation methods tend to use real data and expert judgment to determine relevance. But, this makes it hard to test methods on a large number of datasets in order to arrive at significant, repeatable results.

In information retrieval, a number of factors are taken into account to determine relevance such as the number of times and the location of the occurrence of each keyword. The correct way to incorporate these factors require understanding the data collection and the importance of each factor with respect to this collection. Another challenge of searching the Web is the open nature of the Web. Documents are created and maintained by many different individuals with different standards and styles. There is generally no way to certify the correctness of information provided on a Web site. In fact, the Web contains malicious agents that purposefully misrepresent information. Furthermore, the data are updated in different intervals in each site, hence the data contained at a Web site can be stale and therefore incorrect. To address some of the problems posed by such an environment, search engines take into account a large number of factors to determine relevance in addition to the content of a Web page. Examples of such factors are frequency of updates to a Web page, number of incoming hyperlinks to a Web page and the text contained in the hyperlinks.

Our focus is to find the best way to incorporate the rank information from different sources. We investigate the problem of aggregating different rankers to provide a single ranking that best represents the true ranking of objects. As experts may evaluate their ranking with respect to various alternatives to solving a problem, we may want to find the best ranking based on how much we trust or distrust certain experts on various topics. We need to examine how we can factor these measures

into aggregation. To address our problem, we first examine the properties of rankers.

- Let's assume there exists a ground truth of what the correct ranking should be. We then have to find how each ranker's rankings relate to this ground truth. How much information and how much noise is contained in the rankings? Do rankers make similar mistakes?

- Given these different criteria for characterizing rankers, what is the performance of different rank aggregation methods? Are there different classes of rank aggregation methods that perform well for different cases or do they perform all the same?

- If different aggregators perform well under different cases, how can this information be leveraged to formulate an adaptive aggregation algorithm?

1.2 Motivation

Search is an inherently complex problem, requiring the information system to assess the meaning of a user query, match it to a document's content, such as a Web page, and determine the relevancy of the document to the user query. To determine the relevance of a document to a query consisting of keywords, the system has to first locate the documents that contain the given keywords assuming such documents exist. Note that in the absence of an exact match for the given keywords, the query can be expanded to keywords with similar meaning. However, due to the redundancy of information available in the World Wide Web, this is a rare occurrence. Secondly, as the relevant documents are found, they have to be ordered in terms of their match quality to the given keyword.

Even though the methods used by search engines proved to be quite effective in eliminating irrelevant results, they are still open to manipulation. For example, a document is regarded as spam if it does not contain relevant information but is highly ranked. This may be due to one of many techniques such as text weaving and link farms employed to fool the algorithm for computing these factors. Even though search engines are a good filter for spam, aggregating results from multiple search engines is usually a more effective method for suppressing spam. It is harder

to manipulate multiple distinct algorithms than a single one. This is one of the main motivations for using meta-search engines, which aggregate search results from multiple search engines to produce a new ranking. Another advantage of meta-search engines is that they increase the coverage of the Web by allowing access to a larger database consisting of all search engines it combines.

Aggregating many independent rankers generally improves the ranking quality. However, this is only true if the rankers are simply noised versions of the ground truth. It is possible that the rankers may also introduce misinformation to their rankings for many reasons. For example, they may want to hide their ranking method so they may choose to purposefully alter their ranking for different queries. Similarly, malicious rankers may want to manipulate the final ranking of objects by altering their ranking. In this case, it is not desirable to aggregate all rank information, but to disregard some. Different aggregation methods may disregard different types of information and hence the most appropriate aggregation method should be chosen for a given scenario.

Rank aggregation is not only relevant to information search but in any field where objects are ranked by different individuals or parties and a final ranking of objects is needed. The work introduced in this thesis equally applies to these as long as the ranking method is similar to the one studied here.

1.3 Contributions and Methodology

Given the above issues, the main objective is to examine the rank aggregation problem from a different perspective, where the quality of the input rankers (not simply the individualized results) is an important factor in aggregation but also customization of search results can be achieved through effective aggregation. In this dissertation, we provide the following contributions:

- develop a flexible statistical framework to model the possible different relationships between the rankers and the ground truth;
- develop several aggregation algorithms that capture different aspects of the rank information including precision optimal, iterative best flip and three al-

gorithms that are approximations to the minimum feedback arc set problem;

- show a trade off between information and robustness when selecting the best aggregator and that none of the well-known rankers perform uniformly well in all different conditions; and

- how to identify misinformation of the input rankers and noise in which to effectively customize the search results.

To fully understand the properties of the aggregation methods, we propose a statistical model for aggregation which comprises of properties that are associated with each ranker. Our model contains a ground truth ranker, which corresponds to the correct ordering of objects, and the input rankers that serve as approximations of the ground truth ranker. We assume the ranking of objects is based on a number of factors and the rankers use a similar method to estimate the ground truth ranking. However, they may make mistakes or give different weights to the factors. This framework allows us to generate datasets with many different settings. Each setting allows for rankers to make different types of errors. Using this framework, we evaluate existing aggregation algorithms and introduce others, which are based on algorithms that appear in the literature such as local Kemenization [53], Kernighan and Lin graph partitioning [78] and bi-connected components.

We perform an extensive experimental evaluation of synthetic datasets to understand the quality of rankers. The experimental evaluation is useful for a theoretical study of the impact of the properties of rankers on the performance of the aggregator. Our findings show that the choice of an optimal ranker is a balance of robustness and information. An aggregator that disregards a lot of information is robust to noise. We consider such aggregators as simple. However, as a simple aggregator ignores a lot of information, it does not perform as well as a more complex one in low noise cases. Another issue is the possible asymmetry between rankers, which requires rankers that are robust to outlier ranks such as the median ranker. The amount of robustness needed is a function of noise. As the noise becomes very high, the information available in each ranker is reduced considerably. As a result, more complex rankers become competitive again.

Given this complex equation involving the choice of the best aggregator, the next question is how to determine the best aggregator for a specific scenario. To address this problem, two metrics are introduced to measure noise and misinformation. We show that these metrics can be used to correctly determine the specific noise and misinformation scenario using the statistical framework. We show that by dynamically determining the specific scenario and using the best ranker for this scenario, we are able to improve over the performance of any of the known rank aggregation methods. Our methods can be easily extended to real life settings where the ground truth is not known. Our methods require very little user input for the training phase and can be obtained implicitly by observing user behavior in a non-intrusive way.

We validate our results using real data in two different settings. First, we use the TREC [69] data collection, which provides a large number of rankers that we randomly sample. This data collection has relevance labels for each object. The relevance judgments simply defines objects that are a possible match from those objects that are not a match. However, as the relevance judgments are not ranked, we can only evaluate them using the precision performance measure. The second data set is collected from the Web by us using a large number of search engines. In this case, we conduct a small user study to determine the relevance ordering of select list of objects. We use this information to order the aggregation methods. In both settings, we estimate the noise and misinformation and validate our ordering of rankers.

1.4 Dissertation Outline

The rest of the thesis is organized as follows. Chapter 2 reviews the previous literature in searching, aggregation methods and user preferences. Chapter 3 outlines the appropriate background and terminology we use for the remainder of this work. Chapters 4 and 5 describe rank-based and approximation aggregation algorithms. In Chapter 6, we present the statistical framework for aggregation. An evaluation of the statistical framework, which compares the algorithms presented in Chapters 4 and 5, is conducted in Chapter 7. In Chapter 8, we develop methods to dynamically adjust the rank aggregation method to the properties of the rankers.

Lastly, we summarize and conclude the dissertation.

CHAPTER 2
RELATED WORK

In this section, we review a selected collection of prior literature relevant to the study of rank aggregation in information retrieval and database literature. A great deal of work on merging ranked lists from different sources concentrates on the problem of finding an aggregation method that performs well in a specific scenario. For example, in information retrieval, the emphasis is on finding the best way to incorporate information about different features of text into the final ranking [92, 108]. The results are usually evaluated with respect to typical text collections and their specific properties. Similarly, rank aggregation methods in meta-searching for the Web concentrate on methods to capture the input rankers accurately or suppress noise due to possible spam [53].

These aggregation methods typically treat all the rankers uniformly and do not try to capture the differences among them. There is no work to our knowledge that evaluates different ranking methods based on how well they behave with respect to the characteristics of their inputs. Our work tries to capture the relationship of input rankers to a possible ground truth. This approach allows the user to choose between different and contradictory objectives of capturing the input rankers closely or disregarding outliers. We also show that outliers exist in two dimensions of noise and misinformation with different implications on performance. There is no known work that approaches this problem.

In contrast with the existing work, we estimate the possible asymmetry or difference of opinion between rankers explicitly and use this in aggregation. There is some work on learning user preferences with respect to the input rankers [3, 75]. These methods typically require extensive training data for each user that may not be available in many cases. In contrast, we develop aggregation methods that do not require any user specific training data. To our knowledge, there is no work that attempts to estimate and use the amount of noise and misinformation in the rankers in rank aggregation.

In the following sections, we give the detailed description of the related work in information retrieval, web search and personalization of rank aggregation.

2.1 Overview of Web Search

Ranking and rank aggregation are common methods employed in Web search. The information accessible on the Web is normally considered as a set of documents, each document containing a collection of text and other media elements that are linked to it. There are two main types of web documents: static and dynamic. A static web page is structured in a narrative style with possibly additional navigational elements. Generally, a link in a page carries an anchor text, which has relevant information about the page. A dynamic web page contains information that is usually extracted from a database upon request. Many dynamic web pages are part of the "Deep Web" [19], which is not indexable by search engines.

Web documents are described using the textual contents as well as its location, such as title, author, meta tags or body. The importance of a keyword in a Web document can be determined by the stylistic elements including its font size and type. Any dynamic properties of the Web site, update frequency, amount of time it has been online or number of hyperlinks pointing to the page from the outside world, are used to judge the quality of the Web page. Generally, the content and the importance of a Web page are considered to understand its relevancy when answering a user query against a search engine.

The structure of the Web is generally considered to be a graph $G = (V, E)$ where the Web documents are vertices (V) and the hyperlinks between two Web documents are edges (E). This directed graph may have parallel edges between two nodes in contrast with usual graphs. The structure of the Web is a lot richer in practice, each document contains text as well as many other types of objects. The part of the Web that is indexable by search engines corresponds to a subset of this structure that can be referred to as a collection of documents with links between them.

Since Web documents can be constructed by any person for any reason, the size of this graph G is very large and growing constantly. In order to facilitate the

discovery of Web information, users can search the Web via *search engines* and *meta-search engines* in finding the desired information. These engines normally allow a user to enter a search query, comprising of at least one keyword, and locate the Web documents that best match the search query and have the best quality. Search engines and meta-search engines concentrate not only on relevance of a document returned for the query, but also the underlying quality of the information that is available.

With the possibility of multiple interpretations of keywords, search engines have difficulty distinguishing which definition is desired and which documents to return to the user. Some research on engines have incorporated clustering techniques to separate and label the various meanings of the query. SnakeT [58] is a hierarchical clustering search engine that organizes the query results such that the information is easier to find for the user. The classification of a Web document is done through web snippets. Labels or folders are constructed for the clustered Web documents based on the common phrases observed. Their algorithm is able to construct a compact and balanced hierarchy of folders that may be overlapping since Web documents can be applicable in a number of settings [119]. Chakrabarti et al. [33] categorizes the query results in a database by constructing a hierarchy of labels that the user can choose to explore or ignore. The labels are organized automatically based on the contents of the tuples in the answer set. The hierarchy is designed to be balanced in order to minimize the information overload contained within each label. The authors develop an extensive experimental methodology to examine the benefits of categorization after defining the cost model and workload characteristics estimations.

The work in clustering search engine results has led to the emergence of question-answering systems in which there exists a correct solution. The Tritus system [2] is developed to transform questions to query phrases and learns how to answer the questions.

2.1.1 Information Retrieval

The natural question in web search is how to retrieve documents matching the keywords. This problem has many solutions developed into the research field of

information retrieval. However, it also offers many new challenges specific to web searching. Information retrieval (IR) is a mature research area that aims to index and search a large corpora of text, image and other media. It has foundations in the late 1960's with the work of Salton [108] for determining the relevance of documents to keyword queries. IR systems aim to rank documents in a text collection based on their relevance to a query. The models used range from those that view documents as bag of terms to those that consider the narrative style based on natural language processing. In more recent prior research has extended to the search of multimedia documents and other forms of media such as images and video.

One of the widely used models for information retrieval is the **vector space model** [92]. This model represents documents and queries as vectors in high-dimensional space, in which each dimension denotes a word in the collection. The answers to the query, in sorted order, are determined by a distance function, e.g. cosine, influenced by the frequency count for the keywords in those documents Not all words in a document are represented in the vector space model. A stop list of English grammatical words such as *and* and *the* can be ignored since they have high occurrences and does not have semantics contributing to the retrieval.

Relevance Feedback. The amount of information available in information retrieval systems causes users to have difficulty discovering the data most relevant to the query. In some cases, users know exactly what information they are looking for but the query does not represent their interests well. The user can judge the returned results as relevant or irrelevant. Then, the system alters its ranking strategy based on the user input, which better aligns with the user's judgment. Feedback can be given explicitly by the user, however it may be very difficult to obtain a large amount of useful feedback, or implicitly through observing user behavior to infer which documents are relevant to the user. Relevance feedback learns how to answer these queries uncovering the appropriate results through the training data and performing experiments on the testing data.

The original relevance feedback algorithm initializes the initial query with a set of terms and relevance judgments. Allan [5] investigates relevance feedback

protocols under the assumption that there is a continuous arrival of documents, queries are long-lived ,e.g. repeatedly executed, and the relevance of a document is binary (either relevant or irrelevant). The author modifies the initial relevance feedback algorithm by incrementally feeding back only a fraction of the training set to the system. Each keyword that appears in the relevant documents are ordered with respect to the number of times the keyword occurs in the relevant documents. The re-ranking of the terms is computed using the Rocchio formula, which is $w_{query} + 2 * w_{rel} - \frac{1}{2} * w_{nonrel}$ where w_X is the weight of the term for the query, relevant or irrelevant documents. The Rocchio formula is used in lieu of other better performing algorithms since it is simple and relatively effective.

Chakrabarti [34] introduces a method to improve relevance feedback for Web documents by means of a measure called the Jaccard coefficient. The objective of the Jaccard coefficient is to remove the duplicate documents in the query results. The degree of overlap between two documents is computed as a ratio of the number of shared keywords to the number of keywords processed. If the overlap is large, then the documents are similar to each other and only one document needs to be returned for the relevance feedback.

User feedback is also a key component for Beitzel and Lewis [17] who develop a rule-based automatic classifier using preferences to categorized queries in order to more efficiently and effectively retrieve results to users. In addition to user feedback, exact matching is also applied to return more accurate results that can not be retrieved from using either method individually. The preferences are head-tail pairs of keywords for disambiguation and easier semantic interpretation even though query lengths are a short 2 or 3 keywords. Another approach to learning how to rank objects is done by Bifet et al. [24], which conducts a statistical analysis using logic regression, support vector machines and binary classification. The properties of search engine results are collected in which the method of learning results are compared to Google. Their results showed that the binary classification approach categorizes the results correctly for both the training and all experiments; however, the dataset was small and needs to be tested more extensively.

Inclusion of Personalization. The personalization of queries, and their answers, is a way to discover and remember the results that are useful to the user at a specific time. Personalization allows users to restrict their search space and direct the retrieval. Instead of processing the query and allowing the user to further search for the results aligning with her interests, the user's preferences are considered as well so the results are hopefully fewer and more accurate.

Personalization can be achieved through leveraging the implicit user preferences and re-rank the query results within a relevance feedback system [113]. A study of search algorithms of a user's past actions is done to personalize the current web search. The targeted queries are those with many interpretations and prior research relies on well-formed user profiles and well-defined representation of knowledge. The authors investigate the corpus representation, user representation and document or query representation. The corpus representation estimates the total number of documents in the collection and number of documents that include a specific keyword. The user representation estimates the number of documents which relevance feedback has been provided and the number of these documents that contain the specific keyword. The document or query representation is what keywords are summed over. The performance evaluation used the measures defined in [74], called discounted cumulative gain. The results conclude that the more data used to represent the user, the better the answers. The richer document or query representations also provides better answers.

An ideal ranking consists of an ordering of the highly relevant, relevant, and irrelevant results where ties are broken using a ranking distance function or error measure, such as Kendall-tau, that we will discuss in a subsequent section. When a comparison of the web ranking, personalized ranking and ideal ranking, both the web and personalized ranking are close to the ideal ranking but not to each other implying that the web and personalized rankings are good in their own way. The relevance feedback system gives personalization "freshness" since interests may change over time but the corpus, user and document or query representations are difficult to estimate and depends on the sources of the information.

2.1.2 Query-information need

Most search queries are simple list of keywords assumed to be connected by an 'OR' connective. Most search engines support more advanced search options ranging from exact search phrase matching to other boolean connectives. The search results are typically a list of web pages ordered with respect to their relevance. However, some search engines distinguish resources that contain many useful links[114, 83]. It is also possible to create a summary of the search results by clustering them into subject headings to help users locate the specific topic of interest from their query [119].

Meta-search engines support the same type of queries as regular search engines and may provide similar results. The main difference is that they may not have their own index of the Web but access other search engines for their results. This has the advantage of increasing the coverage over that of a search engine [122]. By integrating the rankings of search engines, it is able to evaluate documents not strictly based on the text content but also the similarities with other search engines.

One of the problems specific to Web searching is rank manipulation. The notion of spam [47, 49, 59] or junk first referred to unsolicited emails and then became synonymous with Web documents that appear relevant to a specific query based on the ranking strategy but are judged irrelevant by users. Research has concentrated on identifying spam either by the textual content, Web document structure or by the structure of the Web graph surrounding the spam page. Fetterly et al. [59] perform a rigorous statistical analysis of over 550 million Web pages by recording information about each URL. Their results showed that some web spam are machine-generated with easily identifiable distribution of in-degree and out-degree from those of human-made Web documents. Most often sites manipulate their page or site content to increase their prominence to gain a larger audience for their site. Most quality judgments used by search engines are engineered to reduce the probability of sites from achieving undeserved rankings such as [18, 110].

2.1.3 Link Analysis

Before discussing merging algorithms that are designed specifically for ranks or relevance scores, we would like to discuss a general technique, called link analysis, that has been used to organize the information returned by search engines. Link analysis is a feature used in ranking for Web documents. The algorithms that make use of link analysis treat the Web as a structured graph of pages, edges indicating hyperlinks between documents. They make use of methods found in social networking [94] for analyzing social structures, citations of documents and modify these methods to judge the importance of a page by the number of pages that link to it. These methods can be roughly classified into general webpage ranking [28] and query-dependent ranking [83] methods. They rely on the social nature of the Web assuming that if a webpage is useful, it is likely that many people will link to it. In this respect, a link is treated as a vote for a specific page over all the others.

The PageRank [28] algorithm ranks all documents to compute a single importance value called the pagerank of that page. This measure models the behavior of a random surfer when at a specific page either chooses among the links on that page uniformly at random, or stops navigating and jumps to a page on the Web uniformly at random. The probability of navigation α determines how much the graph structure influences the PageRank values. If $\alpha = 0$, then all pages have the same PageRank value. If $0 < \alpha < 1$, then PageRank value is determined by both factors. Kleinberg [83] instead proposes a query-dependent method. In this method, first a query specific subset of the Web is constructed by querying a given search engine on that topic. This method assumes that there are two types of pages on the Web called hubs and authorities. A hub page contains links to pages with good content (high out-degree) and an authority page contains useful information (high in-degree). Hence, if a hub points to good authorities, then it is a better hub. Similarly, if many good hubs points to a page, then it is a good authority. Based on this self-reinforcing assumptions, a measure of hub and authority quality is computed for all pages in the given subgraph. Since the graph is constructed for each query, this is a more costly approach.

Both approaches to ranking have their strengths and limitations. Bharat et

al. [21] outline three prevalent problems associated with Kleinberg's connectivity analysis algorithm. The first problem encountered is the mutually reinforcing relationships between hosts. The hub scores assigned to documents are based on many interconnected documents linking to a single document. In the same manner, some high authority scores are given since one document constructs links to many interconnected documents. The *Mutually Reinforcing Relationships Between Hosts* problem exploits a certain organization of documents that heavily dominate the computation. The second problem is automatically generated links. Some Web documents are generated by other web services and as a by-product the web service will generate a link to itself from the the newly generated Web document. The third problem concerns non-relevant nodes. The appearance of non-relevant nodes tends to occur when popular documents are retrieved in the results but have little connection with the query.

Similarly, it is generally hard to construct a topic-specific graph by following the links in pages since the topic tends to shift quickly. In a more recent study, Bharat et al. [22] propose their *Hilltop* ranking scheme for popular topics which highly ranks the most authoritative pages on a query topic. The scheme operates under the notion that the experts on a subject can be located, where an expert provides unbiased recommendations and points to a number of documents with maximal disjoint connected subgraphs on the query topic. Thus, the mutually reinforcing relationships between documents would be minimized. These experts are located during a preprocessing stage and become a subset of the documents in the search engine's index. A page is considered an authority if some threshold number of experts on the query topic point to it. The results, in general, revealed that a more detailed classification of authority documents returned the desired Web document as rank 1. For the lower ranks, Hilltop performed similarly to one of the commercial search engines.

Local link information, such as in [83], only retrieves and orders documents associated with a specific search query whereas global link information orders documents based on the data collection first, such as in [28], then retrieves the relevant documents accordingly. Calado et al. [32] examines and compares the use of local

link information to global link information when answering a search query. The authors address the precision of the top-10 documents and show that the global link information improves precision while the local link information improves precision for documents ranked greater than the top-10, in the general case. The experiments only consider the precision error measure and gains that can be achieved in the number of common Web documents. The authors also compare combinations of local and global link information from the content of the web documents. The results show that a combination of content-based and linked-based sources give better retrieval results than using the methods individually.

SALSA (Lempel et al. [87]) is a stochastic version of the Kleinberg's connectivity analysis and prevents the effects of the mutually reinforcing relationships. Borodin et al. [26] expand upon the work of Kleinberg and introduce a theoretical framework for link analysis ranking. The authors present several link analysis ranking algorithms that are compared and assess of the quality of these algorithms through an extensive experimental evaluation. The algorithms proposed in this work provide a range of methods such as selecting the best hub or authority at each iteration, Bayesian methods or a simple in-degree operator.

PageRank is an algorithm that has been heavily researched to address specific issues and the research that proceeded the introduction of PageRank uncovers certain limitations of this algorithm and make attempts to resolve them. For example, Topic-sensitive PageRank [65] constructs a number of PageRank measures for each page in advance based on some pre-determined topics. Then, at query time, the relevance of each topic to a query is determined and used to combine these measures. The authors also claim that the topic sensitivity can be used to reduce the impact of heavily linked documents receiving a higher rank. Tomlin [115] compares its TrafficRank algorithm, which represents the traffic flow (incoming and outgoing edges) of the Web to determine the hotness or entropy maximization of documents, to the PageRank algorithm. The documents with large amounts of "traffic" are deemed to be important. The TrafficRank concentrates on the outgoing links while the PageRank focuses on the incoming links. When examining documents that are found deeper in the hierarchy (level 1 = /X/Y/, level 2 = /X/Y/Z/, etc), Traffi-

cRank produces comparable values to those observed by PageRank as more levels are considered in the documents.

Eiron et al. [54] address the problem of dangling nodes or link rot associated with documents that are linked to by other documents but have been moved or no longer exist. The PageRank algorithm either disregards these nodes or assigns an undeserved high PageRank value to these pages. To resolve this problem, a penalty propagating backwards is applied for dangling links, hence decreasing the importance of pages that contain such links. They also suggest variations of the PageRank algorithm, called HostRank and DirRank, that assigns importance to individual sites instead of webpages to address some of the issues associated with the vulnerability of PageRank to manipulation of links. Other works on link analysis can be found in [23, 89] where the stability of the PageRank values and the ranks computed from these values are examined.

2.1.4 Evaluation

There are several evaluation techniques used to access the relevance of documents in both IR and Web search. The best methodology for evaluating search engines is to perform experiments on the search engines themselves. Jansen et al. [73] analyze logs of real user queries against the search engine *Excite*. The authors investigate a multitude of factors including the queries, sessions and keywords. Their experiments reveal the mean number of search terms for a search engine (2.21) was much lower than traditional information retrieval systems (7-15). In addition, users only viewed the top-20 results in nearly 77% of requests. Queries that are not independent of each other should be viewed in tandem. The study also examines the components of consecutive queries by the same user as well as keyword frequency. Jansen et al. observe that Web search users are quite different from the traditional information retrieval system users. The users of information retrieval systems are usually aware of what information exists in the system and have a good idea of what information they would like to retrieve. A Web information retrieval system needs to be designed to cater to its search users.

The duration of a document in a search engine's database provides a greater

likelihood of this document to be returned. Search engines also have to consider how to assess document popularity, which is also strongly connected to the document's quality. In Cho et. al. [41], the research concentrates on identifying the impact of search engines on the popularity and evolution of Web documents. In addition, the researchers estimate the time needed for a new Web document to become significantly popular by search engines. As an extension of this research, page quality [42] is proposed as a new ranking metric that finds the high quality pages early in the execution in order to remove the popularity bias. The quality of a page is measured by the conditional probability that a user likes the Web document when observed the first time. Since the page quality is difficult to construct, the current popularity of the document where its popularity contributes to the estimation of the page quality. The authors consider their page quality definition and estimator as a third generation ranking metric with an emphasis on understanding the evolution and change in the link environment. In another approach, Pandey et al. [100] examine the effects of performing partial randomization of query results. The newly-created but highly relevant Web documents are not viewed by users since the focus is on the top-10 results. An algorithm then interjects a percentage of these newly-created but highly relevant Web documents randomly in the result set and shows that new pages will be located faster over the traditional link building method.

Assessing Quality of Rankings. The quality of rankings can be evaluated typically using precision and recall measures [108]. Both of these measures are well-known methods in information retrieval literature based on the notion that the documents that are relevant to a query are known in advance. *Precision* is defined as the proportion of received documents that are relevant. *Recall* is defined as the proportion of relevant documents that will ever be returned. Researchers capture the precision and recall of a search query through a precision-recall curve. The objective is to have precision and recall both equal to 1. However, these two measures generally require balancing opposite trade-offs. Precision and recall are the basic foundation methods of evaluating search. However, most search engines target precision as their primary objective since the number of relevant documents is usually

too high for a user to process. TREC-style average precision (TSAP) as used by Lu et. al. [91] considers the location of the relevant documents such that the i^{th} Web document receives a value of $\frac{1}{i}$ if it is relevant, otherwise the value is 0. The values are then averaged. TREC-style average precision is similar to precision but incorporates the positional information.

Järvelin [74] introduces new evaluation methods, the cumulated gain and cumulated gain with discount measures, and compare them with precision-recall curves. The cumulated gain measure assigns a relevance score to each Web document, which is the sum of ranks up to that particular document. The cumulated gain with discount is a modified version of the cumulated gain measure that also incorporates a discounting function in the document's score since a document becomes less valuable if placed deeper in the ranked list. This discounting function is to divide the document value by the logarithm of its rank. The case study demonstrates that the cumulated gain-based measurements provide richer information for evaluation and the tradition precision-recall curve, in which relevance scores are either 0 or 1, may be too lenient and not able to properly distinguish relevant documents from irrelevant documents.

Buckley [31] introduces *bpref* which makes use of the relevant documents associated with a topic by computing the average number of documents that are ranked above the set of relevant documents. They compare *bpref* with several variations of the precision and recall measures and shows cases where it outperforms the traditional precision and recall. Bpref is developed for the cases when information is not complete and considers incomplete and imperfect information.

Vaughan [118] tackles the definition of recall, which is a subjective measure with many interpretations since the universe of relevant documents is unknown. The discovery of additional relevant Web documents require deeper traversal of the query results therefore adversely affecting the precision. Vaughan also proposes two measures that are counterparts to the widely-used precision and recall. Vaughan's precision is the quality of result ranking, which is measured by the correlation between search engine and human ranking. The Spearman rank-order correlation coefficient is computed, where the higher the correlation coefficient indicates that

the search engine ranking is closer to the human ranking. The Spearman rank-order correlation coefficient [50, 51], also known as the footrule distance, is defined as $1 - \frac{\Sigma D^2}{N*(N-1)}$ where D is the difference between an object's ranks in two rankings and N refers to the number of objects. Vaughan's recall is the percentage of top ranked documents that are retrieved. This modified recall makes an assumption that a portion of the retrieved Web documents are relevant, then we can calculate the top-K. This research assumes that the Web documents needed to accurately compute the precision, recall or one of their variations is available to the search engine for evaluation.

In addition to precision, recall and footrule, a popular method to evaluate search engine rankings is through the Kendall-tau and footrule measurements. The Kendall-tau distance compares two rankings and determines the degree of sorted-ness. The Kendall-tau measure performs pairwise comparisons of the rankings where a disagreement is assigned a penalty of 1 and 0 otherwise. Ideally, we would like to have each document relevant to the query in its correct position. The objective is to have the minimal number of disagreements between two rankings. A more elaborate discussion of these evaluation methods will occur in a subsequent chapter.

The previously mentioned algorithms perform experiments on static result sets and query logs. However, the content of the Web changes daily. This calls into question the validity of the claims and observations over time. Bar-Ilan [11, 12] conduct extensive experimentation of search engine performance on a weekly basis and compare results to previous weeks. The results reveal how specific search engines behave for the same set of queries on a weekly basis where the documents are no longer viewed as static but may be altered over time, removed or newly introduced to the search engine's indexing system. The documents are analyzed over time where properties are measured such as relevance, weekly precision changes and number of forgotten or dropped documents. The tests did not provide significant insights of search engines in a week-by-week comparison. However, the rankings on AlltheWeb were, in general, stable during the weekly experiments. Google, on the other hand, observed constant but local changes in the positional order of Web documents due to the fluctuations of search engine's index and influx of new Web information.

Interestingly, the number of overlapping Web documents was low; hence difficult to perform a proper analysis.

Vaughan [118] uses human subjects to evaluate search engine performance by balancing the precision and recall measures. The author compares human ranking to rankings returned by a search engine. The use of human subjects comes with the caveat that humans are not experts for all search queries where their opinions are subjective in which the definition of an expert is ambiguous. In the case of Vaughan, human ranking is time-consuming, where the human ranking is an agreed upon rank of each Web document. The quality of result ranking indicates that the correlation between Google and the human ranking was the best when using the Spearman correlation coefficient. Google is shown to be the best search engine for retrieving the most relevant pages in this study and has the most stable search engine performance.

Assessing Availability and Cost of Retrieval. All the algorithms we discussed earlier consider the ranked list are already in memory before aggregation. In the literature [36, 57, 116, 30, 52], monotonic aggregation functions that incorporate weights are used to represent different ranked lists. A monotonic aggregation function satisfies $f(x_1, x_2, \ldots, x_n) \leq f(x_1^{'}, x_2^{'}, \ldots, x_n^{'})$ wherever each $x_i \leq x_i^{'}$. However, in some cases retrieving large ranked lists may be very costly. The following algorithms approach this problem.

The medrank [56] algorithm approximates the median rank of a set of rankings by first sorting the ranked lists with respect to arbitrary random vectors. Then, these lists are read in sorted order and the median is calculated when the rankings are found in 50% of the lists. The Threshold (TA) and No Random Access (NRA) algorithms, which are improvements on Fagin's Algorithm (FA) [90], operate on systems where objects can be retrieved either with ranked retrieval or by directly finding the rank or score of an object through random access or probing. Other algorithms in this area are Quick-Combine [63] and Stream-Combine [64], which are proposed to retrieve multi-feature objects efficiently and compared with TA and NRA algorithms. The cost of Quick-Combine and Stream-Combine do not have an

constant factor upper-bound making these algorithms having a smaller amount of flexibility and scalability.

The minimal probing (MPro) [36] algorithm addresses the case when a portion of the ranked lists are accessible in memory while the other ranked lists are not readily available. For instance, a ranked lists can be constructed using a user-defined function, which is only accessible on a per-object basis. MPro first orders the objects with respect to the rankings that can be directly accessed in a database. Then, the objects are ordered with respect to the ranked lists that can not be stored in memory. MPro is compared to the Onion technique proposed in [37] which forms a hierarchical organization of the data and indexes using convex hulls. The Onion technique provides significant speedup over a linear scan of the database. MPro improves upon the Onion technique by reducing the computational overhead of organizing the data in convex hulls and producing an algorithm that is more dynamic in nature. Marian et al. [93] analyzes the TA and MPro algorithms alongside their own sequential algorithm, Upper, to maximize parallelism amongst different databases to reduce query response time while minimizing the number of accesses to objects. The Upper object allows for partial probing of objects by intertwining random and sorted accesses and addresses the response time bottleneck for the TA algorithm.

Other methods have been presented in [39] where the rankings can only be accessed via indexes. The authors make use of filtering conditions for pruning irrelevant information and applying a ranking expression for ordering to the set of acceptable objects. However, the work assumes only boolean expression in the filter condition which are connectives of inequality statements. Tsaparas [116] describe an index structure for pruning the tuples in a database join with respect to top-K queries of monotonic linear combination functions. Bruno et. al. [30] develop methods to estimate an approximate range for nearest neighbor queries using histograms. This work uses a p-norm function to determine how close an object is to a query and adjusts the algorithm to the given workload.

Hristidis [67] created the PREFER system which materializes views over a set of documents to be ranked using weighted linear preference queries. One or

several views can be merged to construct an answer for a query. This approach is not practical in cases where the data is highly dynamic. Later, Hristidis et al. [68] expanded the PREFER system and investigate other methods to compute preference scores in queries using (1) the linear combinations of source attributes, (2) the linear combination of monotone function and (3) the cosine function. Babcock and Olston [10] addresses top-K processing in a distributed environment, where data streams incur the high costs due to the number of transmissions and a reliance of a central repository. The authors propose a scheme that stores a certain ranking given the observed values from data streams for a given time window. Wechsler et al. [120] devise a ranking algorithm which minimizes the response time. This branch-and-bound algorithm is implemented to reduce the transmission time and inspection time needed to process video and audio data formats.

The algorithms discussed above consider documents that were outside of a database. However, the database does not need to just store the information but can be queried and return a ranking. A set of database operators, rank-aware operators, are developed to perform the appropriate classification of the information stored in the database. Ilyas et. al. [70] introduce a rank-join operator that conducts only sorted accesses and develop conditions for early returns if no other object can produce a higher score. J^* [99] is presented for executing top-K *join* queries with user-defined rankings and also assumes only sorted access. Ilyas et al. [72] compare their hash rank-join operator and J^*, Ilyas et al. [71] outline a pruning methodology which estimates the input cardinality for query optimization techniques to their hash rank-join operator. Bruno et al. [29] also present models for ranking objects in a database by using conditional selectivity of the statistics.

Since documents can be stored outside or inside a database, caching can be used to reduce the cost in both cases. The basic problem with caching is deciding what to cache and how long the cached data should remain. Chaudhuri et al [38] addresses the *Many-Answers* problem as a result of a database query retrieving many query results and proceeding to rank these results based on their properties, which looks at both a global score and a conditional score. Saraiva et al. [109] combines two types of caches: cache of query results and cache of inverted lists.

These schemes store rankings associated with a given query, in the case of the cache of query results, or a given keyword, in the case of the cache of inverted lists. The eviction from these caches remove all related objects. The cache of inverted lists consists of *equal byte-sized pages* which are stored portions of the keyword's inverted list but due to the equal pages, more objects are stored than requested (case of indirect prefetching). Lempel et al. [88] group the results based on the result pages where result page 1 has documents 1-10, result page 2 has documents 11-20, and so forth. The grouping of 10 documents on a result page neglects these documents' independence with respect to different but similar queries. Documents are prioritized based on the query topic and entry time frame.

2.2 Merging Algorithms

The merging of information from different sources appears in other areas of computer science such as robotics and machine intelligence, called sensor and data fusion. Sensor fusion is a method of integrating data given by a set of sensors in order to obtain good estimates of a dynamic system's states. Sensor and data fusion is similar to the meta-search problem in the challenges of identifying, retrieving and interpreting data. Sensor networks have the ability to process the data at a sensor before it is sent through the network while meta-search only considers each document's relationship to the other documents e.g. whether or not there is a direct link. Multi-sensor fusion provides coverage of an area using many sensors. The data at these sensors can be merged or fused together to give a consistent and understandable conclusion. A number of strategies address specific issues in multi-sensor integration for a range of applications including artificial intelligence and military operations, which are reviewed by Abidi and Gonzalez [1].

Klein [82] reviews algorithms and architectures that detect, classify and identify sensors' information transmitted through the network. The three main approaches to capturing the information obtained by the sensors are classical inference, Bayesian Inference and Dempster-Shafer evidential theory. Classical inference models uncertainty using uncorrelated random noise. Bayesian inference better defines uncertainty through using previously retrieved information in order to better

estimate the likelihood and probability of error. The Dempster-Shafer evidential theory, on the other hand, models uncertainty with the belief in one or more proposition, including ignorance. This theory incorporates degrees of belief and allows for the unknown which is not present in the other models. These methods typically require a large amount of prior information in order to provide useful estimates and inferences.

Data fusion on the Web has been researched in Tsikrika and Lalmas [117]. Their objective is to compare five merging techniques and maximize the precision. However, their merging methods assume that the title and summary information is accurate and recently updated. The Dempster-Shafer method is applied but it can only merge two bodies of evidence at a time therefore making the outcomes dependent on the order of the merging. Both these merging methods do not consider the existence of incorrect data or spam.

2.2.1 Overview of Meta-Search

Meta-search engines leverage the content from individual search engines by increasing the coverage of the Web and answering the queries more accurately. Fusing search engines have several issues. These include the following: search engines only provide ranks (but not relevance scores) and the documents existing in each search engine's database may not be the same. The three main components of a meta-search engine are *resource description*, *resource selection* and *result merging*. Resource description provides information about the contents of individual databases. Resource selection chooses the set of databases to be used for search and retrieval. Result merging, which is the focus of the thesis, is the procedure for collecting the data from the individual databases and merging the results into a consolidated ranking.

Resource Description. Resource description tries to obtain an accurate model or representation of the databases of different search engines. This area of research typically deals with crawling the Web for new reachable documents that may better satisfy users information needs or updating existing documents that are stored in the search engine's database. Search engines crawl the Web to index data at a

centralized location and use this for search queries. The performance and cost of crawling the Web can be time-consuming and expensive based on the number of documents retrieved and the number of local search engines accessed. Due to the largeness of the Web, accessing all Web documents is impossible in the absence of an appropriate link structure. In meta-searching, these representations can be used to better select which search engine's databases is more adapt to contain relevant documents.

Craswell et al. [48] is one such example that proposes a selective meta-search engine that is a hybrid approach to crawl Web documents. The selective meta-search engine sends the query to several satellite servers to reduce the query time. The objective is to have the servers that can answer the query. Their experiments show applications where traditional local search engine Web crawling, traditional meta-searchers adopting the search engines results and selective meta-searchers are the cost effective choice.

Resource Selection. Meng et al. [95] incorporates the possibility of hundreds of thousands of scalable search engines. They devise a new database selection method that only computes the similarity of the most similar documents in each local database and forms a representative for all databases to represent the collection of databases. Then, the algorithm can determine which databases are most likely to contain the top documents to be returned to the user. A series of experiments are performed to evaluate the percentage of correctly identified databases, percentage of correctly identified documents, database search effort and document search effort. Their new database selection method is showed to be scalable in both computation and space.

In other research efforts, Wu et al. [121] also concentrate on the problem of developing a scalable meta-search engine. This meta-search engine constructs an integrated representative method for the databases using the similarity of the most similar documents. Their method stores a fixed number of documents for each keyword regardless of the number of databases that it appears. The results show that the integrated representatives is highly effective in correctly identifying both

the databases and documents. The number of databases and documents searched is also bounded within a factor of 2 of the number of documents requested by the user.

Pon et al. [101] proposes a framework for ranking data sources using a learning model for data source accuracy by comparing data source behavior to determine the best k data sources. A cohesion function is defined which determines the accuracy of the data sources by observing how well the data sources agree with each other. Once the ranking of the data sources is complete, the best data sources can be selected to answer the meta-search query by maximizing the precision and recall. Other database selection algorithms are found in Powell and French [102] and Andritsos et al. [8].

Result Merging. With models of how to represent and select the necessary databases through resource description and selection, the challenge is how to produce a final ranking from the individual ranked lists returned from these databases. The method to merge results from multiple ranked lists (from a single database or from multiple overlapping databases) has produced many algorithms, such as [9, 62, 97, 111, 112], that fall into two main categories: ranked-based and score-based. For a general overview of other meta-search engine research, see Meng et. al. [96].

2.2.2 Rank-based Merging

In many systems where the aggregators have no access to the internal scores of the documents for a given query, rank-based merge methods must be used. We observe in the literature that aggregation methods are developed using a non-learning (static) or learning approach. We now describe methods for each approach.

Non-learning approaches. Borda's method [25] derived from voting takes the position i of objects in a list of top-K objects and assign it $K - i$ points. The points are then added for each object to find the final ranking. The results are equivalent to taking the average of the rank values. The objective of Borda's count is to provide an unbiased representation producing an aggregate ranker where each ranker is treated equally. Each rank aggregation method begins with some set of

initial rankings in order to determine the most relevant objects. The next stage is to determine the order of these relevant objects. Many of the following algorithms try to find a ranking that minimizes the total Kendall-tau distance to the input rankers. We first discuss the algorithms presented in Dwork et al. [53] and Renda et al. [104] in which the final ranking is constructed using Markov Chain models. Then we describe a series of heuristic algorithms introduced by Chin et al. [40] and Beg et al. [16]. Lastly, we review the RankBoost algorithm [61] that differs from the other algorithms in that it is based on a learning framework and forms a good ranking from a weak set of rules.

Dwork et al. [53] claim an aggregate ranker with the lowest total Kendall-tau distance models a consensus ranker. They claim the consensus ranker has the ability to reduce spam since a junk page is unlikely to have a high rank in a majority of rankings. Since computing the Kendall-tau optimal ranking is NP-hard, they introduce four Markovian Chain methods (MC_1, MC_2, MC_3, MC_4) that approximate the Kendall-tau optimal ranking. Each chain has documents returned by rankers as states and transitions are based on the ranking of objects in the input rankers. In MC_1, the next state is chosen uniformly from the set of documents that are ranked higher than the current document for all rankings. In MC_2, the next state is selected by first choosing a ranking that ranks the current document and randomly transition to a document that is ranked higher. The Markovian Chain MC_3 is similar to MC_2 except that the next state can be selected uniformly and a transition occurs if the next state is higher ranked than the current state. In MC_4, the transition occurs if the next state is ranked higher for the majority of rankings. In addition to the Markovian Chain methods, the Local Kemenization method is introduced, which begins with a ranking, e.g. any one of the Markovian Chain methods, and then performs a series of adjacent swaps as long as the Kendall-tau distance reduces. These rank aggregation methods are compared with the Borda's count and methods that optimize for the footrule measure. The authors suggest that spam documents appear in the final ranking if a majority of the input rankings include the bad documents. In other words, garbage in leads to garbage out.

Renda et al. [104] perform a more extensive study of the Markov chain methods

presented by Dwork et. al. [53]. In addition to constructing the aggregate ranking using the Markov chain methods, the several score and rank normalizations are performed for each object for every ranking where the normalized weight of 1 would be received a rank of 1. The frequency of each object in the rankings is computed. Markov chain methods perform better than the rank normalization methods. MC_1 and MC_4, in general, perform better than the other Markov models.

The problem of finding a ranking that minimizes the total Kendall-tau distance can be reduced to the minimal feedback arc set problem as shown in [14]. Ailon et. al. [4] develop the FAS-Pivot algorithm and its variations that solves the minimum feedback arc problem. The FAS-pivot algorithm is identical to the Condorcet-fuse algorithm [97]. Condorcet-fuse represents the ranked information in a graph where the weighted directed edge orders two documents based on the ordering in the majority of the rankers. The algorithm merges this information by sorting the documents using quicksort. Other algorithms for solving the FAS problem are introduced in [106] where the graph is partitioned recursively to maximize the cut between two subgraphs. Aslam et al. [9] introduces the Borda-fuse algorithm, which is based on a voting scheme. The algorithm assigns a certain number of points for each rank position. The object that receives the most points from the different rankers wins the voting election. Borda-fuse is similar to Borda's count; however unranked objects are assigned some points while in Borda's count gives unranked objects zero points. Condorcet-fuse is shown to outperform Borda-fuse in [97].

In the special case of federated systems where searching is performed on a variety of sources including database systems, Khoussainov et al. [79] address the challenges of federated web search by concentrating on the organizational dynamics through learning techniques for topic-specific search engines. Their model is a decentralized management of the various resources that incorporates an economic framework. The objective is to optimize the effectiveness of the search engines and the meta-search layer where the aggregated results are best-suited for the query.

Learning approaches. The relevance of an object may also be learned implicitly through using historical information or explicitly using human subject studies. We

mention several works that use human subjects to rank objects and other learning models that tries to infer the ranking in the presence of training data. The study in [75] tries to identify the relevant documents by recording this data for a specific user. The clicked web documents provide ample information in distinguishing the more relevant Web documents from the others returned by the search engine. This work shows that by using these preferences, the system is able to learn the factors that are most relevant to a user in ranking documents. Amento et al. [7] also set up a small-scale user study comprising of several experts for a set of popular search queries. The objective of the study is to understand how the rankings match human opinion using in-degree, out-degree, PageRank and Kleinberg's authority and hub score. Similarity and correlation among the Web documents is described by using footrule and Kendall-tau for the top-5 and top-10 Web documents. The conclusions indicate that the in-degree metric performs just as well as the more complex PageRank and Kleinberg approaches. The experts ordered the Web documents with some consensus but not enough to warrant a decisive conclusion.

Cohen et al. [44] addresses the problem of ordering rankings using ranks, user feedback and a learned preference function. A two-stage approach is outlined to learn how to order objects given a set of rankings by first learning the order relationship between each pair of objects and second to maximize the number of agreements to the relationships produced in the first stage. The ordering of every pair of objects is learned by minimizing the disagreement between the user feedback and the current preference function. Each ranking initializes its own preference function for each pair of objects where each preference function is weighted equally; however as convergence occurs, the impact of the user feedback modifies the weight of each preference function. Two greedy and one randomized algorithms are introduced, including a topological sort greedy algorithm using user feedback, as approximations to maximize the agreement amongst objects with respect to the learned preference function.

The Bayes-fuse algorithm [9], presented alongside the Borda-fuse algorithm, computes the probability of relevance and irrelevance for each object in each ranker. This algorithm makes the assumption that the ranks assigned to a given relevant doc-

uments in two different rankers are independent. In order to compute the relevance of an object in the final ranking, we sum the log of the ratio of these probabilities for all rankers. When the input rankers are more diverse, the Bayes-fuse algorithm would be a better choice over the Borda-fuse or CombMNZ algorithms.

The cranking model [85] proposes a generalized variation of Mallows model that aims to order objects, e.g. web pages, according to their input rankers through estimating each object's rank. The maximum likelihood estimation approach is used to learn the proper settings for the location parameter (π) and dispersion parameter (Θ) given a set of partial rankings. As a result, the expected rank of each object can be calculated and a final ordering of the objects can be returned to the user. Cranking aims to build probability distributions over the input rankers to minimize the rank rate, which is defined as the average of rank of the correct page. In a realistic setting, the correct ranks are unknown and impossible to obtain. The experiments assumes that there exists correct pages to answer the queries and the search engines (rankers) have these correct pages.

The Condorcet criteria states that if an object is ranked ahead of all other object by the majority of voters (or rankers, in our case), then this object is the winner. This winning object may not exist if at least two objects have a different set of majority votes. The extended Condorcet criteria says that if the set of objects can be partitioned such as one subset are ranked above the objects in the other subset, then the first subset should be declared the winner.

Coherence [40] is a heuristic rank aggregation algorithm using a weighted version of the extended Condorcet criteria and optimizes the final ranking with respect to the Kendall-tau measure. Weights are placed on the objects based on which ranking it appears. An object o_i is ranked above an object o_j if o_i has a higher value (object rank times input ranking weight) than o_j in a majority of the rankings. The consensus ranking is further optimized using a modified version of the Local Kemenization [53] optimization. But instead of only performing adjacent swaps, their algorithm allows for swaps of neighboring objects while maintaining the weighted extended Condorcet criteria. The adjustment procedure initializes with one object that has the greatest likelihood of having rank 1. With each new object added

during the adjustment procedure, a new ranking is formed, including the new object, while maintaining a consistent ordering. The authors then apply Coherence to a polynomial time approximation scheme (PTAS) in order to assign the optimal rank to each object. The PTAS tries to learn the best placement of each object by repetitively sampling objects and computing the probability of each object having the optimal placement.

Beg et al. [16] propose a rank aggregation algorithm which optimizes the footrule measure by minimizing the distance between each input ranking and the aggregate ranking. Their genetic algorithm (GA) uses reproductions, crossover and mutation to construct the combined ranking through a user-defined number of iterations. For reproduction, the set of valid permutations are evaluated in turn and only the orderings that gives a lesser value are retained. The crossover model decides to exchange a group of the first K objects for a pair of rankings only if the new rankings remain valid permutations. The mutation technique will randomly swap the "to be mutated" object with another object. Other soft computing methods that incorporate fuzzy information are presented and compared. The membership function ordering (MFO) assigns a rank to each object based on the maximum value of the mean and variance of the object's positions. The mean-by-variance (MBV) constructs a ratio of the mean and variance of each object in order to avoid loopholes in MFO. The entropy minimization technique uses an entropy equation for a particular region that moves a threshold value between upper and lower bounds until the minimum entropy is located. When comparing the Borda's count and GA, GA produce the better results with a low number of iterations. When comparing MFO, MBV and entropy minimization techniques, the entropy minimization method revealed to be the best performer in the aggregate distance and execution time, following the Borda's count.

Freund et al. [61] presents a formal framework for learning how to accurately order documents by combining their ranking functions. Assuming the *a priori* knowledge of the relative ranking of individual pairs of objects, the RankBoost algorithm uses the input rankers and a feedback function to determine an accurate ordering of each object pair. Each iteration of the algorithm emphasizes different aspects of the

training data and a weak learner that produces a weak ranking. The weak ranking can then be updated to find the optimal location of each object. The aggregate ranking is a weight combination of the weak rankings. Their work is not concentrated on the scores or ranks assigned to the documents, only the relative order and describes how to determine a good ranking given good feedback. The results show that the RankBoost algorithm was able to locate highly accurate prediction rules. RankBoost also maintained its good performance with varying dataset sizes.

2.2.3 Score-based Merging

When scores are available from individual rankings, they can be used to provide more effective aggregation methods. In some cases, scores from search engines may not be available but additional scores for each ranked object can be obtained by the meta-search engine at query time. As done in Section 2.2.2, we discuss both non-learning and learning approaches for aggregating scores.

Non-learning approaches. The Borda's method [25] cannot be applied for scores but mathematical functions of average, minimum, maximum or weighted versions of these methods can be used depending on the objective of the user. Rank aggregation methods can also use one of these mathematical formulas but the domain of the ranks for each ranking is uniform while the domain of score-based rankings do not fall into a fixed interval.

Lee [86] examine several static score-based methods that makes use of multiple resources through constructing a final ranking based on a linear combination approach. The algorithms include: the CombMIN, CombMAX and CombSUM algorithms which construct the final ranking according the the minimum, maximum, summation of the object's scores in each ranked list. The CombANZ and CombMNZ are variations of the CombMNZ algorithm that produce the aggregate ranking by dividing or multiplying the number of ranked lists the object appears by the CombSUM value. Since score values can be duplicated, a normalized similarity value is computed based on the minimum and maximum values observed in the ranking. Without similarity normalization, the CombSUM method produce the best average precision. With the similarity normalization, the CombMNZ method gives similar

or better average precision results over CombSUM. Their work shows that different rankings contain the same set of relevant Web documents but a different set of irrelevant Web documents. The CombMNZ is generally considered the standard for static score-based algorithms as it is used as a baseline algorithm in many of the work described below.

Renda et al. [104] compare rank and score-based aggregation methods. A z-score and score normalization methods are also constructed. When based on scores, the performance evaluation show that frequency counts have little bearing on the average precision results. The z-score normalization outperforms score normalization. The score normalization using the number of times an object appear in different rankers wins the majority of the tests. The Markov chain models [53] and score normalization methods produce comparable performance result contrary to the popular assumption that score based methods are better.

Motro et al. [98] consider that the ranking or fusion of data is flawed with respect to assessing the usefulness of data originating from multiple sources. They present a utility function that is based on the performance of the sources including the features of recentness, cost, accuracy, availability, priority and quality of the data. The objective is to maximize the utility function, which is a linear combination of features and the weights assigned to each feature. The research outlines how to formally represent the features and constructs, several empirical examples of how the features can be used to determine the ranking and/or fusion of data. The features and associated values are assumed to be accessible for evaluation purposes which may not be the case in a real-world situation.

With data coming from news sources, Rasolofo et al. [103] introduce a method that handle the challenges of frequently updating Web documents. The baseline meta-search strategy is a ranking function that uses a round-robin approach to create the final results ranking. Additional information used to rank results include a generic document scoring function that assigns a raw score to each document regardless of its source based on the number of matching keywords in the document, the number of keywords in the query and the length of the Web document. The generic scoring function may also incorporate the document title scoring or docu-

ment summary score, blended scoring function of the document title and summary, document date, server usefulness or content-based scoring that requires the full text to be downloaded and stored. The experiments concluded that a blended scoring function based on the document information outperformed the round-robin and content-based scoring.

Learning approaches. Si and Callan [111] consider merging results in a distributed environment. A query-by-query tuning approach on the results merging function is taken to convert the database-specific scores into an approximation score that can be adjusted. In an extended version of this work, Si and Callan [112] develop a learning algorithm that maps the database-specific document scores to database-independent document scores. As queries are posed to the selected databases, the centralized database is also receiving the queries and updating the database-independent document scores of the returned results. The main premise of the research is to construct a merged ranked list of documents that could approximate the final ranked list as though the databases were stored collectively in one single large database.

Gravano et al. [62] presents the TOP algorithm, which explores result merging from heterogeneous data sources with the use of a metabroker. The metabroker serves as a centralized system that accepts the query, obtains the relevant data even if it is not ranked highly on the data sources and organizes the results. The algorithm considers each ranker as an attribute to the query in which the objects are ranked only according to one criteria. The objective is then to learn the optimal weight of each ranker to satisfy the query posed by the user. The stop condition occurs when the target weight for each ranker is achieved within a given threshold. However, the TOP algorithm can not guarantee efficient execution since a large portion of data may be retrieved.

2.3 Preferences and Personalization

The merging algorithms makes an assumption that documents can be reduced to a rank or relevance score and the documents with high rank or relevance score

matches the users needs. The main issue is that users information needs vary. Users pose a query to an engine with the expectation of retrieving self-defined relevant documents. However, the determination of relevance may not be binary for a user, as well as, relevance is undecided for some (or most) documents.

Quantitative evaluation of documents or objects occurs when relevance is formed numerically through ranks or scores. The goodness of objects is based on a scoring function, which is a mathematical formula that maps the properties of the Web information to numeric values. The other method for understanding the relationship between Web information is qualitative. The qualitative approach does not make use of scores for the data but instead assess the relative relationship between documents with inferences and deductive reasoning. Users tend to use the qualitative approach in order to express their interests. The use of scores is counter-intuitive to users since the data is viewed as independent components in the scoring function calculations. But, users tend to think in terms of categories such as "Person X likes A better than B". User preferences are defined as a series of statements that used in unison with individualized needs. The objective of user preferences is to personalize the search results to the specific needs of the users. Furthermore, preferences can also be used to evaluate the quality of rankings. We discuss the two types of preferences, representation of preferences and the role of personalization in the literature.

2.3.1 Types of Preferences

The types of preferences can either follow a strict or fuzzing ordering of the objects or objectives. Strict ordering sets one object as better or worse than another object. A fuzzy ordering is more flexible than strict ordering by allowing equality such as "A is just as good or better than B". A total ranking of object occurs when a strict or fuzzy ordering includes a statement for every pair of objects. A valid ordering can be formed if the statements are not conflicting such as "A is better than B" and "B is better than A". In either case, there is the issue of incomplete information. The statements defining the relationship between each pair of objects may not be available since the user may not supply them, has not made a decision

about the order or does not care about the order.

The work in [3] examine on how to express and combine preferences that capture the choices of a broad scope of users. Each entry in the search query contains a specific element or the wild card (*) element which indicates that the user accepts any value for this element. The preferences are combined and becomes numeric scores and then sorted in ascending order. In the case when the preferences are undefined or unknown, the preference can be generalized to include only the known user interests. The problem with using numeric values is that users do not identify their interests by numbers but by determining the relationship between each pair of preferences.

Ross [105] delved into fuzzy ordering where rankings represent Gaussian distributions with different mean values. The dependence amongst the objects are assessed through pairwise comparison. Given two objects, x and y, and some comparison function c, a matrix can be constructed for each pair of objects where $c(x, y) \neq c(y, x)$. The pairwise comparison offer better representation of the objects with less sensitivity to bias. The benefit of these approaches is also its primary drawback, which is the lack of expressiveness of data. The information is represented are numeric scores. Numeric scores allow for easy comparison; however the cause of the actual distinction among tuples is removed.

Strict ordering is considered in [80], where preferences are modeled in terms of the intuitive 'better-than' scheme that is called strict partial orders. Preference terms are proposed and defined in order to capture the user's interest such as the AROUND and BETWEEN preferences. The resulting preference algebra addresses many intricacies of preferences but the use of this approach is not straightforward and it is not flexible to the construction of new preference terms. The algebra cannot handle objects that are equivalent or contains indifference. Kiessling [81] extend the preference algebra to deal with equivalent and indifferent objects. Holland et al. [66] mine preferences using this preference algebra by recording the frequency of a value and defining numerical and categorical data-driven preferences. The complexity of preference mining is discussed to be polynomial-time with regard to their domain size but the experimental evaluation only considered only two to six preferences

where numerical data domain is 200 values while categorical data domain has 20 categories.

2.3.2 Representation of Preferences

Preferences can be represented as operators [15, 20, 35, 43, 80] or, in the simplest form, numeric values [3].

Chomicki [43] propose embedding preference formulas in relational databases using the winnow operator. The winnow operator returns the best match to the given query and subsequent top objects are retrieved by multiple executions of the winnow operator. Independently, Kiessling [80] developed another operator that is similar to winnow, called the Best-Match-Operator (BMO). The theoretical foundations of preference constructors are discussed with definitions and structure to how to efficiently process preferences. These constructors can be issued in conjunction of each other and/or prioritization of at least two preference formulas. Most importantly, due to the fuzzy nature of preferences, the author concludes that the intermediate stages do not suffer from the flooding problem, many potential candidates for top-K which are subsequently disregarded. As with most preference formulas, the operator serves as a filtering technique which returns top candidates, sometimes more than one Web document, that can be ordered. If the operator returns $K' < K$, then the operator performs another execution with a new answer space excluding the previous results. The winnow operator does not impose restrictions on the preference formulas as in BMO. The winnow operator have more expressive power and can represent user interests in better detail. The Skyline operator [27] retrieves the set of objects in a relational table that are not dominated by other objects. Both winnow and BMO can be used to express preference queries and a generalization of the Skyline operator. Another limitation of the Skyline operator is its handling of partially ordered attributes, which is addressed by Chan et al. [35] by devising algorithms to exploit the index structure of the data. Other research such as [15, 20] has concentrated on handling multimedia preference queries.

A step in the right direction is concentrating on properly understanding how to construct preferences so they can be discovered by an information system or

database but still be intuitive to users. Holland et al. [66] models preferences as strict partial orders. Algorithms are presented to automatically mine the strict partial orders. These preferences can be numerical, categorical or a combination. The numerical preference categorization are defined as upper and lower bound frequency counts with a fixed error interval. Histograms are used to efficiently estimate the density of the numerical preferences. For categorical preferences, the upper and lower bounds are clear since a user will like or not like a property. The k-means clustering algorithms is implemented to group the categorical preferences. In the evaluation of preference mining, precision and recall are the performance measurements with precision at about 60% while recall is set lower at 20-40% depending on the type of preference. When investigating the runtime of their model, the numerical preferences mined the fastest, then the categorical preferences. The explicit preferences, well-defined preferences, is time-consuming and almost linear in the number of tuples.

Query personalization, a process of dynamically enhancing a query with user preferences, is addressed by Koutrika and Ioannidis [84]. A personalized answer is based on K of the top preferences from the user profile that should influence the answer and at least l ($l < K$) preferences that are satisfied. A general model is proposed that combines expressiveness and concision. A degree of interest for each query is defined given the user preferences satisfying the profile's values. The preferences may be positive, negative or indifferent. A user's concern is captured by the presence or absence of values. The elasticity of preferences may be exact or elastic depending on the numerical or categorical nature of the values. The personalization of query results is constructed through a personalization (acyclic) graph that connects relation nodes and attribute nodes with selection edges and join edges. This graph is constructed and maintained according to the degree of criticality using their FakeCrit Preference Selection algorithm were all edges are assigned the maximum weight in order to reduce the storage and maintenance costs. Another algorithm, Progressive Personalized Answers, is presented that generates ranked, personalized and self-explanatory answers. The experimental evaluation shows the benefit of both of these algorithms and the effectiveness of personalized

searches over non-personalized searches.

CHAPTER 3
BASIC CONCEPTS

This chapter presents the definitions of the terms used in this thesis. We first describe the search problem and the result ranking problem. In Section 3.2, we review the performance measures used as comparison tools to evaluate the quality of the rankings.

3.1 Terminology

Suppose there are n *objects* denoted by $\mathcal{O} = \{o_1, \ldots, o_n\}$ that exist in a database and can be queried. Each object o_i has content and a set of properties or features, denoted by $f_1, \ldots, f_m$. The features are positive scores that correspond to the relevance of the object for a specific criteria. For example in a web searching application, objects can be referred to as web pages and features can be keyword occurrence, recency of web page update, or retrieval frequency. A *search query* Q, is an information need, normally expressed by a set of keywords, e.g. words or phrases. Features may or may not be query specific. A *search engine* or *ranker* assigns a positive weight w_i to each feature f_i from an object $o \in \mathcal{O}$. The weights $\mathbf{w} = \langle w_1, \ldots, w_m \rangle$ constitute the weight vector of this ranker. Rankers normally contain (or index) a subset of $\mathcal{O}$. We write $o \in r$ to denote that object o appears in a ranker r. The set of all objects indexed by the ranker r constitutes its database, denoted by $DB(r)$.

Given a search query Q, a ranker estimates the values for each feature f_i of an object $o \in r$ that it indexes. The value of each feature f_i is denoted by $o.f_i$. Note that in the following discussion, we will assume that the query is fixed for a specific test for a set of rankers and hence we will not specify the query explicitly unless it is necessary. The overall score $score(r, o)$ of the object o for ranker r on a query Q is then given by a function over its features and the weight function of the ranker. One possible function is the *linear combination function* given by $score(r, o) = w_1 * o.f_1 + \ldots + w_m * o.f_m$. Generally, all features have values between

0 and 1 and the weight vector is given by $w_1 + \ldots + w_m = 1$. The scores given by each ranker also take values between 0 and 1 where 1 refers to the best match and 0 refers to the worst match. The scores are sorted in descending order. The scores can then be translated into *ranks* of monotonically increasing integers. Let $rank(r, o)$ denote the rank of object o in ranker r with 1 being the highest rank. If object o is not indexed by r, then both $score(r, o)$ and $rank(r, o)$ are undefined for this object. If $DB(r)$ consist of all the objects in $\mathcal{O}$, then it is referred to as a *full list*; otherwise we call it a *partial list*. We will also refer to partial lists as rankers for uniformity. We will denote the partial list consisting of the top-K ranked objects of ranker r with $[r]_K$.

Given as input, a set of rankers $\{r_1, \ldots, r_s\}$, the aggregate is ranker r_A of objects in $DB(r_1) \cup \ldots \cup DB(r_s)$ based on their ranks or scores in $\{r_1, \ldots, r_s\}$. The *rank aggregation problem* is finding an aggregate ranker that optimizes an error or performance measure, which we discuss in Section 3.2. The rank aggregation can be performed on the rank values or scores based on the availability and reliability of either type of information. The choice of using the scores or ranks can impact the quality of the aggregation when constructing the ranker. When ranks are used instead of scores, some information is lost due to the difference between scores that each rank represents. Suppose the score distributions for each ranker follow a zipf distribution. In this case, we expect a sharp decrease in the scores, but the ranks will not reflect this information. As a result, using rank values in aggregation results in a harder aggregation problem due to the amount of information loss. In applications like meta-search engines (that merges the search results of search engines), scores are rarely available. It is possible to incorporate additional information about the objects at the time of aggregation. The use of scores and ranks together is outside the scope of this thesis. In other applications, e.g. voting, there are only ranks or positional information available for processing.

3.2 Performance measures

To evaluate rank aggregation methods, we will use a performance measure that assesses the degree of closeness or similarity between two rankers, r_1 and r_2.

In one type of error measure, both rankers are treated symmetrically, as in the case of Kendall-tau measure. In other performance measures, which are inherited from the traditional information retrieval systems, one ranker is used as a reference point to model the "correct" ranker. The recall and precision performance measures are examples of this. We will refer to the reference ranker as the *ground truth*. Given that rankers may not rank the same objects, performance measures must be able to cope with this problem as well. For each performance measure, we assume each ranker is a partial list of objects. The first two measures listed below are measures of performance, but we will refer to all of them as performance measures for uniformity. We will indicate whether we are attempting to minimize or maximize each measure where appropriate.

3.2.1 Recall

Recall is typically defined in the prior literature as the proportion of relevant documents that are retrieved. Hence given r_1 and r_2, we can define *recall* as the lowest j such that $[r_2]_K$ contains all the objects in r_1, where $recall(r_1, r_2) = max\{j | \forall o \in r_1, j = rank(r_2, o)\}$. This assumes that ranker r_1 is the correct ranking or the ground truth. In order to effectively measure *recall*, we assume that ranker r_2 is a full list with respect to r_1, where $DB(r_1) \cup DB(r_2) = DB(r_1)$. Otherwise, the object is undefined in r_2 and hence can not compute the j value.

3.2.2 Precision and TREC-style average precision

These measures also assume r_1 to be the reference ranker or ground truth. *Precision* $pr(r_1, r_2)$ gives the number of common objects in both rankers. Assuming we have $[r_1]_K$ and $[r_2]_K$, a frequently used variation of precision is the *TREC-style average precision (TSAP)*, which is given by

$$tsap(r_1, r_2) = \frac{\sum_i rel_i}{K}$$

where $rel_i = 1/i$ if the i^{th} object in r_1 is in r_1 and $rel_i = 0$ otherwise. The TSAP measure takes into account not only the number of relevant objects, but also where they occur. The precision and TSAP performance measures can be computed on

either full or partial lists. The precision and TSAP performance measures disregards the objects in the ranker's database that do not appear in both rankers, implicitly labeling them as irrelevant.

3.2.3 Spearman's rho footrule and Kendall-tau

The definition of the distance measure, Spearman footrule, is as follows [50, 51]. *Spearman footrule* is the sum of the difference of ranks over all objects $o \in U_A$ in both rankers. Mathematically, we have $F(r_1, r_2) = \sum_{o \in DB(r_1) \cup DB(r_2)} |rank(r_1, o) - rank(r_2, o)|$. The footrule error measure assumes that an object o is assigned a rank in rankers r_1 and r_2. In the case of partial lists, if an object o is undefined in one ranker, then a default rank is imposed.

Another performance measure is Kendall-tau $\tau(r_1, r_2)$, which is the total number of pairwise disagreements between rankers r_1 and r_2 is given by

$$\tau(r_1, r_2) = \sum_{o_1, o_2 \in r_1, r_2} \overline{K}^{(p)}_{o_i, o_2}(r_1, r_2)$$

The disagreements are counted as follows:

$$\overline{K}^{(p)}_{o_1, o_2} = 1 \textbf{ if } o_1, o_2 \in r_1 \text{ and } o_1, o_2 \in r_2$$
$$\textbf{then } rank(r_2, o_1) < rank(r_1, o_2) \textbf{ and } rank(r_2, o_1) > rank(r_2, o_2)$$
$$(\text{or } rank(r_1, o_1) > rank(r_1, o_2) \text{ but } rank(r_2, o_1) < rank(r_2, o_2))$$

where a penalty p is used in case of missing objects. The penalties p_1 and p_2 are defined specifically for certain cases of missing objects. Kendall-tau is sensitive to sortedness of the objects. It computes the bubble-sort distance, which is the total number of flips required to sort one ranker to make it identical to the other. In Table 3.1, the Kendall-tau disagreement penalty for different cases.

For the cases where objects are missing, a penalty function is introduced based on the assumptions made about the system. There are four cases to consider: In *Case 1*, o_1 and o_2 appear in both rankers as explained above. *Case 2* states that o_1 and o_2 both appear in one ranker (r_1), and only one of o_1 or o_2 appears in the other ranker (r_2). Suppose now $rank(r_1, o_1) < rank(r_1, o_2)$. If o_2 appears in the other ranker but not o_1, then there will be a disagreement if object o_2 exists in ranker r_2.

	Order Preserved	Pairwise Disagreement
Case 1	$\overline{K}^{(p)}_{o_1,o_2}((o_1,o_2) \in r_1, r_2) = 0$	$\overline{K}^{(p)}_{o_1,o_2}((o_1,o_2) \in r_1, (o_2,o_1) \in r_2) = 1$
Case 2	$\overline{K}^{(p)}_{o_1,o_2}((o_1,o_2) \in r_1, o_1 \in r_2) = 0$	$\overline{K}^{(p)}_{o_1,o_2}((o_1,o_2) \in r_1, o_2 \in r_2) = 1$
Case 3	N/A	$\overline{K}^{(p)}_{o_1,o_2}(o_1 \in r_1, o_2 \in r_2) = p_1$
Case 4	N/A	$\overline{K}^{(p)}_{o_1,o_2}((o_1,o_2) \in r_1, (o_1,o_2) \notin r_2) = p_2$

Table 3.1: Kendall-tau disagreement penalties

The penalty in this case is set to p_1 which denotes the probability of o_2 appearing in the other ranker. *Case 3* suggests that o_1, but not o_2, appears in a ranker (r_1), and o_2 but not o_1, appears in the other ranker (r_2) (or the reverse). Again, there will be a pairwise disagreement if o_2 and o_1 will appear in the respective rankers and the penalty is set to p_1.

Lastly in *Case 4*, o_1 and o_2 both appear in one ranker (r_1), but neither appear in the other ranker (r_2). In this case, a penalty p_2 is assigned which corresponds to the number of times o_1 and o_2 can be expected to be in the same order in the other ranker as well. If $p_2 = 1/2$, then we use the approximate case assumption, in which the order is not always accurate. If $p_2 = 0$, then we are assuming that the objects are in the correct order so no penalty is imposed. If we assume that all the rankers rank the same objects, then missing ranks can only be larger than existing ranks. It is then possible to get $p_1 = 1$. In the case when both objects in one ranker are missing their relative ranks, there are two options. An average case is to set $p_2 = 0.5$. For a more optimistic view, we can set $p_2 = 0$. This allows us to only count actual disagreements.

Note that these measures are not normalized, i.e. measures for rankers of different lengths are not immediately comparable to each other. The Kendall-tau and footrule performance measures require more calculations as the number of objects observed increase. While normalization is possible, as shown in Fagin et al. [55], this may introduce a bias against the missing objects. We chose to use the non-normalized version of Kendall-tau.

CHAPTER 4
RANK BASED ALGORITHMS

In this chapter, we first describe several well-known aggregation methods, which serve as base methods in the analysis provided in this thesis. The rank aggregation methods take as function a number of top-K rankers and return as output a top-K aggregate ranker. The average and median aggregation methods have been use frequently in the literature (such as [56, 86, 104]) as they provide meaningful representations of the rankers and are easy to compute. The PageRank aggregator, a form of Markov chain aggregator [53], is slightly more costly to implement based on a graph representation of the rankers. We then present an aggregator called precision optimal (PrOpt), although relatively simple, has not been discussed in prior literature to the best of our knowledge. As we will see, it also provides a meaningful representation of the rankers with low computational cost. We also present a novel aggregation method based on optimizing the Kendall-tau performance measure called iterative best flip (IBF). We also discuss another Kendall-tau optimization method called adjacent pairs (ADJ) that was discussed in previous literature.

4.1 Background

Given a set of rankers $\{r_1, \ldots, r_s\}$, and an aggregate ranker r_A, our objective is to reduce the average error between r_A and $\{r_1, \ldots, r_s\}$ where we define the average error to be $\mathcal{E}_{av} = \frac{1}{s}\sum_{i=1}^{s}\mathcal{E}(r_i, r_A)$ where $\mathcal{E}(r_i, r_A)$ could be any one of the error measures discussed previously. We consider two aggregators which attempt to optimize the Kendall-tau error in Section 4.7. One based on optimizing using adjacent flips (ADJ), and a new optimizer based on an iterative best flip (IBF) approach. With any rank aggregation method, ties among the objects may exist. The choice of how to break ties can have an impact on the quality of the aggregation method. Unless otherwise indicated, ties are broken randomly.

4.2 Average (Av) and Median (Me)

The *average* and *median* aggregators are defined in the usual way. The average (or median) value of all the ranks of an object in the set of rankers are used to obtain the aggregate ranker of the objects. If an object is undefined because it is not in the top-K of the ranker, then we assign a default rank of $K+1$ for these aggregators assuming that the object is in the database of the given ranker and hence will have a lower rank.

4.3 CombMNZ

The prior work of Fox and Shaw [60] and Lee [86] present the CombMNZ algorithm, initially using scores [60] and later using ranks [86]. This algorithm orders objects based on the frequency of appearance in the input rankers and ranks. More formally, we implement CombMNZ as follows. Let $DB(r)$ be the set of objects that appear in an ranker r and $|DB(r)|$ be the number of objects. We define aggregate ranker r_A as the sorted list of objects from the rank aggregation methods. We now denote $DB(r_A) = DB(r_1) \cup \ldots \cup DB(r_s)$ for s rankers. Given rankers $\{r_1, \ldots, r_s\}$, we perform Borda rank normalization (brn) for each object $o \in DB(r_A)$, as presented in Renda et al. [104].

$$brn^i(o) = \begin{cases} 1 - \frac{rank(r_i,o)-1}{|DB(r_A)|} & \text{if } o \in r_i \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Now we can compute the aggregate score $sc(o)$ for each object o. We denote $h(\{r_1, \ldots, r_s\}, o)$ as the number of times object o appears in the rankers (in range [1,s]). The set of aggregate scores are then sorted in decreasing order.

$$CombMNZ : sc(o) = h(\{r_1, \ldots, r_s\}, o) * \sum_{i=1}^{s} brn^i(o) \tag{4.2}$$

4.4 PageRank (Pg) – A Markov Chain Aggregator

The Markov chain aggregator MC_4 in Dwork et al. [53] constructs the aggregate ranker by means of a Markov chain in which the transition from an object in MC_4 occurs if the destination object is ranked higher for the majority of the

input rankers. The steady-state of the Markov chain corresponds to the rankings of the aggregate ranker. This algorithm is introduced as a way to approximate an aggregation that reduces the total Kendall-tau error between the aggregate and the input rankers. Since it is not described explicitly how sink nodes are handled in this work, we use the PageRank algorithm [28] as an approximation to MC_4. The transitions are identical to MC_4 but the random jump probability removes the sink node. Given a set of rankers, the PageRank algorithm proceeds as follows. Each distinct object in $(DB(r_1) \cup \ldots \cup DB(r_s))$ represents a node in the graph $G = (V, E)$. A directed edge from object o_i to object o_j is introduced in E which ranks o_i above o_j including the implied ranks. The link is given weight $w(o_i, o_j)$ that is proportional to the difference of ranks it represents. These weighted edges are added to the graph for each object-pair in the input rankers. This means that our graph G contains parallel edges with possibly different weights. The weights are then normalized so that outgoing edges have total weight of 1 for each node. The pagerank of an object is given by a combination of the probability of navigating to that object from another or by randomly jumping to that object. The pagerank computation reduces to the problem of computing the steady state probability of a Markov chain with the same transition probabilities. The pagerank $Pg(o_i)$ of an object o_i is given by

$$Pg(o_i) = (1 - \alpha)p_i + \alpha * \sum_{(o_j, o_i) \in E} \frac{Pg(o_j) * w(o_i, o_j)}{outdeg(o_j)}$$

where *outdeg* is the outdegree of a node. The probability of randomly jumping to a site is proportional to the indegree of that node where $p_i = \frac{indeg(o_i)}{\sum_{o_j \in V} indeg(o_j)}$. This measure approximates the ranking produced by the average rank aggregator.

4.5 Precision Optimal Aggregation (PrOpt)

We compute the aggregate ranker by ranking objects by the number of times they appear in the top-K for a set of rankers. The top-K objects are selected from this aggregate. If there are ties for any rank including the K^{th} rank, then we break ties with respect to the order imposed by the average aggregator. If there are still ties, then we break them randomly. A general model of the aggregation method is

presented below.

1: **function** PrecisionOptimal(rankers: $\{r_1, \ldots, r_s\}$, retrieval size K) **returns** ranker: r_A
2: $position \leftarrow 1$
3: **for** each object $o \in \mathcal{O}$ **do**
4: **for** each ranker r_j **do**
5: **if** $rank(r_j, o) ==$ undefined **then**
6: $rank(r_j, o) \leftarrow K + 1$
7: **else**
8: $freq(o) \leftarrow freq(o) + 1$ //$freq(\cdot)$ = frequency count of object o in rankers
9: $sort(freq(\cdot))$ //descending order
10: **while** $position < |\mathcal{O}|$ **do**
11: let $\mathbf{F}$ be a vector of all objects with same frequency count
12: **if** $|\mathbf{F}| == 1$ **then**
13: $r_A(position) \leftarrow \mathbf{F}$
14: **else**
15: $sort(avgrank(\mathbf{F}))$ //ascending order
16: let $\mathbf{A}$ be vector of objects with same average rank in $\mathbf{F}$
17: **if** $|\mathbf{A}| == 1$ **then**
18: $r_A(position) \leftarrow \mathbf{A}$
19: **else**
20: randomly pick an object $\mathbf{A}$ from $\mathbf{F}$
21: $r_A \leftarrow \mathbf{A}$
22: increment $position$
23: **return** r_A

4.6 Condorcet-fuse

The Condorcet-fuse algorithm [97] constructs a graph $G = (V, E)$ where $V = DB(r_A)$ and $e(o_i, o_j) \in E$ is an unweighted directed edge where $o_i \rightarrow o_j$ (or $o_j \rightarrow o_i$) indicates that o_i (o_j) dominated o_j (o_i) in the majority of rankers. Thus, for every pair of objects, there is at most one edge. The objects are then sorted in such a

way to remain consistent with the directed edges. Therefore, the objects are listed such that all edges are forward edges. In the case of ties, the objects are randomly ordered. More formally, we show the algorithm below.

```
1: function cfuse(graph G) returns ranker: r_A
2: for all pairs of objects o_1 and o_2 in V do
3:    count = 0
4:    for each ranker r_i do
5:       if r_i ranks o_1 above o_2 then
6:          count++
7:       if r_i ranks o_2 above o_1 then
8:          count--
9:       if count > 0 then
10:         rank o_1 above o_2 (create an edge e(o_1, o_2))
11:      else
12:         rank o_2 above o_1 (create an edge e(o_2, o_1))
13: R^A = empty
14: for each object o ∈ V do
15:    InsertionSort(r_A, o)
16: output r_A
```

4.7 Kendall-tau Optimal Aggregators

In this section, we present methods to reduce the Kendall-tau error between the rankers and aggregate ranker. Finding a ranker that is optimal with respect to this performance measure is known to be an NP-complete problem [14]. To compute it approximately, we implement two heuristic optimization techniques. Each method takes as input an initial aggregation and then tries to reduce the Kendall-tau error through a localized or global search method.

4.7.1 Adjacent Pairs (ADJ)

A ranker is considered locally Kemenized if there does not exist any pair of objects o_i and o_{i+1} with adjacent ranks such that when these objects are flipped the

overall Kendall-tau error will be reduced. An algorithm to find a locally Kemenized ranker was proposed in Dwork et al. [53]. The authors also show how their algorithm can be implemented approximately in time $O(n\ log\ n)$. The general approach is to take a ranker and then perform a sequence of swaps between adjacent objects that would lead to a reduction in the Kendall-tau. We will call this algorithm adjacent pairs (ADJ) optimization. To compute it, we begin with an initial ranker, using one of the elementary aggregation methods discussed above, and iterate through each object checking if an adjacent swap will further minimize the Kendall-tau. The algorithm repeatedly checks the aggregator for adjacent swaps until no further reduction can be obtained. For completeness, we include a description of this algorithm.

1: **function** adjPairs(rankers: $\{r_1, \ldots, r_s\}$, aggregate ranker: r_A)
returns ranker: r_A
2: **for** i=1 to n **do**
3: Let o_1 be ranked such that $rank(r_A, o_1) = i$
4: Let o_2 be ranked such that $rank(r_A, o_2) = i + 1$
5: swap o_1 with o_2 in r_A
6: compute $\mathcal{E}_{av}$ after the swap;
7: **if** $\mathcal{E}_{av}$ reduced **then**
8: permanently swap objects
9: repeat for-loop until no further reductions can be performed
10: **return** r_A with minimum $\mathcal{E}_{av}$;

4.7.2 Iterative Best Flip (IBF)

In this section, we describe a novel Kendall-tau optimization method. The algorithm performs local combinatorial optimization, originally introduced by Kernighan and Lin [78] in the specific context of graph partitioning. The general idea behind our algorithm is to perform a sequence of greedy swaps that eventually leads to a good local minimum of the average error. A key feature is that greedy swap is performed even if the error increases. In this way, the algorithm has a limited amount of look ahead. We begin an initial ranking, which could be one of the other aggregators. The algorithm proceeds as follows.

1: **function** bestFlip(rankers: $\{r_1, \ldots, r_s\}$, aggregate ranker: r_A) **returns** ranker: r_A
2: **repeat**
3: $r_{old} \leftarrow r_A$, $Config \leftarrow \langle r_A \rangle$, $finished \leftarrow false$;
4: **for** each object $o_i \in r_A$ **do**
5: **for** every object $o_j \in r_A$ and $(o_i \neq o_j)$ **do**
6: compute $\mathcal{E}_{av}$ after the swap;
7: perform the swap with minimum $\mathcal{E}_{av}$ in r_A; {$\mathcal{E}_{av}$ may increase as a result}
8: add r_A to $Config$
9: Let r_{new} be the ranking in $Config$ with the minimum $\mathcal{E}_{av}$;
10: **if** r_{new} has smaller error than r_{old} or r_{new} has the same error as r_{old} but is a new configuration **then**
11: $r_A \leftarrow r_{new}$
12: **else**
13: $finished \leftarrow true$
14: **until** finished
15: **return** r_A

Note that the algorithm is *forced* to make a swap when considering each object sequentially (according to some arbitrary ordering). The best swap is made even if this leads to a temporary increase in the average error. It is exactly this flexibility which has been found to help the algorithm escape from bad local minima. The algorithm above is repeatedly executed, each time starting from its own output until no further progress is made meaning the ranker no longer changes. We also add an additional step where we check if the error does not change after an iteration of the outer for loop but a new ranker that has not been seen yet has been found (in line 10). In this case, we continue iterating. A straightforward implementation which computes the average error after each swap would have computational complexity $O(s \cdot f(n) \cdot n^2)$ where $f(n)$ is the cost of computing the average error for an aggregate ranker where n is the number of objects. We are using the Kendall-tau performance measure, for which $f(n) = O(n^2)$. By performing a pre-processing step which allows us to update the average error, instead of recomputing it from scratch, each time a

swap is made, one can improve the computational complexity to $O(n^3)$).

Computational Complexity of IBF. We now describe how to reduce the computational complexity of the IBF algorithm from $O(s \cdot f(n) \cdot n^2)$ to $O(n^3)$. To efficiently implement the IBF optimization, we perform a pre-processing step that stores the Kendall-tau changes for every pair of objects. We then use this information to update the Kendall-tau changes incrementally for each possible flip.

We present the algorithm for this stage below. We construct a lookup table of each object and its rank in every ranker as described in Lines 4-7. Then, for every pair of distinct objects o_i and o_j, we compute the number of rankers that rank o_i above o_j and o_j above o_i. In this computation, we do not consider if the ranks are actual or estimated to be $K+1$. In Lines 18-19 and 21-22, if $p_1 = 1$, then the order of the objects is assumed to be incorrect and the inversion cost is 1. In the case of $p_1 = 0$, we assume there is no inversion , which is a mistake and Kendall-tau should be reduced by 1 for every occurrence. The correct Kendall-tau counts are calculated for the pair of objects, where $KTM(o_i, o_j)$ refers to the change in Kendall-tau count when o_i is ranked above o_j and $KTM(o_j, o_i)$ refers to the change in count when o_j is ranked above o_i.

1: **function** InversionPenalty(rankers: $\{r_1, \ldots, r_s\}$, aggregate ranker: r_A, penalty: p_1) **returns** $n \cdot n$ matrix: KTM
2: Let RM be an $n \cdot s$ matrix containing object ranks in each ranker
3: Let KTM be an $n \cdot n$ matrix where each entry $KTM(o_i, o_j) = \overline{K}(o_i, o_j)$
4: **for** each object $o \in \mathcal{O}$ **do**
5: **for** each ranker r_j **do**
6: **if** *rank(r_j,o)* $> K$ **then**
7: *rank(r_j,o)* $\leftarrow K+1$
8: add $rank(r_j, o)$ to RM(o,:)
9: **for** $o_i \in r_A$ **do**
10: **for** $o_j \in r_A$ **do**
11: **if** o_i != o_j **then**
12: $\mathbf{v}_i \leftarrow RM(o_i, :)$, $\mathbf{v}_j \leftarrow RM(o_j, :)$

13: $\mathbf{v}_{ij} \leftarrow \mathbf{v}_i - \mathbf{v}_j$, $\mathbf{v}_{ji} \leftarrow \mathbf{v}_j - \mathbf{v}_i$

14: $x_{ij} \leftarrow |rank(r, o_i) < rank(r, o_j)|$ s.t.
$\forall rank(r, o_i) \leq K, \forall rank(r, o_j) \leq K$ in v_{ij}

15: $x_{ji} \leftarrow |rank(r, o_i) > rank(r, o_j)|$ s.t.
$\forall rank(r, o_i) \leq K, \forall rank(r, o_j) \leq K$ in v_{ji}

16: $y_{ij} \leftarrow |rank(r, o_i) < rank(r, o_j)|$ s.t.
$\forall rank(r, o_i) \leq K, \forall rank(r, o_j) = K + 1$ in v_{ij}

17: $y_{ji} \leftarrow |rank(r, o_i) > rank(r, o_j)|$ s.t.
$\forall rank(r, o_i) = K + 1, \forall rank(r, o_j) \leq K$ in v_{ji}

18: **if** $\mathbf{v}_{ij}$ has values $> K$ **then**

19: $x_{ij} \leftarrow x_{ij} - (1 - p_1) * y_{ij}$

20: $x_{ji} \leftarrow x_{ji} - (1 - p_1) * y_{ji}$

21: **if** $\mathbf{v}_{ji}$ has values $> K$ **then**

22: $x_{ij} \leftarrow x_{ij} - (1 - p_1) * y_{ij}$

23: $x_{ji} \leftarrow x_{ji} - (1 - p_1) * y_{ji}$

24: $KTM(o_i, o_j) \leftarrow x_{ij}$, $KTM(o_j, o_i) \leftarrow x_{ji}$

25: **return** KTM

Using the KTM and RM, we can apply a shortcut to compute the Kendall-tau error for the IBF optimization algorithm. The change of the Kendall-tau for any pair of objects is determined efficiently by reducing the number of function calls to the Kendall-tau computation. We now describe the method to update the Kendall-tau in flipping two objects using the incremental Kendall-tau computation algorithm below. The incremental algorithm counts the change in the number of inversions from the initial aggregate ranker to the new aggregate ranker, in which the new aggregate ranker r_A is a swap of two objects from the initial aggregate ranker. The pairs of objects affected lie between two positions e.g. $pos_i = rank(r, o_i)$ and $pos_j = rank(r, o_j)$. Two for-loops are performed that counts the change in the number of inversions if we flip objects o_i and o_j. In line 6 and 9, *kcount* counts for the inversion penalty for all the objects between pos_i and pos_j since the Kendall-tau error will change. In the first for-loop, we consider the effect of flipping object o_i while in the second for-loop, we address the change in Kendall-tau error of flipping

object o_j.

1: **function** IncrementalKendall(objects matrix:KTM, aggregate ranker: r_A, $rank(r, o_i)$: pos_i, $rank(r, o_j)$: pos_j) **returns** integer: $kcount$
2: $kcount \leftarrow 0$
3: determine highest rank between pos_i and pos_j, let's assume $pos_i < pos_j$
4: **for** $z = pos_i + 1$ to pos_j **do**
5: let o_z be the object at position z
6: $kcount \leftarrow kcount + (KTM(o_z, o_i) - KTM(o_i, o_z))$
7: **for** $z = pos_j - 1$ to $pos_i + 1$ **do**
8: let o_z be the object at position z
9: $kcount \leftarrow kcount + (KTM(o_j, o_z) - KTM(o_z, o_j))$
10: **return** $kcount$

4.8 Information vs. robustness trade-off

Now, we will elaborate on the properties of the above rank aggregation methods. In general, an aggregate ranker is considered a "complex" ranker if it adapts its final ranking to the subtle nuances in the input rankers. Thus, necessarily, such a complex aggregator will easily be misled by noise in the data – it is too sensitive to small fluctuations (inconsistencies) in the data, and as a result its performance rapidly degrades as such inaccuracies appear in the data. Conversely, consider the other extreme, an aggregator which considers little or none of the information contained in the rankers – for example an aggregator which completely ignores the input rankers and outputs a constant ranking. Such an ignorant ranker will have a poor performance, however its performance will not degrade as inaccuracies appear in the input rankers. Such a ranker uses less information, however it is *robust*. We consider an aggregator that uses less of the information contained in the input rankers as a "simple" aggregator even though a simple aggregator may be hard (computationally complex) to construct.

One of the simplest aggregator we introduce is the *precision optimal aggregator (PrOpt)* that disregards all information regarding the actual ranks except for the number of times an object appears in the input rankers. However, when the input

rankers contain almost the same objects, then the ranks produced by PrOpt are very similar to the output of the average aggregation method which is used for breaking the ties. The *median* aggregator disregards a specific type of information since it throws away all rank information for an object except for the middle one. Hence, it is not affected by changes in the actual rank values of outliers. The *average* aggregator is one of the most complex methods in our tests since it includes all the rank values in the computation. Note that neither median nor average explicitly take into account the number of times an object appears in the input rankers.

The question is then whether optimizing for Kendall-tau introduces more or less information about the input rankers. One of the best known voting paradoxes as discussed by Saari [107] shows that when aggregating votes to find the optimal ranking of candidates, the winner of pairwise elections may not be the winner of the plurality vote. We examine what this means for the Kendall-tau optimal aggregator. In the table below, we show an example of two rankers, r_1 and r_2, for three objects. As we can see in the table, all three orderings given are Kendall-tau optimal with

r_1	r_2	average	Kendall-tau optimal
o_1	o_3	o_1	o_3 | o_1 | o_1
o_2	o_1	o_3	o_1 | o_2 | o_3
o_3	o_2	o_2	o_2 | o_3 | o_2

respect to the rankers r_1, r_2, while only one of them is corresponds to the unique average aggregator. So, the Kendall-tau optimal ranking that uses only pairwise comparisons seems to *ignore* some information about the rankers which the average ranker uses. To see why, note that for r_1, when comparing o_3 to o_2 and o_1, we do not take into account the fact that o_2 is ranked below o_1 and hence o_3 is ranked third. Thus in the Kendall-tau, a flip is a flip, no matter how far apart the flipped objects are. Hence, it is insensitive to the location of the flips. On the other hand, the average takes into consideration the distance between flipped objects.

Given that Kendall-tau optimal ranking ignores some information, it leads to an aggregation method that is robust to noise. This was the main motivation behind the introduction of this optimization method in the literature [53]. The two optimization methods we study here *ADJ* and *IBF* as well as the PageRank (*Pg*)

methods are approximations of the Kendall-tau optimal ranking. As our tests show, the performance of these methods depends on the initial starting ranking from which the optimization proceeds. In the case of *Pg*, the average rank serves as the starting point. We evaluate all other methods using different starting rankings. Our results show that *IBF* is a far superior optimizer than *ADJ*, hence it produces a good approximation to the true Kendall-tau optimal aggregator (even if our optimizer starts from a random ranking). Therefore the *IBF* optimized version of any ranker depends less on the input rankers than the *ADJ* optimized version. Generally, we expect that both *Pg* and the *AvIBF* (IBF following average aggregator) contain less information about the location of flips than the average aggregator. However, any Kendall-tau optimization possibly leads to an aggregator that contains more information than *PrOpt* especially in noisy scenarios. If comparing *Me* with *MeIBF*, we can argue that *MeIBF* possibly contains less information about the middle ranker but more information about the other rankers than *Me*.

These aggregators incorporate different types of information to varying degrees. This provides us with a fairly extensive set of methods to test the information and robustness trade-off in aggregation methods and highlight when a specific aggregation method outperforms the others.

CHAPTER 5
APPROXIMATION ALGORITHMS

In this chapter, we introduce three graph based algorithms that can be used to solve a related problem, called *minimum feedback arc set (MFAS)*. It has been shown that finding the Kemeny optimal ranking [76, 77] reduces to the problem of solving the MFAS problem, which is NP-hard [6, 14]. Two algorithms, Greedy and SubIBF, initially construct biconnected components[1] of the objects in order to find (and remove) cycles, in addition to, trying to organize the objects with respect to the dominating order of every object pair. The third algorithm, CUT [106], recursively divides the objects into two subgraphs with the objective of minimizing the total edge weight across the subgraphs. The CUT algorithm terminates when each object is its own subgraph and a distinct ordering of object can be generated.

5.1 Minimum Feedback Arc Set (MFAS) Problem

We first formally define the MFAS problem and how the rank aggregation problem and the MFAS problem are related. We also discuss how to construct the aggregate ranking given a series of sub-graphs.

Minimum Weight Feedback Arc Set. Given a weighted directed graph $G = (V, E)$ where each pair of objects o_i and o_j, a weight $w(o_i, o_j)$ describes a weight of an edge from o_i to o_j. The minimum weight feedback arc set finds a set E' of edges such that (1) $G' = (V, E - E')$ is acyclic and (2) $\sum_{(o_i,o_j)\in E'} w(o_i, o_j)$ is minimized.

Rank aggregation to MFAS. The rank aggregation problem can be reduced the MFAS problem as follows. Given rankers $r_1, \ldots, r_s$ of length K each, we construct a graph $G = (V, E)$ such that V contains all the objects ranked by the input rankers. Given objects o_i, o_j, suppose l_i rankers rank o_i higher than o_j and l_j rank o_i lower than o_j. Note that $l_i + l_j \leq s$ as not all objects are ranked by all the rankers.

[1] Biconnected components in our case means strongly connected components, such that for all vertices u, v there is a path from u to v and a path from v to u.

If $l_i > l_j$ (i.e. majority of rankers rank o_i higher than o_j) then, we add an edge $(o_j \rightarrow o_i)$ with weight $l_i - l_j$. Otherwise, we add an edge $(o_j \rightarrow o_i)$ with weight $l_j - l_i$. Any missing ranks are assigned a default rank of $K+1$. Note that this graph is a special case of a TOURNAMENT where only one of $(o_i \rightarrow o_j)$ or $(o_j \rightarrow o_i)$ may exist. In our case, there may be no edges between o_i and o_j in case of a tie. This graph representation of the rank aggregation problem differs slightly from the one given in [4] which would assign weights l_i/s and l_j/s in each direction and disregard the one with the lower weight when solving the MFAS problem. As an acyclic graph is found as a solution to the MFAS problem, we can then use it to find an ordering among the initial objects.

Rank ordering of a graph. Given an acyclic graph G, the ordering (ranking) of the vertices induced by G is given by the topological ordering of the vertices and by breaking ties arbitrarily. The top node with only incoming edges and no outgoing edges is considered rank 1, the highest rank. Prior literature [45] examines the special case of feedback arc set problem to solve rank aggregation through ordering with respect to the weighted incoming edges.

5.2 Approximation Algorithms for MFAS

We introduce a series of algorithms using the strongly connected components of G. Suppose, G has exactly two biconnected components, G_1 and G_2. Then, all edges between these two components must be in one direction, suppose from G_2 to G_1 resulting in an acyclic hyper-graph. Otherwise, they would be part of the same biconnected component. The problem can then be reduced to solving the MFAS problem for each sub-graph. The biconnected components of graph G can be constructed in $\Theta(V + E)$ time [46], and partitions the graph into sub-graphs $Sub = \{G_1, G_2, \ldots, G_f\}$ where $|V_i| \geq 1$. We use the notation BCC(G) to refer to the biconnected components algorithm. As discussed above, if all the subgraphs are reduced acyclic graphs, then G will also be acyclic.

5.2.1 Greedy Algorithm (Greedy)

The greedy algorithm operates on biconnected components of the input graph. It picks a biconnected component at random, locates an edge with the minimum weight and removes the edge from the graph. The biconnected components are updated after each edge removal. The algorithm continues to run until all the biconnected components are of size 1 which is an acyclic graph.

1: **function** greedy(G) **returns** acyclic graph Sub
2: $Sub \leftarrow$ BCC(G)
3: **repeat**
4: Pick $G_i = (V_i, E_i)$ from Sub uniformly at random
5: **if** $|V_i| > 1$ **then**
6: find the cheapest edge in G_i and remove it
7: call this new sub-graph G^*
8: $Sub \leftarrow$ BCC(G^*) $\cup$ ($Sub - \{G_i\}$)
9: **until** for all $G_i = (V_i, E_i)$ in Sub and $|V_i| == 1$
10: **return** Sub

5.2.2 CUT-recursive Algorithm (CUT)

The cut partitioning algorithm first constructs a random cut by separating the vertices into two subgraphs with the Generator algorithm. Then, it tries to improve the cut continuously using the Improve and Scan algorithms. The aim is to maximize the total weight of forward edges from one cut to the other. The algorithm finds the maximal cut for different starting points and chooses the best one among those tried.

The CUT procedure. The Cut algorithm maximizes the edge weights from the right partition to the left partition. Once the edge weights are maximized, the edges in the opposite direction are reversed. The cost of the cut is then given by the total cost of the reversed edges. As the best cut is found, the algorithm then tries to partition the vertices in each subgraph recursively, reversing edges when necessary. The algorithm stops when each object is its own partition and returns the resulting

acyclic graph.

1: **function** CUT(G)
   returns acyclic graph G'_A, cost C
2: $Configs \leftarrow empty$
3: select a parameter T, e.g. $T \leftarrow ln(n)$
4: **for** i=1, i<T; i++ **do**
5: $G_1, G_2 \leftarrow$ GENERATOR_RANDOM(G)
6: $G_1^*, G_2^* \leftarrow$ IMPROVE(G_1, G_2)
7: $G'_1, C_1 \leftarrow$ CUT(G_1^*)
8: $G'_2, C_2 \leftarrow$ CUT(G_2^*)
9: $E' \leftarrow \{(o_i, o_j) \mid o_i \in V'_2, o_j \in V'_1, (o_i, o_j) \in E\}$
10: $Cost \leftarrow \sum_{(o_i,o_j)\in E'} w(o_i, o_j)$
11: $G'_A = (V, E'_1 \cup E'_2 \cup \{(o_j, o_i) \mid (o_i, o_j) \in E'\})$
12: add $(G'_A, Cost + C_1 + C_2)$ to $Configs$
13: **return** $(G'_A, Cost)$ with the minimum-cost from $Configs$

The Generator procedure. The first phase of the CUT algorithm is to initialize the two partitions by randomly selecting the set of objects in each partition.

1: **function** GENERATOR_RANDOM(G)
   returns graphs: G_1, G_2
2: **for** i=1 to n **do**
3: x = random()
4: **if** x < 1/2 **then**
5: put o_i in V_1
6: **else**
7: put o_i in V_2
8: Let G_1 be the graph induced by V_1
9: Let G_2 be the graph induced by V_2
10: **return** G_1, G_2

It is also possible to divide the nodes in the graph with respect to some in initial ordering as follows. If the initial ordering is given by some other aggregator x, we will call this version of the Cut algorithm xCUT.

1: **function** GENERATOR_ORDERED(G, r)
returns graphs: G_1, G_2
2: **for** i=1 to n **do**
3: **if** o_i is in the half of the ranking r **then**
4: put o_i in V_1
5: **else**
6: put o_i in V_2
7: Let G_1 be the graph induced by V_1
8: Let G_2 be the graph induced by V_2
9: **return** G_1, G_2

The Improve procedure. The second phase of the CUT algorithm is to iterate through the vertices in each partition and determine which partition it belongs. The aim of the Improve algorithm is to maximize the total weight of the edges across the partition or cut in any one direction. Given two partitions $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ of $G = (V, E)$, an edge within a partition is in E_1 or E_2. An edge *across* the partition is one in $E - (E_1 \cup E_2)$.

For each vertex, we compute the total weight of incoming and outgoing edges *within* a specific subgraph G_1 or G_2, denoted by in^w, out^w. Similarly, we compute the total weight of incoming and outgoing edges *across* from one subgraph G_1 (or G_2) to another subgraph G_2 (or G_1). For example, if $o_i \in V_1$, $in^w(i) = \sum\{w(o_j, o_i) \mid (o_j, o_i) \in E, o_j \in V_1\}$ and $in^a(i) = \sum\{w(o_j, o_i) \mid (o_j, o_i) \in E, j \in V_2\}$. The outgoing edges are computed similarly. We then try to move one vertex o_i from one partition to another to try to improve $out^a(i)$ for that vertex. We switch each vertex that reduces the cost once resulting in a new configuration and then choose the configuration with the lowest cost among all tried. This process is repeated until the total weight of edges from one to the other no longer improves.

1: **function** Improve(graphs: G_1, G_2)

returns graphs: G_1, G_2
$L \leftarrow 0$, $R \leftarrow 0$, $width \leftarrow 0$
for each vertex o_i **do**
  compute $in^w(i)$, $in^a(i)$, $out^w(i)$, $out^a(i)$
  if $o_i \in V_1$ **then**
    $L \leftarrow L + out^w(i)$
  else
    $R \leftarrow R + out^w(i)$
if $R < L$ **then**
  $swap(G_1, G_2)$ //renames G_1 and G_2
  width $\leftarrow R$
else
  width$\leftarrow L$
while *width* changes **do**
  $G_1, G_2, width \leftarrow$ SCAN$(G_1, G_2, width)$
return G_1, G_2

function Scan(graphs $G_1, G_2, width$)
returns graphs $G_1, G_2, width$
partitions $\leftarrow$ *empty*
for each vertex o_i **do**
  hot $\leftarrow o_i$
for j=1 to max$\{|V_1|, |V_2|\}$ **do**
  if G_1 has hot vertices **then**
    select a hot vertex $o_k \in G_1$
    if $out^a(k) < in^w(k)$ **then**
      Put o_k in G_2
    cold $\leftarrow o_k$
  if G_2 has hot vertices **then**
    find a hot vertex $o_k \in G_2$
    if $out^w(k) > in^a(k)$ **then**
      Put o_k ind G_1

15: cold $\leftarrow o_k$
16: update *width*
17: $p \leftarrow (G_1, G_2, width)$
18: add p to *partitions*
19: **return** $(G_1, G_2, width)$ with the minimal *width* from *partitions*

5.2.3 Sublist-IBF Algorithm (SubIBF)

Recall the IBF algorithm given in Section 4.7.2. This algorithm takes as input an initial aggregate ranking and a set of input rankers that is used to compute the error of a ranking. For the MFAS algorithm, the error function that uses the input graph is shown below. Let $o_i = findObject(r, i)$ be a function that locates the object o_i that appears in the i^{th} rank in some ranker r. For simplicity, we reuse the term $\mathcal{E}_{av}$, which we defined in Section 4.1.

1: **function** error(graph: G, aggregate ranker: r_A)
returns integer $\mathcal{E}_{av}$
2: $\mathcal{E}_{av} \leftarrow 0$
3: **for** $l = 1$ to $K - 1$ **do**
4: Find o_l such that $o_l = findObject(r_A, l)$
5: $\mathcal{E}_{av} \leftarrow \mathcal{E}_{av} + \sum_{(o_l, o_j) \in E} w(o_l, o_j)$
6: $E = E - (\{(o_l, o_j) \mid (o_l, o_j) \in E\})$
7: **return** $\mathcal{E}_{av}$

By substituting the above error function in the IBF function, we obtain a graph based version of the IBF algorithm. The Sublist-IBF (SubIBF) algorithm first finds biconnected components and then executes the IBF algorithm on each subgraph. We refer to IBF(G, r_A) as a modified version of the $bestFlip(\{r_1, \ldots, r_s\}, r_A)$ in which G is a graphical representation of the content in the input rankers $\{r_1, \ldots, r_s\}$ and minimizes the error between the input data and aggregate ranker r_a.

1: **function** SubIBF(graph: G, aggregate ranker: r_A)
returns graph G
2: $Sub \leftarrow$BCC(G)

3: **for** each $G_i = (V_i, E_i)$ in Sub **do**
4: **if** $|V_i| > 1$ **then**
5: Let r_{A_i} be the portion of r_A corresponding to vertices in G_i
6: $r'_{A_i} \leftarrow \text{IBF}(G_i, r_{A_i})$
7: Remove from G all edges that contradict with r'_{A_i}
8: **return** G

CHAPTER 6
STATISTICAL FRAMEWORK FOR AGGREGATION

In this chapter, we describe a statistical model that we use to evaluate the rank aggregation algorithms and their input rankers. When evaluating rank aggregation methods, it is very hard to develop objective test cases. Most often, relevance quality of result are given by an expert or user. The problem with this approach is that it is difficult to develop extensive tests to examine the relationship between the input rankers and the correct ranking. To address this problem, we introduce a statistical model where we first define a ground truth and then develop models of noise, misinformation and spam. We first describe the terminology and definition associated with the statistical model in which all data is known. Then we discuss the impact of missing objects and correlation with respect to the statistical framework.

6.1 Framework Basics

In our framework, we assume that there is a ground truth ranker r, which is a vector containing all n objects in sorted order. The ground truth rank of an object (a web page in the case of a search engine) is determined using a *score* computed from factors $\{f_1, \ldots, f_m\}$, where $f_\ell \in [-3, 3]$ for $\ell \in [1, m]$. Each factor f_ℓ measures some property of the object such as webpages pagerank[2], the number of occurrences of the query keywords in the object's text, the amount of time the page has been live, and the frequency of updates. To simplify notation, we collect $\{f_1, \ldots, f_m\}$ into the vector $\mathbf{f}$, and write $\mathbf{f}(o_i)$ for the factors of object o_i. The *score* (or value) of object o_i, denoted V_i, is a weighted linear combination of the factors,

$$V_i = \mathbf{wf}(o_i) = \sum_{\ell=1}^{m} w_\ell \cdot f_\ell(o_i).$$

The weight vector $\mathbf{w}$ determines the relative importance of the factors. A negative weight vector indicates that that particular factor is detrimental to the value. In

[2]Note that Pageank here is assumed to be computed against the Web graph to find the pagerank of a webpage. Our algorithm Pagerank computes aggregates with respect to input rankers

our experiments, we have set $\mathbf{w} > 0$ with $\sum \mathbf{w}_\ell = 1$. We collect all the scores in the vector $\mathbf{V}$ and define the factor matrix $\mathbf{F}$ in which the rows of $\mathbf{F}$ are the object factor vectors $\mathbf{f}$, i.e. $F_{i\ell} = f_\ell(o_i)$. Then, $\mathbf{V} = \mathbf{F}\mathbf{w}$.

For simplicity, we will assume that no two objects have the same score. The ground truth ranks $rank(r, o_i)$ are obtained by ordering the objects according to their scores. The top-K ground truth ranking $[r]_K$ is what we would like to estimate. The available data are the top-K' rankings of some other rankers ($K \leq K' \leq n$). Thus, there are other rankers $\{r_1, \ldots, r_s\}$ which are each somehow related to the ground truth ranker r (we use subscript to refer to rankers). Our statistical framework provides a natural probabilistic approach to model the relationship between ranker r_j and the ground truth r. Intuitively, each *input ranker* r_j is an approximation to r constructed as follows: the input ranker attempts to measure the same factors $\mathbf{f}$ which are relevant to the ground truth ranker. However, its measurements may incur some errors, so the factor matrix obtained by ranker j is written as $\mathbf{F}_j = \mathbf{F} + \boldsymbol{\epsilon}_j$. Ranker j may also not have the correct relative weights for the factors. Denoting ranker j's weights by $\mathbf{w}_j$, we have

$$\mathbf{V}_j = \mathbf{F}_j \mathbf{w}_j = (\mathbf{F} + \boldsymbol{\epsilon}_j)\mathbf{w}_j.$$

The ranking r_j is obtained by ordering objects according $\mathbf{V}_j$. The top-K' lists are the inputs to the aggregation algorithm. The ground truth ranker and the inputs to the aggregation algorithms are completely specified by $\mathbf{F}, \boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_s, \mathbf{w}, \mathbf{w}_1, \ldots, \mathbf{w}_s$. The statistical model is therefore completely specified by the joint probability distribution

$$P(\mathbf{F}, \boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_s, \mathbf{w}, \mathbf{w}_1, \ldots, \mathbf{w}_s).$$

Such a general model can take into account: correlations among factor values (correlations in $\mathbf{F}$); correlations between factor values and ranker errors (correlations between $\mathbf{F}$ and $\boldsymbol{\epsilon}_j$); correlations among ranker errors (correlations among the $\boldsymbol{\epsilon}_j$); correlations between true weights $\mathbf{w}$ and ranker weights $\mathbf{w}_j$ (the degree of similarity between rankers and the truth); correlations between ranker weights and the errors of different rankers. First, we set the true weights $\mathbf{w}$ to all equal $\frac{2}{m(m+1)}[1, 2, \ldots, m]$.

We consider two possibilities for the $\mathbf{w}_j$: $\mathbf{w}_j = \mathbf{w}$ and $\mathbf{w}_j = \mathbf{w}_r = \frac{2}{m(m+1)}[m, \ldots, 2, 1]$, which represents a biased treatment of the factors with respect to the true weights. We introduce correlations between the errors and the factors, so $\epsilon_{ji\ell}$ depends only on $F_{i\ell}$. More specifically, the variance $Var(\epsilon_{ji\ell})$ is a function of $F_{i\ell}$,

$$Var(\epsilon_{ji\ell}) = \sigma^2 \frac{(\gamma - F_{i\ell})^\delta \cdot (\gamma + F_{i\ell})^\beta}{\max_{f \in [-3,3]} (\gamma - f)^\delta \cdot (\gamma + f)^\beta}. \tag{6.1}$$

This functional dependence allows us to model spam by setting the variance in the error for negative valued factors to be large, which means they experience large errors that may propel them high into the rankings. The parameters γ, δ, β are shape parameters which determine how spam enters the rankings, and σ^2 is a parameter governing the maximum possible variance – how noisy the ranker factors are. The noise parameter σ^2 measures how much the true information is getting corrupted. The shape parameters γ, δ, β determine which information gets corrupted.

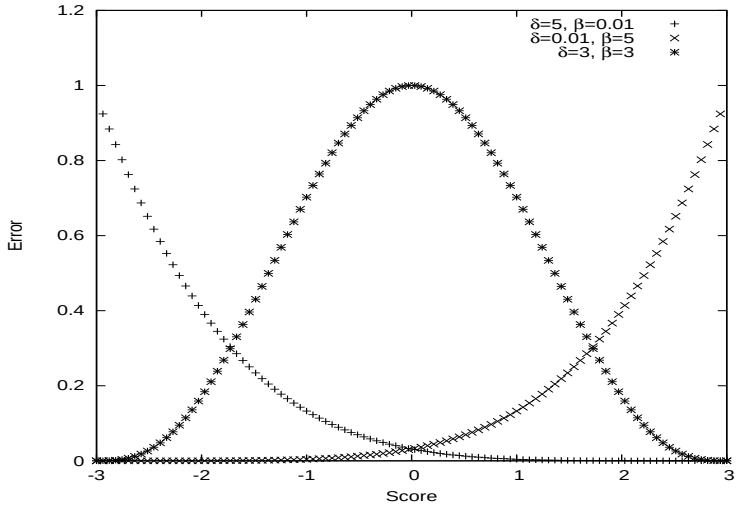


Figure 6.1: Possible shapes of the correlation between factors and the magnitude of errors. The x-axis displays the interval of the factor scores where a positive value indicates more relevant factor. The y-axis displays the amount of error being adding to the factor scores. The δ and β denote the degree in which these factor scores are corrupted.

To complete the model description, the factors for each object are chosen independently and identically from a uniform distribution with variance 1, and hence

lie approximately in the range $[-3, 3]$. The errors for each factor and each ranker are chosen independently from a uniform distribution with mean zero and variance given by the formula above for some choice of $\sigma^2, \gamma, \delta, \beta$. The input to the aggregation algorithm are the top-K' lists corresponding to each ranking. In our experiments we selected $K' = K$. Figure 6.1 shows three possible shapes of this function for $\gamma = 3$ and $\sigma^2 = 1$. Note that the rightmost values show the highest scores. When $\delta = \beta$, then the errors are lowest for objects with the highest and lowest values. When $\delta >> \beta$, then the rankers are more likely to make large mistakes for low valued objects which may propel them high into the rankings. This might be due to the adversarial techniques used by the engines to mislead the algorithms used by the rankers to estimate a specific factor. We use this last setting in our tests to model spam.

Let $\mathcal{A}$ generically refer to an aggregator, and let $\mathcal{E}([r]_K, [r_{\mathcal{A}}]_K)$ be a performance measure, such as Kendall-tau, that measures the difference between the ground truth ranker r and the ranker $r_{\mathcal{A}}$ obtained by the aggregator. In a realistic setting, $[r]_K$ is not known, however in our setting, $[r]_K$ is known. Thus, among the available aggregators, we can select the aggregator with the smallest average error through simulation within this statistical setting. The statistical framework can embed qualitative features of the aggregation setting through the choice of $P(\mathbf{F}, \boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_s, \mathbf{w}, \mathbf{w}_1, \ldots, \mathbf{w}_s)$; rigorous simulation can then be used to obtain the appropriate aggregator for that particular aggregation setting.

The complete list of the statistical framework parameters are displayed in Figure 6.2. In the case where factors, objects and rankers are not independent from each other, additional parameters can be used to simulate the correlation. The correlation parameters σ_f^*, σ_n^* and σ_{nf}^* impose a degree of dependence between all rankers $r, r_1, \ldots, r_s$. The relationship between these correlation parameters are as follows: $\sigma_{nf}^* < min(\sigma_n^*, \sigma_f^*)$ and $\sigma_{nf}^* > \sigma_n^* + \sigma_f^* - 1$. Valid correlations between factors and objects must maintain these dependencies otherwise there is an unfair bias in the factors or objects. The σ_f, σ_n and σ_{nf} parameters are used to impose correlations between specific pairs of rankers, $r_1, \ldots, r_s$ implying the situation when two rankers' factors and/or objects are highly similar to each other but dissimilar

Notation	Description
n	number of objects
m	number of factors
s	number of rankers
$\mathbf{w}$	true ranker's weights
$\mathbf{w}_j$	weights for ranker j
δ	shape parameter where large variance for positive factors near 3
β	shape parameter where large variance for negative factors near -3
σ^2	the maximum variance in the desired range
σ_f	correlation between two factors of the same object, < 1
σ_n	correlation between two objects for the same factor,< 1
σ_{nf}	correlation between two different objects for different factors, < 1
σ_f^*	correlation between two factors of the same object
σ_n^*	correlation between two objects for the same factor
σ_{nf}^*	correlation between two different objects for different factors
σ_R^*	correlation among rankers

Figure 6.2: Statistical framework symbols and definitions

to the remaining rankers' factors and/or objects. Note that σ_f, σ_n and σ_{nf} are applicable to both the true and input rankers while σ_f^*, σ_n^* and σ_{nf}^* can be assigned only to the input rankers. The σ_R^* parameter is a higher level of correlation that allows the input rankers to have some sense of correlation not specific to the factors or objects. We also include a parameter to change the original distribution of the ground truth scores which can be set to be uniform or normal, we use the uniform distribution in our current tests.

In preliminary experiments, we made several assumptions such as the ground truth ranker's ability to order all the objects in the universe based on a linear combination of factors and weights. The input rankers are therefore permutations of the ground truth ranker. Each object is assigned a rank in each ranker. The conclusions from our evaluation are based on a broad and somewhat unrealistic knowledge base. Assuming that all objects will ultimately be ranked, the default rank of $K + 1$ is assigned to these objects. There is also a slight bias in the Kendall-tau computation since the penalty values assume existing rankers are always correct in their ordering when ranks of two objects are missing in a ranker. It is also important to consider different distribution for scores, such as a heavy tailed distribution instead of our

current uniform distribution since there is data to suggest that different factors are likely to follow such a distribution [48]. We also assume that the factors, objects and rankers are independent from each other, but other correlations exist in real-world applications. The possibility of missing objects and correlations are discuss below.

6.2 Missing data

It is also possible to model missing objects by assuming with some probability p_m^i ranker i does not have a specific object in its database and does not rank this object at all. The ground truth ranker models an ordering of all the objects in the universe while the input rankers will only index and order a fraction of the objects, where each input ranker orders a different set of objects. We address the cases where the input rankers have not assigned ranks to some objects and the influence of missing information in the formulation of an aggregate ranker. We are interested in observing how the aggregators perform in the presence of missing information. As the amount of missing data increases, there is less useful rank information for all the rankers as an increasing number of ranks will be missing. We examine these effects under different ranker properties.

6.3 Correlated data

Similar to missing data, we impose varying constraints on the correlations between objects for the same factor (σ_n, σ_n^*), correlations between factors for the same objects (σ_f, σ_f^*) and correlations between input rankers (σ_R^*). We consider both positive and negative correlations, where a positive value denotes a dependence relationship, a negative value denotes that the degree for the lack of dependence and a value of 0.0 denotes that there is independence. In this section, we examine the real world motivating factors for different correlations.

The first type of correlation introduced is between the errors and the factors (defined by $\sigma^2, \gamma, \delta, \beta$), where $\epsilon_{i\ell}^j$ depends on $F_{i\ell}$. More specifically, we set the variance $Var(\epsilon_{i\ell}^j)$ to be a function of $F_{i\ell}$ as described in Equation 6.1. Figure 6.1 shows three possible shapes of this function for $\gamma = 3$ and $\sigma^2 = 1$. Note that the rightmost values show the highest scores. When $\delta = \beta$, then the errors are lowest

for objects with the highest and lowest values. When $\delta >> \beta$, then the rankers are more likely to make large mistakes for low valued objects which may propel them high into the rankings. This might be due to the adversarial techniques used by the engines to mislead the algorithms used by the rankers to estimate a specific factor. We use this last setting in our tests to model spam.

σ_f Correlation between two factors of the same object in the ground truth ranker.

This models whether two factors measure the same quantity or independent quantities. For example, the length of page and frequency of keywords would be correlated positively. Similarly, if we are counting the frequency of a keyword in title, text and the outgoing links (to measure a hub value for the page) than it is likely that there will be a correlation. However, this correlation can be negative as higher occurrence of a keyword in one field may imply a lower occurrence of another keyword for two texts of the same length.

σ_n Correlation between two objects for the same factor in the ground truth ranker.

This models the distribution of values for a single factor over all the objects. Hence, if the correlation is low, the objects take more or less random values. Otherwise, if there is a positive correlation, one object having low score may imply another having low score. So, if scores are very close to each other as a result, the problem becomes harder since the objects are indistinguishable from each other. For example, for very popular keywords, the distribution will be very dense meaning there will be lots of pages with the same frequency of keywords. However, for keywords that have political connotations, one can imagine a bipartisan situation. Certain objects from one party having a high occurrence may imply a low occurrence for the objects from the other party, leading to a negative correlation.

σ_{nf} Correlation between two different objects for different factors in the ground truth ranker.

This models when the value of an object for a factor may depend partially on the value of another object for another factor. It is possible to model this as

$\sigma_n * \sigma_f$. It makes little sense to have a non-zero value for σ_{nf} when the other two are zero. For example, suppose we consider two objects, o_1 and o_2, where one of the factors of o_1 is the pagerank of o_1 which links to o_2 and one of the factors of o_2 is the frequency of keywords in the anchor links for o_2. Since a search engine may include the keywords of the link from page o_1 in the content of o_2, then the frequency depends on the number of number of incoming links to o_1. But, the pagerank of o_1 depends on the number of its outgoing links and where the links lead to, which forms a cycle. So, there is a correlation between these two objects through different factors.

σ_f^* Correlation between the errors made by rankers for two factors of the same object (one set for each ranker).

The errors made by rankers for two different factors may depend on each other since rankers use similar algorithms for both. For example, frequency of keywords in anchor text and in regular text may be independent of each other. But, if the same algorithm for stemming and categorization is used then the algorithm makes similar errors in both. Another reason the errors of two factors may be correlated is when they depend on the underlying index of pages. For example, factors that use a statistical method for normalizing the scores will make errors that have dependence on each other. Finally, time dependent factors will make errors that depend on the time of measurement and the actual time a specific change was made.

σ_n^* Correlation between the errors made by rankers for two objects for the same factor (one set for each ranker).

This simply means that the algorithm makes similar mistakes for two objects. If the value of the factors for two objects are correlated, then this is a reasonable assumption. For example, two rankers may use the same database of web pages to compute the pagerank independently. Even though the computation may differ, the pagerank values depend on the underlying graph. If the pagerank of a page is underestimated due to missing edges, then the pagerank of all the pages that are pointed to this page will also be underestimated resulting

in a positive correlation.

σ^*_{nf} Correlation between the errors made by rankers for two different objects for different factors (one set for each ranker).

Suppose the ranker makes an error determining when a page was last updated and the last update time is a factor. Now suppose this page links to another page and the pagerank depends on when the links are added (i.e. pages accumulating links very quickly are demoted). Then, this would effect the pagerank of pages that it links to and make the errors correlated.

σ^*_R Correlation among rankers.

Basically this models the case of a ranker making errors that correlate with the errors another ranker makes. An easy example of this is the case where a ranker may use the output of another ranker. This is an explicit relationship in the case of a meta-search engine. There are also other implicit relationships where a ranker may rely on another partially for different query types such as directory lookups or sponsored links.

CHAPTER 7
EXPERIMENTAL EVALUATION

We study how the information and noise in the input rankers affects the performance of rank aggregation methods. We use five rankers, five factors and 100 objects. For the PageRank algorithm, we fixed the α[3] parameter at 0.85 since we did not observe any significant dependence on α (when $\alpha > 0$) in our experiments. The ground truth weights are set to $\mathbf{w} = \langle \frac{1}{15}, \frac{2}{15}, \frac{3}{15}, \frac{4}{15}, \frac{5}{15} \rangle$. We model spam in our model ($Var(\epsilon_{ji\ell})$) by setting $\delta = 5.0$ and $\beta = 0.01$. This results in smaller errors in factors with high scores and low rank, and larger errors in factors with very low scores. Hence, while good objects will have high scores, bad objects may also get high scores occasionally. We vary the variance parameter σ^2 between $0.1, 1, 5$ and 7.5 for all factors. Increasing the variance models more **noise**: higher values increase the likelihood of objects getting undeserved high scores.

We also vary **misinformation** by changing the weights used by n_{MI} of the rankers to $\mathbf{w}' = \langle \frac{5}{15}, \frac{4}{15}, \frac{3}{15}, \frac{2}{15}, \frac{1}{15} \rangle$. The remaining $5 - n_{MI}$ rankers have the same weights as the ground truth ranker. When $n_{MI} = 0$, there is no misinformation, all rankers have the same weights as the ground truth. As n_{MI} increases, the information about the input factors being transmitted by the rankers decreases. We call this an increase in misinformation. To see how this is different from noise, consider the case when we have infinite number of rankers. It is then possible that by averaging these rankers we are able to average out all the noise. However, the rankers with incorrect weights suffer from information loss that can never be recovered in this case. Misinformation models the case when the rankers use weights that differ from the user's preferences. For example, the user may not care about recency of updates to a page in determining the final ranking, but the rankers may. The noise models the case where rankers incorrectly estimate the score of a factor; this is the case in many text based spam methods which result in inflated scores for specific keywords. Other examples of noise are errors made in the pagerank computation

[3]The α parameter is the probability of navigating to a particular object from another object.

due to the incompleteness of the underlying web graph and errors in time based factors due to the frequency of crawls to a site.

Method	Description
Av	average
Me	median
CombMNZ	CombMNZ
Pg	PageRank
Cfuse	Condorcet-fuse
PrOpt	precision optimal
Rnd	random
Greedy	greedy
xIBF	iterative best flip opt. after algorithm x
xADJ	adjacent pairs opt. after algorithm x
xSubIBF	sublist IBF opt. after algorithm x
xCUT	cut partitioning after algorithm x

Table 7.1: Legend of algorithms

Given these two settings, we perform tests with and without the adjacent (ADJ) and iterative best flip (IBF) optimization resulting in three different versions of each aggregator. Each test is repeated for 40,000 datasets where each dataset contains its own ground truth ranker and five input rankers. For each performance measure, we compute the performance of the aggregation algorithms. Table 7.1 lists the aggregation methods used in our tests. We should note that the precision and TSAP errors are both to be maximized, whereas the Kendall-tau error is to be minimized.

We perform a pairwise comparison of the aggregation methods [4]. We use the notation xADJ to denote adjacent pairs optimization starting from aggregator x (and similarly for xIBF). For every pair of aggregation methods A_i, A_j, we calculate the difference $(A_i - A_j)$ of the performance measure values on each dataset. Based on the variance of these differences, we obtain a 99.9% confidence interval on the difference. If this confidence interval includes zero, then the two aggregators are incomparable (or equivalent). On the other hand, if the confidence interval is always

[4]For the rank based algorithms, there are 13 aggregators: Av, Me, Pg, PrOpt, CombMNZ, Cfuse,xADJ, xIBF, RndIBF. For the graph based algorithms, we add the greedy, xCUT and xSubIBF algorithms.

positive (resp. negative), then A_j is better (resp. worse) than A_i, written $A_j > A_i$ (resp. $A_j < A_i$). These ordering relations are shown in the graphs of Figures 7.2-7.6 (precision) and Figures 7.7- 7.11 (Kendall-tau). In each graph, an edge from aggregator A_i to A_j exists if A_i is a better aggregator than A_j for that performance measure. To reduce the complexity of the graph, we remove all edges that would be implied by transitivity.

7.1 Rank Based Algorithms

For the rank based algorithms given in Chapter 4, we perform three types of experiments. In the first experiment, there is no missing information and the factors, objects and input rankers are independent from each other. In the second, each input ranker has a certain percentage of missing objects are not ranked. The third set of experiments considers several correlations amongst the factors, objects or input rankers.

7.1.1 Baseline results

Figure 7.1 summarizes the findings for the precision error. We use the notation $*$IBF to denote IBF starting from any initial aggregator, and $x*$ to denote aggregator x with or without optimization. In each box, we list the best three aggregators. However, in most cases, the difference between the performance of the listed aggregators is very small. When the misinformation is low ($n_{MI} = 0$) and the noise is low ($\sigma^2 = 0.10$), almost all aggregation methods are equivalent. PrOpt reduces to Av due its tie breaking methodology. When misinformation is low, as the noise increases, there is a greater need for robustness. In this case PrOpt, Pg and IBF optimized rankers become the winners while the average and median aggregators do not produce an orderings that exploits the information provided.

When the noise is low, as misinformation increases ($n_{MI} = 1, 2$), median becomes the dominant aggregation method as it is not effected by the outliers. This is a "bi-partisan" case where the majority of the rankers are correct, but there are one or two outliers. In these cases, as noise increases, there is a greater need for robustness. In this case, MeIBF is the clear winner (Me in Figure 7.4(b)) while the

	$n_{MI} = 0$	$n_{MI} = 1$	$n_{MI} = 2$	$n_{MI} = 3$	$n_{MI} = 4$
$\sigma^2 = 7.5$	PrOpt, Pg*, CombMNZ MeIBF, RndIBF AvIBF	PrOpt, Pg*, *IBF, CombMNZ Cfuse AvADJ	PrOpt, Pg*, CombMNZ, *IBF Cfuse AvADJ	PrOpt, Pg, CombMNZ, PgADJ PgIBF MeIBF, RndIBF	Pg PrOpt, PgADJ, CombMNZ PgIBF
$\sigma^2 = 5.0$	PrOpt, CombMNZ Pg, PgADJ PgIBF	PrOpt, PgIBF, MeIBF RndIBF, CombMNZ, Pg, PgADJ, AvIBF Cfuse	MeIBF, RndIBF AvIBF, PgIBF PrOpt, CombMNZ	Av Pg AvADJ, PgADJ	Av AvADJ Pg
$\sigma^2 = 1.0$	PrOpt, Pg*, Av*, Me, CombMNZ MeADJ, Cfuse MeIBF	MeIBF, PgIBF, CombMNZ, Cfuse, PrOpt PgIBF	Me MeADJ Cfuse	Av Pg AvADJ	Av AvADJ Me, Pg
$\sigma^2 = 0.10$	Av PrOpt,*ADJ, *IBF, Cfuse, CombMNZ RndIBF	PgADJ, CombMNZ PrOpt, MeADJ, Cfuse, PgIBF, MeIBF Me	Me MeADJ PrOpt	Av Pg AvADJ	Av AvADJ RndIBF

(a) Summary of results for precision

	less misinformation				more misinformation
high noise	PrOpt, CombMNZ Pg PgADJ	PrOpt, CombMNZ Pg PgADJ	PrOpt, CombMNZ Pg, PgADJ, MeIBF RndIBF	Pg PrOpt, CombMNZ PgADJ	Pg PgADJ PrOpt, CombMNZ
	PrOpt, CombMNZ Pg PgADJ	PgADJ Pg, PgIBF, MeIBF, RndIBF AvIBF, CombMNZ	MeIBF, PgIBF AvIBF. RndIBF PgADJ	Pg PrOpt, CombMNZ Av	Av Pg AvADJ
	Pg Av CombMNZ	MeIBF RndIBF PgADJ	MeADJ Me MeIBF	Av Pg PrOpt	Av PrOpt CombMNZ
low noise	PrOpt, Av, Pg *ADJ, *IBF, Cfuse, CombMNZ Me	MeIBF MeADJ Cfuse	Me MeADJ MeIBF	Av Pg CombMNZ	PrOpt CombMNZ Av

(b) Summary of results for Kendall-tau

Figure 7.1: Summary of results for the baseline case: no missing objects and no correlation. The x-axis denotes the misinformation (n_{MI}). The columns from left to right increase the number of input rankers that use the weight function w′. The y-axis denotes the noise (σ^2). The rows from bottom to top increase the maximum variance of noise for the factors. The top-3 aggregators for each misinformation and noise case is displayed. Part (a) gives the top aggregators using the precision performance measure and part (b) gives the top aggregators using the Kendall-tau performance measure. Refer to Table 7.1 for the notation of the aggregators.

average aggregator produces the worst ranking. The average, in this case, allows for more information to be considered as compared to Me which ignores a great deal of information. In the presence of increased noise (Figure 7.6(c)), the Me produces the second worst ordering. ADJ and IBF optimizations produce rankings that contain little useful information due to the heavy noise; hence, these aggregators do not perform as well as Pg and PrOpt.

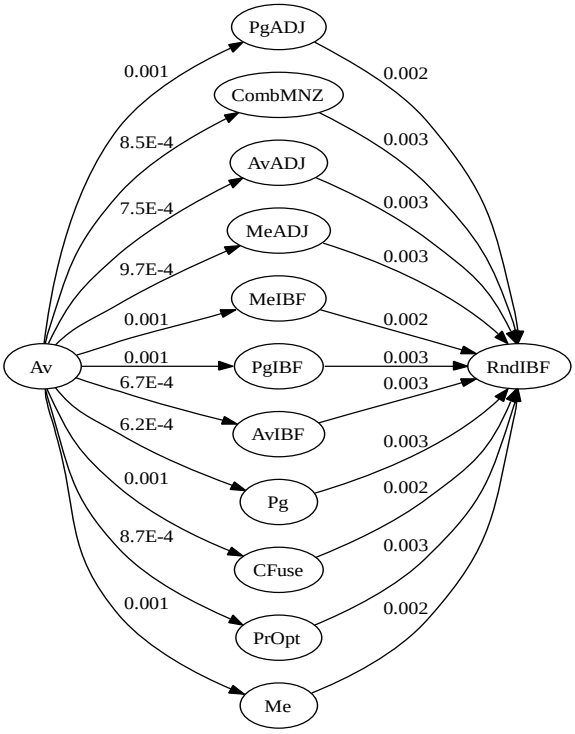
When noise is low but misinformation is high ($n_{MI} = 3, 4$), there is a greater need to incorporate as much information as possible from the input rankers. Hence, average becomes the best ranker again, also see Figure 7.6(a). This remains true even in the presence of moderate levels of noise. The average and median aggregators have a mixture of misinformation, which can not leverage the robustness to give a good ranking in case of high noise. The ADJ and IBF algorithms give better orderings over their elementary aggregator counterparts. We note that when the noise is high, PrOpt, Pg and IBF optimization appears to be best aggregation methods.

We observe that, for most cases, CombMNZ performs similarly to PrOpt in which an ordering can not be determined in the topological sorting for these two aggregators. Both aggregators consider the frequency of appearance and average rank for each object as a determining factor in ordering and their parallel behavior in the results are shown. Cfuse performs reasonably well in the presence of less misinformation ($n_{MI} = 1, 2$) for any noise level, e.g. see Figures 7.3 - 7.4. However, when misinformation is high ($n_{MI} = 3, 4$), Cfuse is one of the worst orderings except in the presence of high noise ($\sigma^2 = 7.5$). We had unweighted edges in our implementation of Cfuse, which may be the primary contributing factor to the performance of this aggregator.

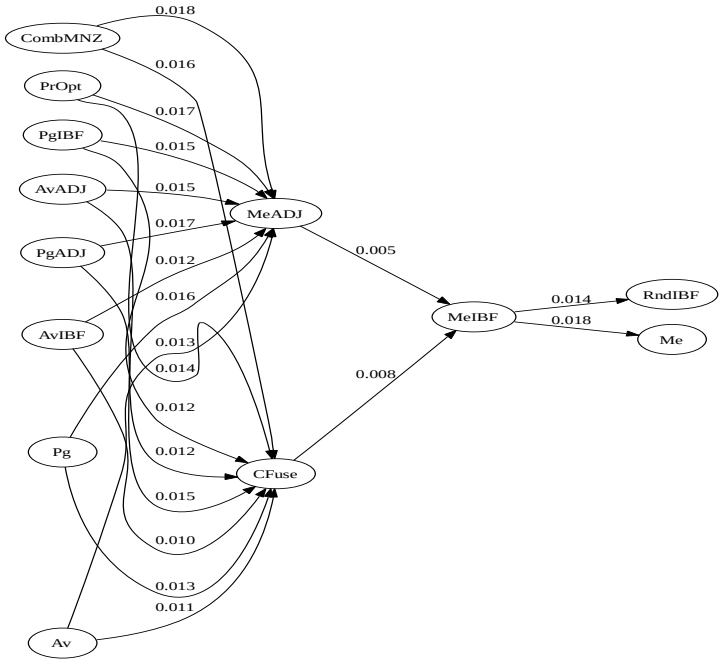
These results for precision remain unchanged for the TSAP performance measure except for high levels of noise. In the high noise cases, CombMNZ, PrOpt and Pg appear to be winners but IBF optimization appears to loose its competitiveness. This is a surprising result as optimizing for positional information, in fact, results in a loss of information that hurts performance for a measure that relies on positional information. The results for Kendall-tau error, Figures 7.7 - 7.11, are similar to precision as well. Note that this performance measure of the objects are in relatively

correct order. For instance, for Figures 7.4(a) and 7.9(a) where Me clearly outperforms Av, the benefit of using median is reinforced with respect to the Kendall-tau. While for Av, the objects in the top-K may be incorrect and out of order when the average aggregator is selected. In part (d), the noise level is a significant contributor to comparability of aggregators. The precision performance measure concludes the performing the IBF algorithm on any basic aggregator is a good approach but the order (minimization of the Kendall-tau) may suffer.

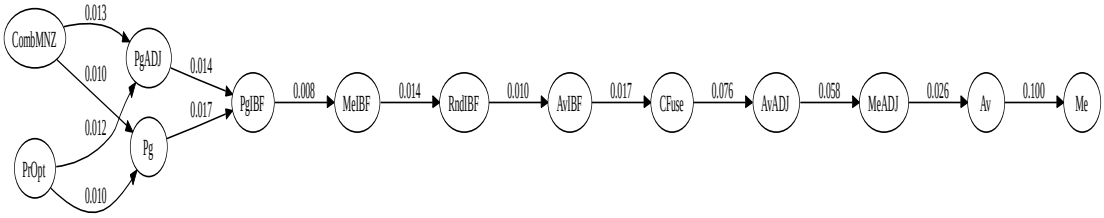
The first difference we note is that for high noise, PrOpt and CombMNZ does not always do as well since it does not directly optimize for positional information. For the highest noise value and $n_{MI} = 0, 1, 2$, PrOpt and CombMNZ performs be better than all others. However, for $n_{MI} = 3, 4$, Pg does better. For precision, both of these aggregation methods have equivalent performance. Since Pg incorporates information about the objects with missing ranks implicitly, this allows Pg to incorporate more useful information about the rankers. MeIBF appears to do very well (first or second place) in almost all noise cases for $n_{MI} = 1, 2$. Another interesting thing that we notice is that for high noise cases, IBF optimizers do not as well as PrOpt and Pg. Note that, IBF optimizer reduces the error with respect to the input rankers but ends up with worse performance with respect to the ground ranker for the high noise cases. Similar to TSAP, more information about the rankers needs to incorporated in these cases.



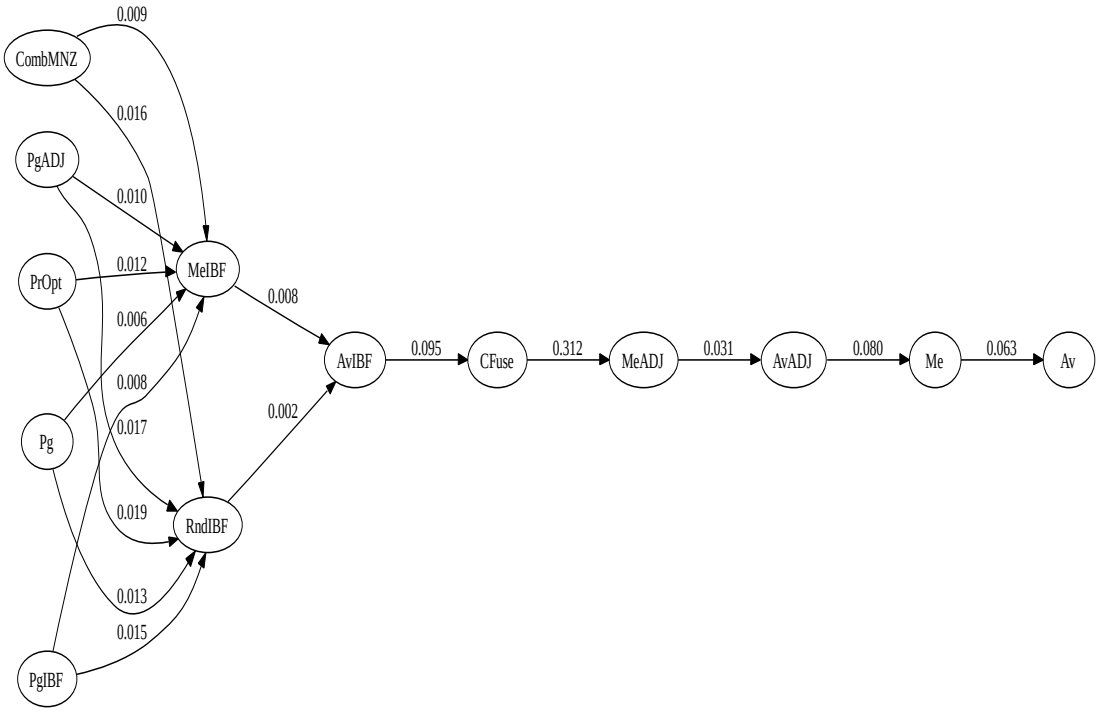
(a) $n_{MI} = 0, \sigma^2$=0.10



(b) $n_{MI} = 0, \sigma^2$=1.0

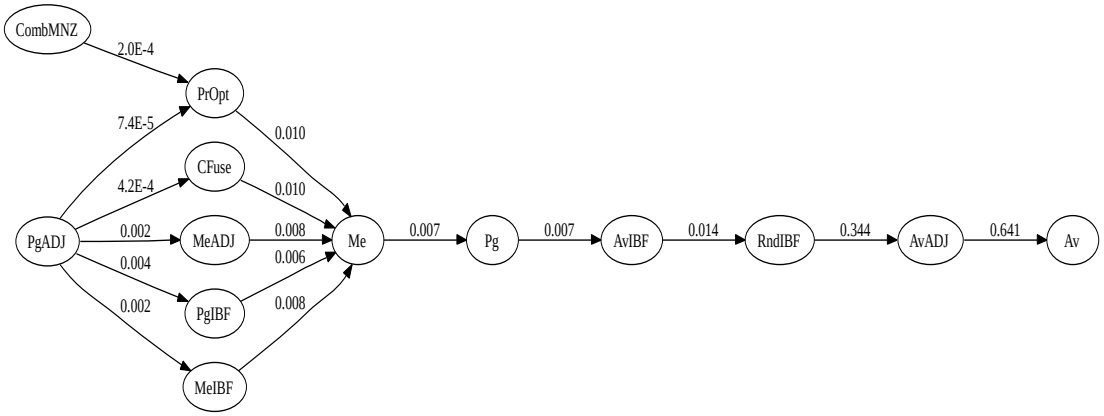


(c) $n_{MI} = 0, \sigma^2$=5.0

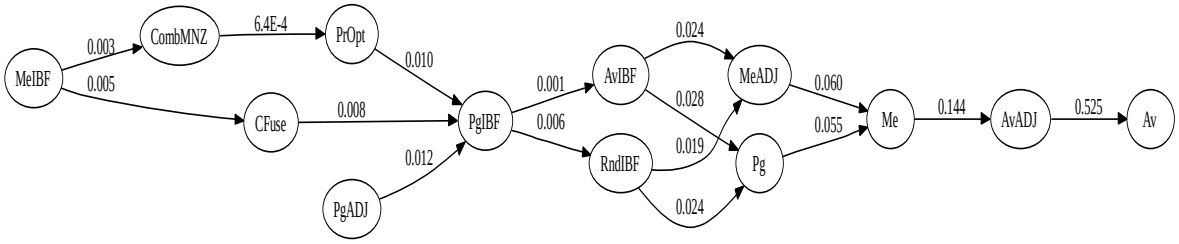


(d) $n_{MI} = 0, \sigma^2$=7.5

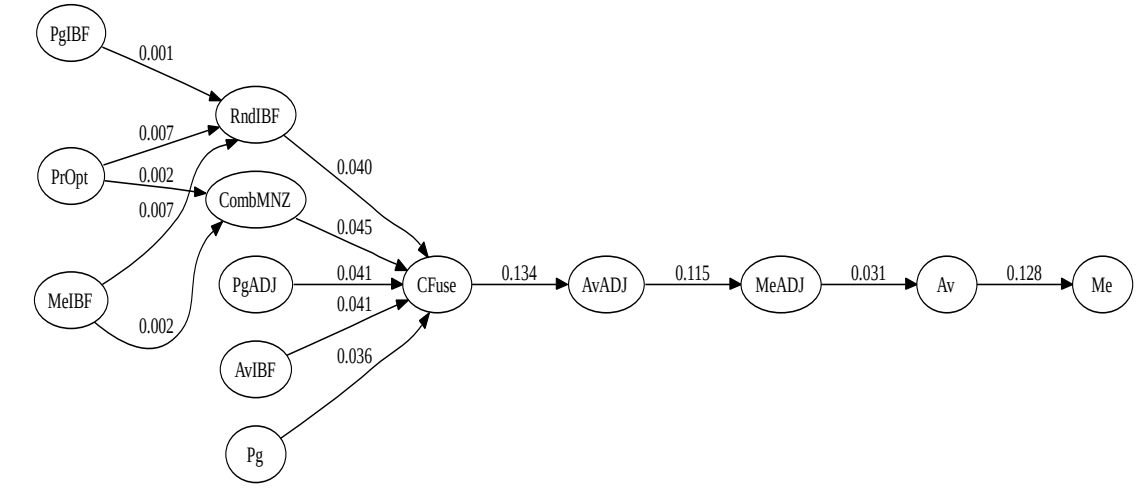
Figure 7.2: precision performance for $n_{MI} = 0$ with different levels of noise. The graphs (a)-(d) display the topological sort results of the aggregation methods where the quality of the aggregation method decreases from left to right for each misinformation (n_{MI}) and noise (σ^2). The edge weight indicate how much more precision is gained between adjacent aggregators. Refer to Table 7.1 for the notation of the aggregators.



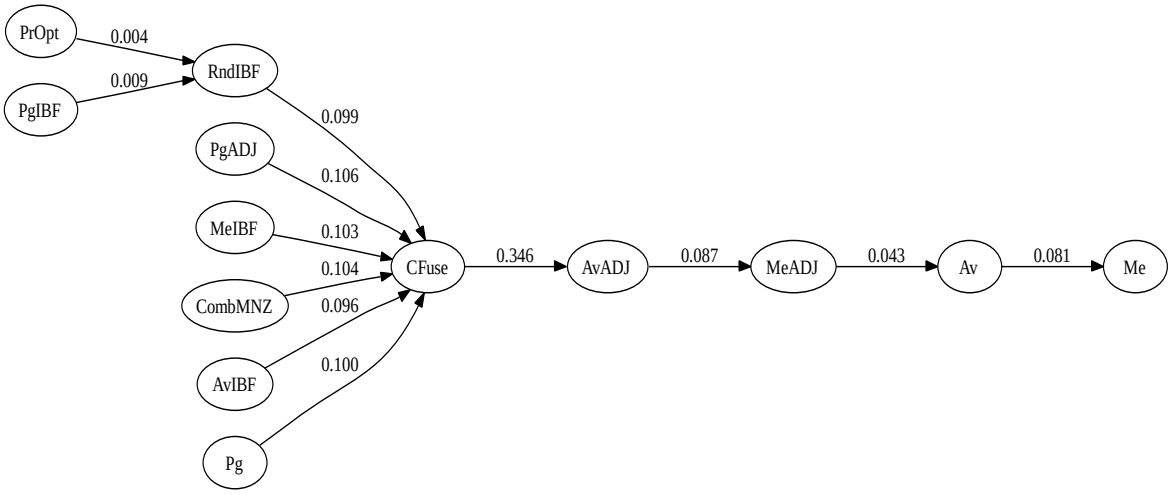
(a) $n_{MI} = 1, \sigma^2 = 0.10$



(b) $n_{MI} = 1, \sigma^2 = 1.0$

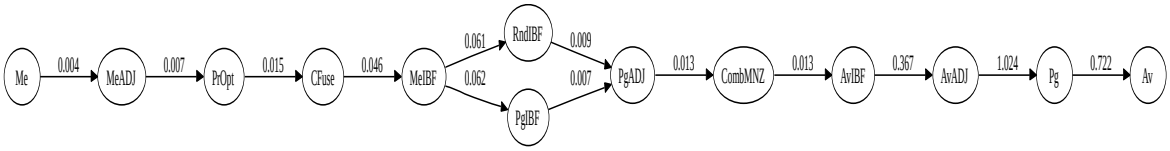


(c) $n_{MI} = 1, \sigma^2 = 5.0$

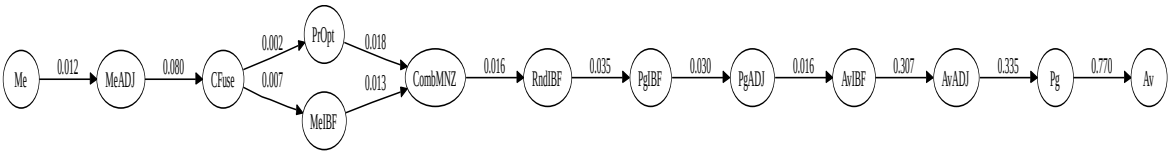


(d) $n_{MI} = 1, \sigma^2 = 7.5$

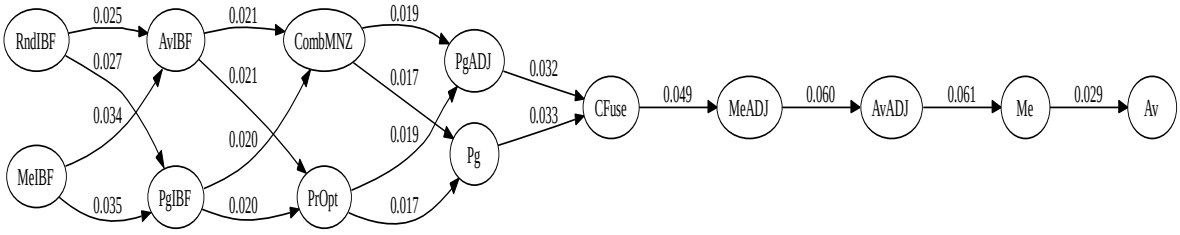
Figure 7.3: Precision performance for $n_{MI} = 1$ with different levels of noise. Refer to Figure 7.2 for the description of these graphs.



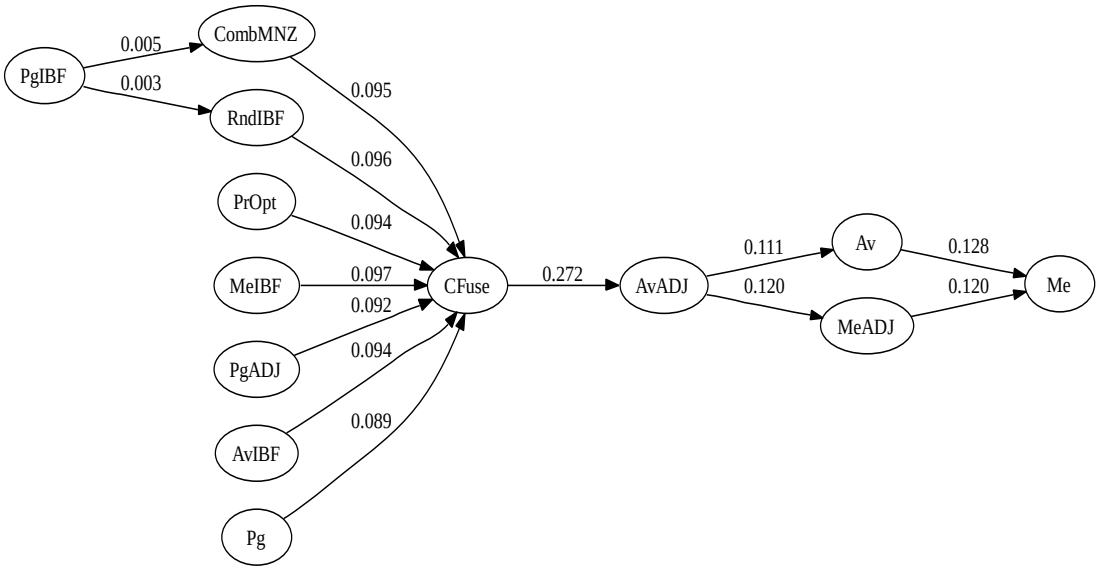
(a) $n_{MI} = 2, \sigma^2 = 0.10$



(b) $n_{MI} = 2, \sigma^2 = 1.0$

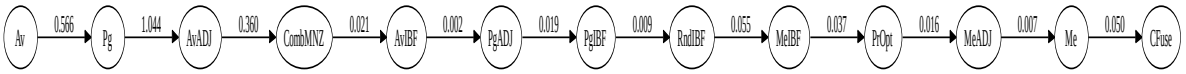


(c) $n_{MI} = 2, \sigma^2 = 5.0$

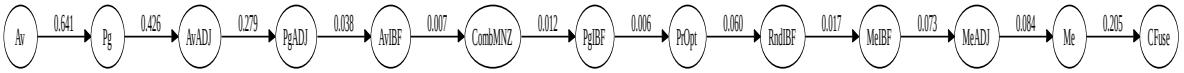


(d) $n_{MI} = 2, \sigma^2 = 7.5$

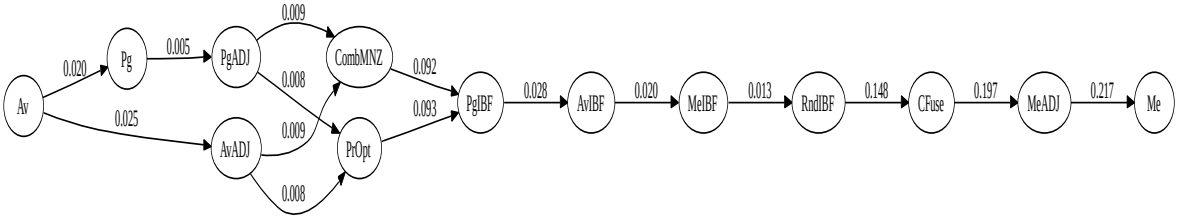
Figure 7.4: Precision performance for $n_{MI} = 2$ with different levels of noise. Refer to Figure 7.2 for the description of these graphs.



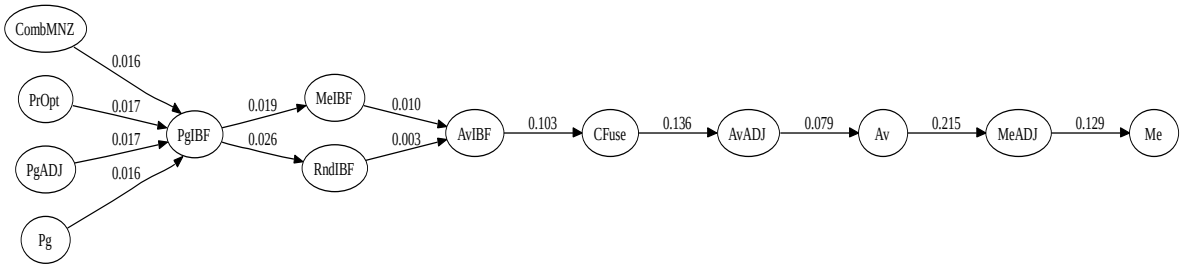
(a) $n_{MI} = 3, \sigma^2 = 0.10$



(b) $n_{MI} = 3, \sigma^2 = 1.0$

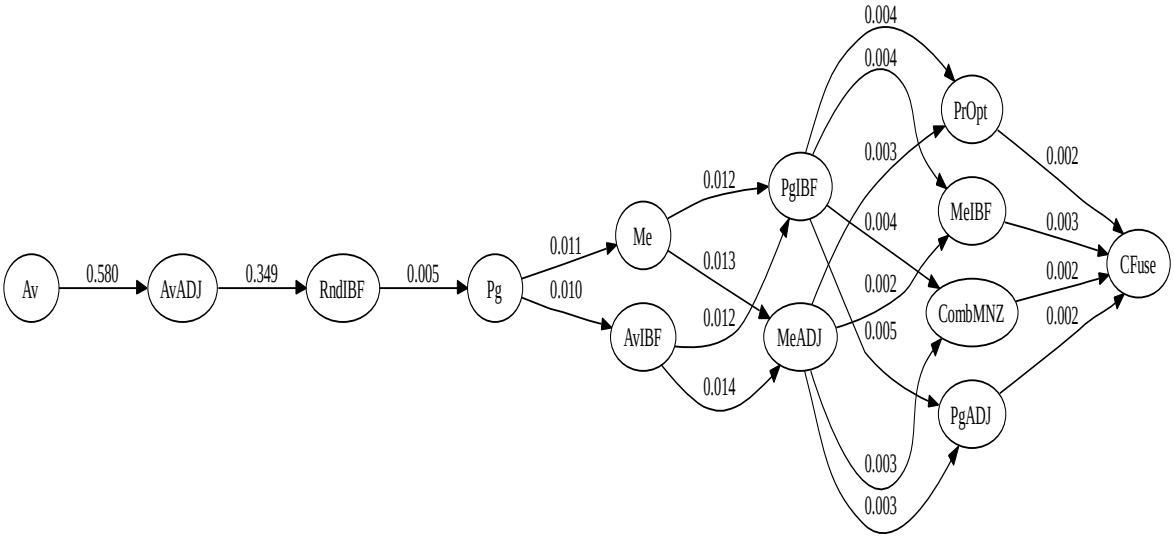


(c) $n_{MI} = 3, \sigma^2 = 5.0$

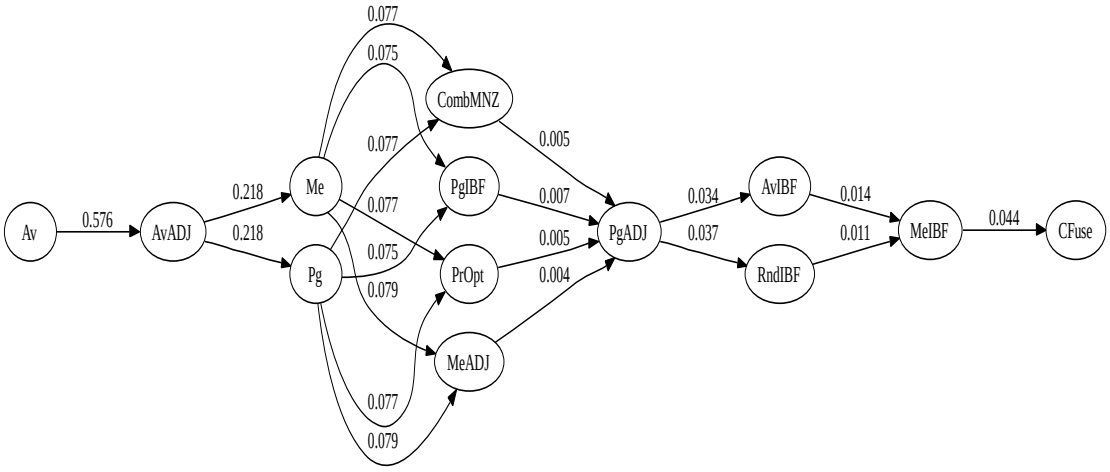


(d) $n_{MI} = 3, \sigma^2 = 7.5$

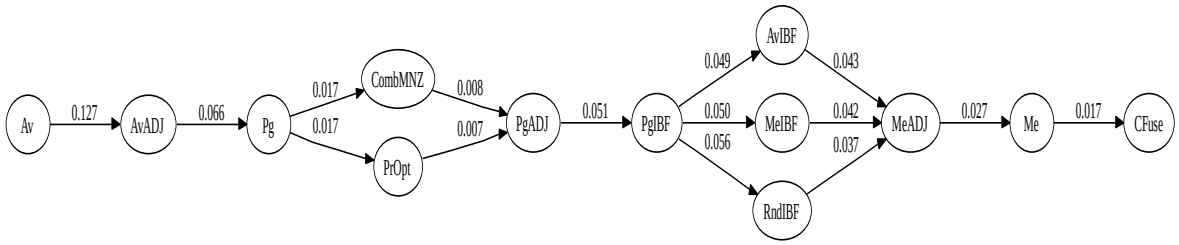
Figure 7.5: Precision performance for $n_{MI} = 3$ with different levels of noise. Refer to Figure 7.2 for the description of these graphs.



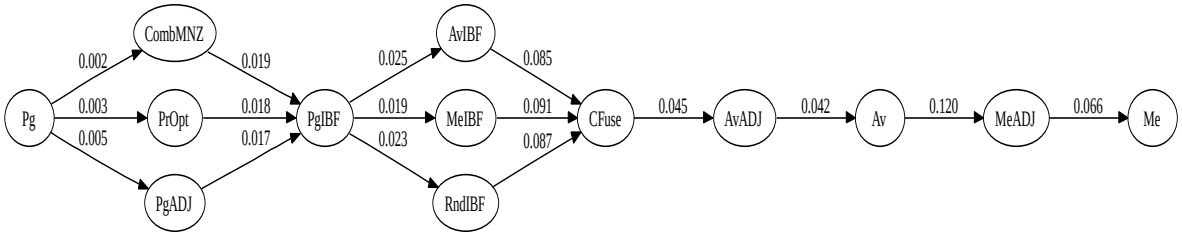
(a) $n_{MI} = 4, \sigma^2 = 0.10$



(b) $n_{MI} = 4, \sigma^2 = 1.0$

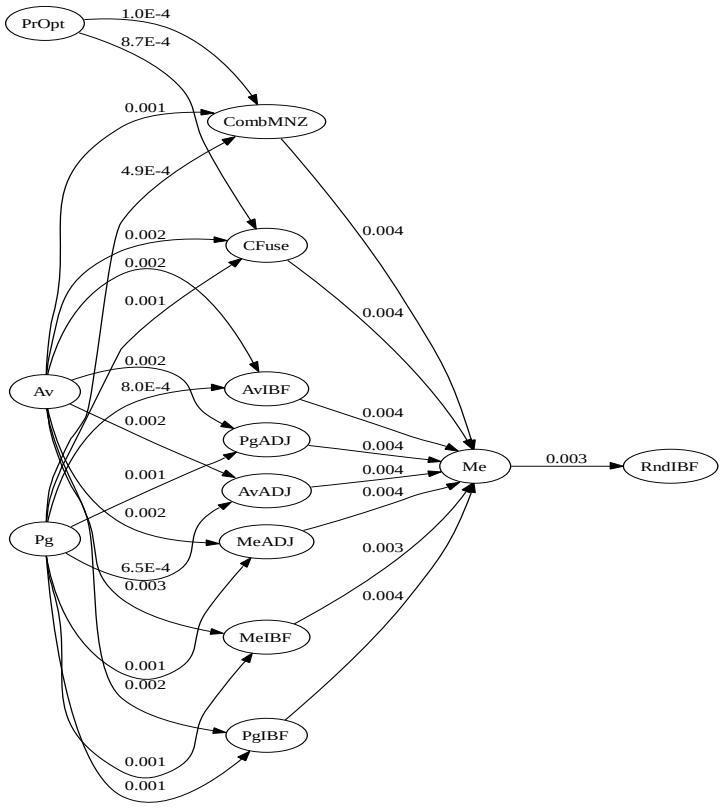


(c) $n_{MI} = 4, \sigma^2 = 5.0$

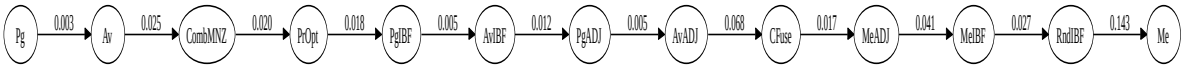


(d) $n_{MI} = 4, \sigma^2 = 7.5$

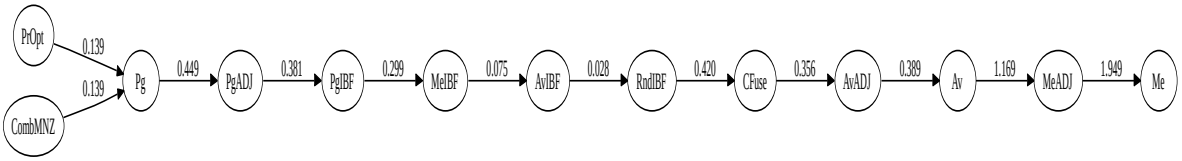
Figure 7.6: Precision performance for $n_{MI} = 4$ with different levels of noise. Refer to Figure 7.2 for the description of these graphs.



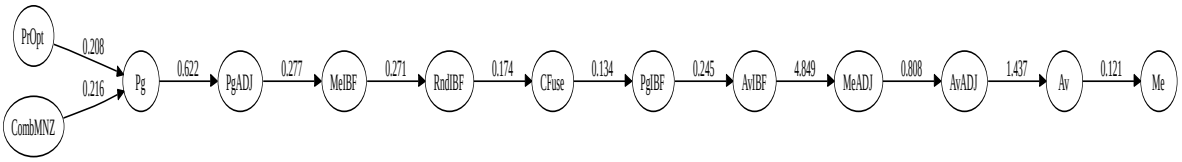
(a) $n_{MI} = 0, \sigma^2$=0.10



(b) $n_{MI} = 0, \sigma^2$=1.0

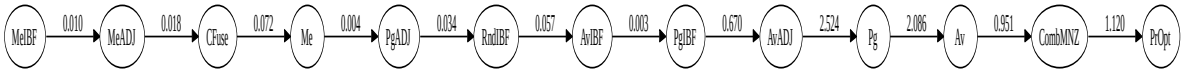


(c) $n_{MI} = 0, \sigma^2$=5.0

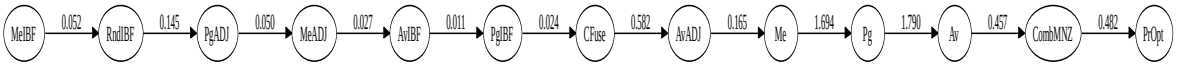


(d) $n_{MI} = 0, \sigma^2$=7.5

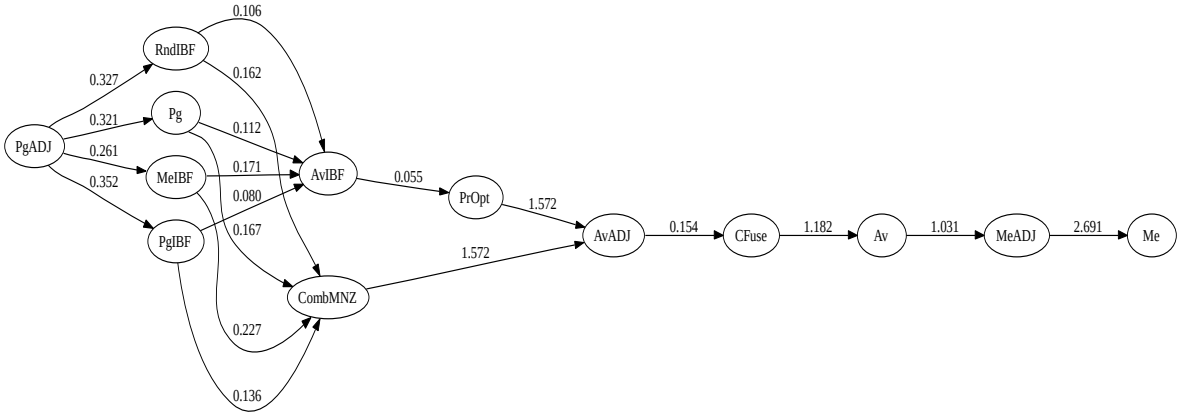
Figure 7.7: Kendall-tau performance for $n_{MI} = 0$ with different levels of noise. The graphs (a)-(d) display the topological sort results of the aggregation methods where the quality of the aggregation method decreases from left to right for each misinformation (n_{MI}) and noise (σ^2). The edge weight indicate the difference in swaps between adjacent aggregators. Refer to Table 7.1 for the notation of the aggregators.



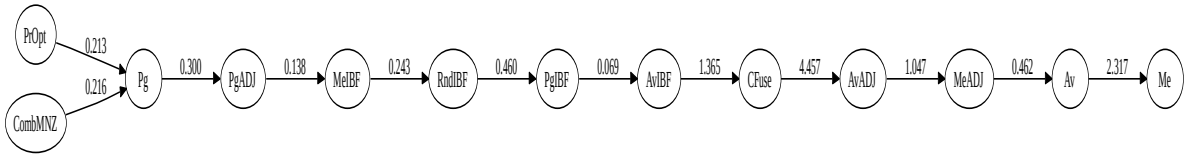
(a) $n_{MI} = 1, \sigma^2 = 0.10$



(b) $n_{MI} = 1, \sigma^2 = 1.0$

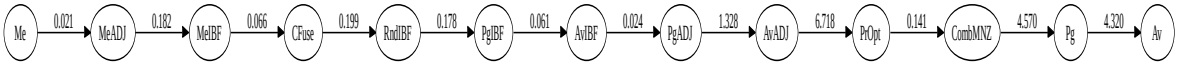


(c) $n_{MI} = 1, \sigma^2 = 5.0$

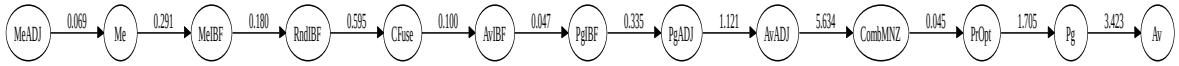


(d) $n_{MI} = 1, \sigma^2 = 7.5$

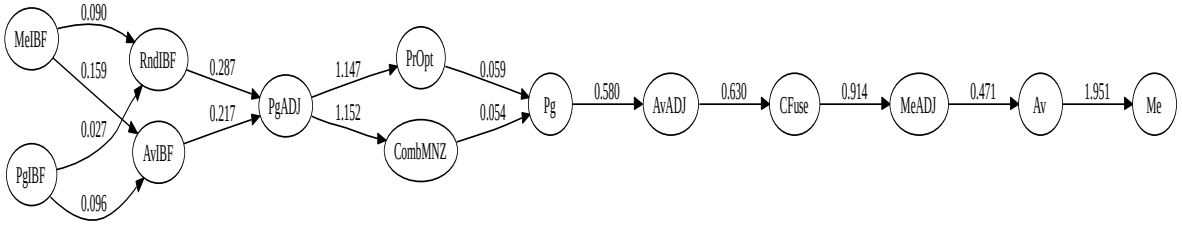
Figure 7.8: Kendall-tau performance for $n_{MI} = 1$ with different levels of noise. Refer to Figure 7.7 for the description of these graphs.



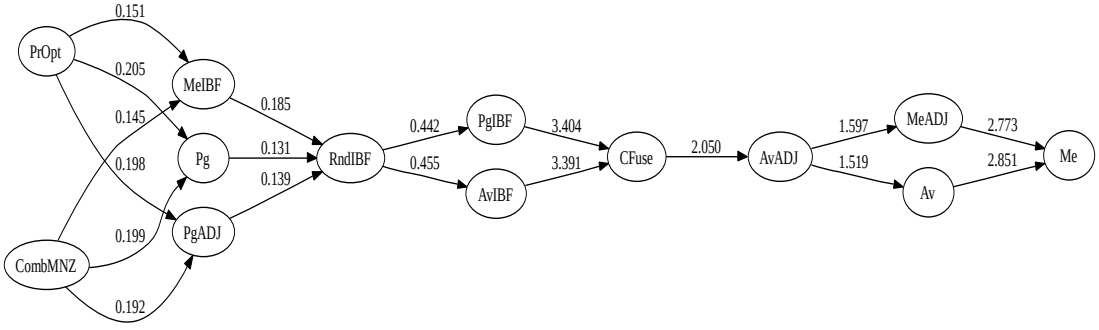
(a) $n_{MI} = 2, \sigma^2 = 0.10$



(b) $n_{MI} = 2, \sigma^2 = 1.0$

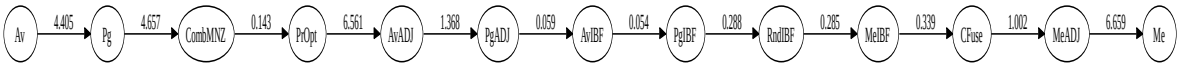


(c) $n_{MI} = 2, \sigma^2 = 5.0$

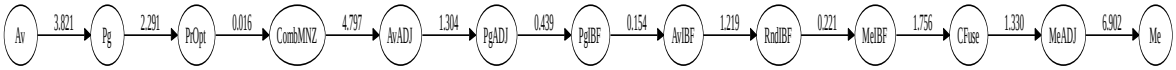


(d) $n_{MI} = 2, \sigma^2 = 7.5$

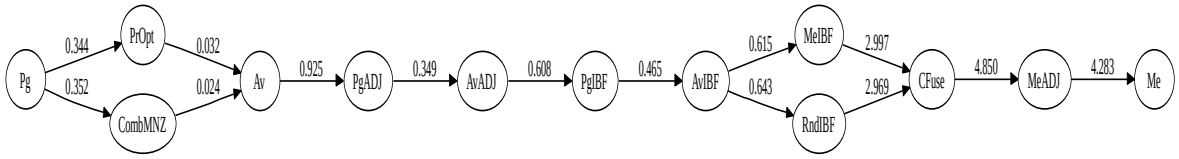
Figure 7.9: Kendall-tau performance for $n_{MI} = 2$ with different levels of noise. Refer to Figure 7.7 for the description of these graphs.



(a) $n_{MI} = 3, \sigma^2 = 0.10$



(b) $n_{MI} = 3, \sigma^2 = 1.0$

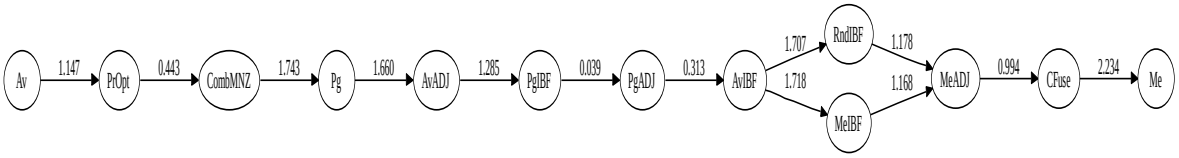


(c) $n_{MI} = 3, \sigma^2 = 5.0$

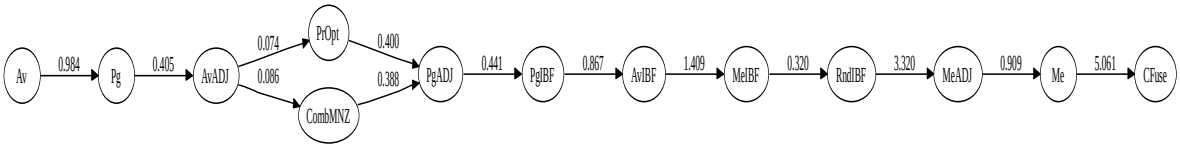
(d) $n_{MI} = 3, \sigma^2 = 7.5$

Figure 7.10: Kendall-tau performance for $n_{MI} = 3$ with different levels of noise. Refer to Figure 7.7 for the description of these graphs.

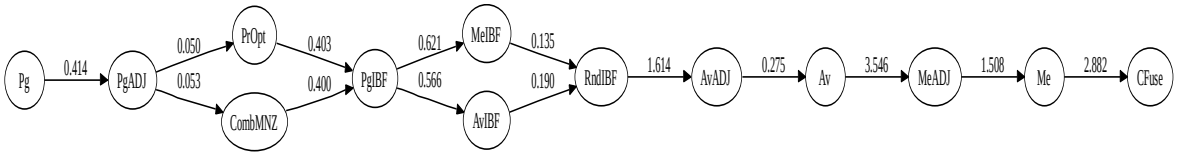
(a) $n_{MI} = 4, \sigma^2 = 0.10$



(b) $n_{MI} = 4, \sigma^2 = 1.0$



(c) $n_{MI} = 4, \sigma^2 = 5.0$



(d) $n_{MI} = 4, \sigma^2 = 7.5$

Figure 7.11: Kendall-tau performance for $n_{MI} = 4$ with different levels of noise. Refer to Figure 7.7 for the description of these graphs.

Significance of edge weights. The edge weights represent the degree that the left-hand aggregator outperforms the right-hand aggregator. In the precision optimized topological sorts, the edge weights are generally less than 1. The advantage of selecting a top aggregator is approximately one object. If the edge weights for the top aggregators becomes higher, then the benefit of using a top aggregator becomes greater. As in Figure 7.10(a) as compared to Figure 7.10(b), Av and Pg are the first and second aggregators for both noise levels. When $\sigma^2 = 0.10$, the edge weight is 4.405 meaning that Av is more than 4 swaps better than Pg while when $\sigma^2 = 1.0$, Av is almost 4 swaps better. The number of objects in our framework and the number of retrieved objects contribute to the limited impact of precision. In an environment with more objects and lower number of objects in the union of the rankers, the precision would not be initially so high and improvements can be made.

For the Kendall-tau optimized topological sorts, the number of swaps necessary to produce the minimal Kendall-tau are the edge weights. Figure 7.7(d) is a good example of the influence of swaps. The top aggregators are separated by approximately one swap but the median and average aggregators need more than one swap to coincide with the ordering of AvADJ. The differences observed between AvIBF and MeADJ are nearly 5 swaps so even though precision does not make a strong case for avoiding the median, average and their ADJ variants, the Kendall-tau performance measure does. If we sum the edge weights from the top aggregator to the bottom aggregator, such as Figure 7.11(a), there is nearly a 15 swap-advantage of PrOpt over the Me so the selection of an aggregator is more important but is not as significant for Figure 7.11(b). The objective in this case is to simply avoid the bottom aggregators since the majority of swaps occur toward the bottom of the topological sort. It is interesting to note that the lower performing aggregators are worse with respect to the Kendall-tau with at least one swap difference. The rankings outputted by the median and average as compared to the precision optimal require swaps of objects generally in the lower positions of the list.

Notation	Description
n	number of objects
m	number of factors
s	number of rankers
$\mathbf{w}$	true ranker's weights
$\mathbf{w}_j$	weights for ranker j
δ	shape parameter where large variance for positive factors near 3
β	shape parameter where large variance for negative factors near -3
σ^2	the maximum variance in the desired range
σ_f	correlation between two factors of the same object, < 1
σ_n	correlation between two objects for the same factor,< 1
σ_{nf}	correlation between two different objects for different factors, < 1
σ_f^*	correlation between two factors of the same object
σ_n^*	correlation between two objects for the same factor
σ_{nf}^*	correlation between two different objects for different factors
σ_R^*	correlation among rankers

Figure 7.12: Statistical framework symbols and definitions

7.1.2 Missing data

We reissue our experiments for each misinformation and noise level combination and examine the changes in the best aggregators from our initial evaluation. For simplicity, we duplicate the symbols and definitions of the statistical model in Figure 7.12. We run experiments where each ranker is missing 10% and 50% of all the objects in the database and examine how the choice of aggregator may change. In Figures 7.13 and 7.14, the summary of the best three aggregators for both the precision and Kendall-tau performance measures for 10% and 50% missing objects. With missing objects, there are fewer pairs of objects to compare for IBF and ADJ optimizations and the median values are based on fewer rankings. In our prior evaluation, more misinformation led to the average aggregator (Av) to be the best with PageRank (Pg) a close second best aggregator. As a result, the effectiveness of these rankers is reduced. For example, for 10% missing objects, Me is no longer the best for the cases where there is asymmetry between rankers and *IBF becomes prominent in these cases. As for 50% missing objects, we see that Av and AvADJ become very prominent for almost all the cases as more and more information needs to be incorporated. AvADJ provides small amount of robustness over Av and becomes

an important aggregator.

high noise	PrOpt, Pg*, CombMNZ MeIBF AvIBF, RndIBF	PrOpt, Pg*, CombMNZ, MeIBF AvIBF, RndIBF Cfuse	PgIBF CombMNZ, PrOpt, *IBF, Pg, PgADJ Cfuse	PrOpt, CombMNZ, Pg, PgADJ PgIBF MeIBF, RndIBF	Pg PrOpt, CombMNZ, PgADJ PgIBF
	PrOpt, CombMNZ Pg PgADJ	PrOpt, Pg*, CombMNZ, MeIBF AvIBF, RndIBF Cfuse	RndIBF, MeIBF AvIBF, PgIBF PrOpt, CombMNZ	Av Pg, AvADJ PrOpt, CombMNZ, PgADJ	Av AvADJ Pg
	Av AvADJ Pg, PgADJ, AvIBF	AvIBF Cfuse, RndIBF, PgIBF MeIBF	Cfuse RndIBF PgIBF	Av AvADJ Pg	Av AvADJ Pg
low noise	AvADJ PgADJ Av, AvIBF	Cfuse, AvIBF PgIBF RndIBF	Cfuse RndIBF PgIBF	Av Pg AvADJ	Av Pg PrOpt, CombMNZ, AvADJ
	less misinformation				**more misinformation**

(a) precision performance measure

high noise	PrOpt, Pg, CombMNZ PgADJ MeIBF	PrOpt, CombMNZ Pg PgADJ	PrOpt, CombMNZ PgADJ Pg, MeIBF	Pg PrOpt, CombMNZ PgADJ	Pg PgADJ PrOpt, CombMNZ
	PgADJ PgIBF Pg, AvIBF	PgADJ PgIBF MeIBF, AvIBF, RndIBF	PgIBF AvIBF MeIBF, RndIBF	Av Pg PrOpt, CombMNZ	Av AvADJ Pg
	AvIBF Cfuse PgIBF	RndIBF AvIBF MeIBF	Cfuse, RndIBF AvIBF PgIBF	Av Pg AvADJ	Av PrOpt, CombMNZ Pg
low noise	Cfuse MeADJ AvIBF	Cfuse AvIBF AvADJ	Cfuse RndIBF AvIBF	Av Pg AvADJ	Av PrOpt CombMNZ
	less misinformation				**more misinformation**

(b) Kendall-tau performance measure

Figure 7.13: 10% Missing Objects. Refer to Figure 7.1 for the full description of these tables.

high noise	PrOpt, CombMNZ Pg, PgADJ PgIBF	PrOpt, Pg,PgADJ CombMNZ PgIBF AvIBF, MeIBF, RndIBF	Pg PgADJ, PrOpt, CombMNZ PgIBF	PrOpt, Pg PgADJ, CombMNZ PgIBF	Pg, PrOpt, CombMNZ PgADJ Av, AvADJ, PgIBF
	Av AvADJ AvIBF	Av AvADJ PrOpt, CombMNZ, AvIBF, Pg	Av AvADJ Cfuse, CombMNZ	Av AvADJ Pg, Cfuse	Av AvADJ Pg
	Av AvADJ AvIBF	Av AvADJ AvIBF	Av AvADJ Cfuse	Av AvADJ Cfuse	Av AvADJ Pg
low noise	Av AvADJ AvIBF	Av AvADJ AvIBF	Av AvADJ AvIBF, Cfuse	Av AvADJ Cfuse	Av AvADJ Pg
	less misinformation				**more misinformation**

(a) precision performance measure

high noise	PgIBF PgADJ AvIBF	PgIBF PgADJ PrOpt, CombMNZ	PgADJ, PgIBF PrOpt, CombMNZ Pg	PgADJ Pg, PgIBF PrOpt, AvADJ, CombMNZ	AvADJ Av, PgADJ, PgIBF Pg, PrOpt, CombMNZ
	Av AvADJ PgIBF	AvADJ Av PgIBF	Av AvADJ PgIBF	Av AvADJ PgIBF	Av AvADJ PgIBF
	Av, AvADJ AvIBF PgIBF	AvADJ Av PgIBF	Av AvADJ PgIBF	Av AvADJ PgIBF, AvIBF	Av AvADJ AvIBF
low noise	AvADJ Av AvIBF	AvADJ Av AvIBF	AvADJ Av PgIBF	Av AvADJ PgIBF, AvIBF	Av AvADJ AvIBF
	less misinformation				**more misinformation**

(b) Kendall-tau performance measure

Figure 7.14: 50% Missing Objects. Refer to Figure 7.1 for the full description of these tables.

7.1.3 Correlated data

Experiments are ran that vary different correlations as discussed in Section 6. In each test, we vary only one of the correlations.

Correlation between two objects for the same factor for the ground truth (σ_n) In these tests, we introduce a correlation between objects for one factor. Positive correlation is tested in Figure 7.15 and negative correlations in Figure 7.16. We do not see a significant change in the results for positive correlations. However, for negative correlations, PrOpt and IBF become more prominent which signals a need for robustness.

Correlation between two factors of the same object for the ground truth (σ_f) In the next set of tests, we introduce correlations between two factors for the ground truth and consider the case where the correlation is positive in Figure 7.17 and negative in Figure 7.18. Again, for positive correlation, we do not see a big difference. In case of negative correlations, PrOpt becomes prominent for precision in low noise cases and Pg becomes prominent for Kendall-tau performance measure. In high noise cases, Av and AvADJ become the best rankers as more and more information need to be incorporated from the input rankers.

Correlation between the errors made by rankers for two objects for the same factor (one set for each ranker) (σ_n^*) In these experiments, we assign the correlated errors of two of the five rankers to be fixed at 0.60. The results are shown in Figure 7.19. The PrOpt aggregator no longer dominates as the best aggregation for high noise cases as compared to the other experiments. *Me** is robust with respect to noise in the bi-partisan case when a majority of the rankers align with the ground truth ($n_{MI} = 1, 2$). We observe that our IBF optimization performs well in a large number of cases, including high noise and some misinformation.

Correlation amongst rankers (σ_R^*) We performed experiments for positive correlations amongst all the rankers in Figure 7.20. With our low setting of 0.10, there

was no significant difference in performance for the precision and Kendall-tau performance measures as compared to our experiments without any correlations.

high noise	PrOpt, Pg*, CombMNZ, MeIBF AvIBF, RndIBF Cfuse	PrOpt, Pg*, CombMNZ MeIBF, RndIBF AvIBF	Pg*, *IBF, PrOpt, CombMNZ Cfuse AvADJ	PrOpt, Pg*, CombMNZ RndIBF, MeIBF, AvIBF Cfuse	Pg, PgADJ, PrOpt, CombMNZ PgIBF MeIBF, RndIBF
	PrOpt Pg, PgADJ, CombMNZ PgIBF	PrOpt, Pg*, CombMNZ, MeIBF AvIBF, RndIBF Cfuse	RndIBF, PgIBF, MeIBF PrOpt, CombMNZ, Pg, PgADJ, AvIBF Cfuse	Pg PrOpt, CombMNZ, PgADJ PgIBF	Pg PrOpt, PgADJ, CombMNZ PgIBF
	Pg PrOpt, CombMNZ, PgADJ, PgIBF AvADJ, AvIBF	MeIBF PgADJ PrOpt, PgIBF, RndIBF	MeADJ Me MeIBF	Av Pg AvADJ	Av AvADJ Me
low noise	Av AvIBF, PgADJ, CombMNZ PgIBF, MeIBF, AvADJ, MeADJ, Cfuse, PrOpt,	PrOpt, PgADJ, MeIBF, Cfuse, CombMNZ PgIBF MeADJ	Me MeADJ PrOpt	Av Pg AvADJ	Av AvADJ Pg
	less misinformation				**more misinformation**

(a) precision performance measure

high noise	Cfuse PrOpt, Pg, CombMNZ MeIBF, RndIBF	Cfuse PrOpt, CombMNZ MeIBF, RndIBF	MeIBF RndIBF, CombMNZ PrOpt, Pg	PrOpt Pg, MeIBF, CombMNZ PgADJ	Pg PrOpt, CombMNZ PgADJ
	CombMNZ PrOpt Pg	PrOpt, CombMNZ Pg PgADJ	PgADJ PrOpt, CombMNZ Pg, MeIBF	Pg PrOpt, CombMNZ PgADJ	Pg PrOpt, CombMNZ PgADJ
	Pg CombMNZ PrOpt, Av	RndIBF MeIBF PgADJ	MeIBF RndIBF MeADJ	Av Pg PrOpt	Av PrOpt CombMNZ
low noise	Av, Pg CombMNZ PrOpt, PgIBF	MeIBF MeADJ Cfuse	Me MeADJ MeIBF	Av Pg CombMNZ	PrOpt CombMNZ Av
	less misinformation				**more misinformation**

(b) Kendall-tau performance measure

Figure 7.15: Objects are positively correlated for one factor of the ground truth with $\sigma_n = 0.60$. Refer to Figure 7.1 for the full description of these tables.

	less misinformation				more misinformation
high noise	PrOpt, Pg*, *IBF, CombMNZ Cfuse AvADJ	Pg* PrOpt, AvIBF, CombMNZ MeIBF, RndIBF, Cfuse	Pg*, AvIBF PrOpt, MeIBF, RndIBF, CombMNZ Cfuse	PrOpt, Pg*, *IBF, CombMNZ Cfuse AvADJ	PrOpt, Pg*, *IBF, CombMNZ Cfuse AvADJ
	PrOpt, Pg*, *IBF, CombMNZ Cfuse AvADJ	PrOpt, Pg*, *IBF, CombMNZ Cfuse AvADJ	PrOpt, Pg*, *IBF, CombMNZ Cfuse AvADJ	PrOpt, Pg*, *IBF, CombMNZ Cfuse AvADJ	PrOpt, Pg, PgADJ PgIBF, AvIBF, MeIBF, CombMNZ Av, AvADJ, Me, MeADJ, Cfuse
	PrOpt, CombMNZ Pg PgADJ	PrOpt, PgADJ, Pg, CombMNZ MeIBF, PgIBF, RndIBF AvIBF, Cfuse	AvIBF RndIBF MeIBF	Pg, PgADJ PrOpt, CombMNZ Av	Me MeADJ Av
low noise	Pg PgADJ Av, AvADJ, PgIBF, CombMNZ	Pg, PgADJ PrOpt, PgIBF, CombMNZ AvIBF	Cfuse RndIBF PgIBF, MeIBF, AvIBF	Av AvADJ Pg	Av AvADJ Me

(a) precision performance measure

	less misinformation				more misinformation
high noise	PrOpt, Pg*, *IBF, CombMNZ, Cfuse AvADJ Av	PrOpt, CombMNZ, MeIBF, RndIBF Pg*, AvIBF AvADJ	PrOpt, CombMNZ Pg*, *IBF AvADJ	Pg, PgADJ MeIBF, RndIBF, CombMNZ PrOpt, PgIBF, AvADJ	Pg PgADJ, MeIBF, AvIBF, PgIBF, RndIBF, AvADJ
	Cfuse MeIBF RndIBF	MeIBF RndIBF PrOpt, CombMNZ	MeIBF RndIBF PrOpt, CombMNZ	Pg, PgIBF, MeIBF PgIBF, RndIBF, CombMNZ PrOpt, AvIBF	Pg PgADJ PgIBF, Me
	PrOpt, CombMNZ Pg PgADJ	PrOpt, CombMNZ Pg PgADJ	PrOpt, CombMNZ Pg, PgIBF, AvIBF, RndIBF PgADJ, MeIBF	Pg PgADJ PrOpt, CombMNZ	Me Av MeADJ
low noise	Pg Av CombMNZ	PgADJ PgIBF RndIBF	RndIBF MeIBF, AvIBF PgIBF	Av AvADJ Pg	Av AvADJ PrOpt

(b) Kendall-tau performance measure

Figure 7.16: Objects are negatively correlated for one factor of the ground truth with $\sigma_n = -0.60$. Refer to Figure 7.1 for the full description of these tables.

high noise	PrOpt, CombMNZ PgADJ MeIBF, Cufse	PrOpt, CombMNZ MeIBF PgADJ	MeADJ, MeIBF PrOpt CombMNZ	Pg PrOpt, PgADJ, Av, CombMNZ PgIBF, AvADJ	PrOpt, CombMNZ Av, Pg, PgADJ AvADJ
	PrOpt, Pg*, *IBF, CombMNZ Av, Cfuse MwADJ, MeIBF	CombMNZ PrOpt PgADJ, MeIBF, Cfuse	MeADJ Me MeIBF	Av Pg AvADJ	Av Pg AvIBF
	Av, AvIBF PrOpt, PgADJ, AvADJ MeADJ, MeIBF, Cfuse, CombMNZ	PrOpt, PgADJ, MeADJ, Cfuse, CombMNZ MeIBF Me	Me, MeADJ PrOpt, MeIBF Cfuse	Av Pg AvIBF, CombMNZ	Av AvIBF Pg
low noise	All Aggregators	PrOpt, PgADJ, MeADJ, MeIBF, Cfuse, CombMNZ Me PgIBF	Me, MeADJ PrOpt Cfuse	Av Pg CombMNZ	Av AvIBF, RndIBF
	less misinformation				**more misinformation**

(a) precision performance measure

high noise	PrOpt, CombMNZ PgADJ Pg, PgIBF	MeIBF PgADJ, RndIBF PgIBF	MeIBF RndIBF MeADJ	PrOpt, CombMNZ Pg Av	PrOpt CombMNZ Pg
	Pg Av, CombMNZ PrOpt	MeIBF, PgADJ AvADJ, MeADJ, Cfuse RndIBF	MeADJ MeIBF Me, Cfuse	Av Pg PrOpt	PrOpt Av CombMNZ
	Pg, Av PrOpt, AvADJ, AvIBF, PgIBF MeADJ, PgADJ, Cfuse, CombMNZ	MeADJ MeIBF, Cfuse PgADJ	Me, MeADJ MeIBF Cfuse	Av Pg CombMNZ	PrOpt CombMNZ Av
low noise	All Aggregators	MeADJ MeIBF, Cfuse Me	Me, MeADJ MeIBF, Cfuse AvADJ, PgADJ	Av Pg CombMNZ	PrOpt CombMNZ Av
	less misinformation				**more misinformation**

(b) Kendall-tau performance measure

Figure 7.17: Two factors are positively correlated for the ground truth with $\sigma_f = 0.60$. Refer to Figure 7.1 for the full description of these tables.

high noise	Av AvADJ Pg	Av AvADJ Pg	Av AvADJ Pg	Av AvADJ Pg	Av AvADJ Pg
	Av AvADJ Pg	Av AvADJ PrOpt, Pg, PgADJ, CombMNZ	Av, AvADJ PrOpt, Pg, PgADJ, CombMNZ PgIBF	Av AvADJ PrOpt, Pg, CombMNZ	Av AvADJ PrOpt, Pg, PgADJ, CombMNZ
	Pg, PrOpt, CombMNZ PgADJ PgIBF	PrOpt, CombMNZ Pg PgADJ	PrOpt, CombMNZ Pg, PgADJ PgIBF, MeIBF	PrOpt, CombMNZ Pg PgADJ	PrOpt, CombMNZ Pg PgADJ
	Av, AvIBF AvADJ, PgADJ, PgIBF	Av*, PgADJ, PrOpt, CombMNZ PrOpt, PgIBF, CombMNZ	AvADJ, PgADJ, AvIBF, CombMNZ PrOpt, Pg, PgIBF, MeADJ, Av, Cfuse	Av*, PrOpt, PgADJ PgIBF, MeADJ, Cfuse	PrOpt, Av*, PgADJ, CombMNZ Pg, PgIBF, MeADJ, Cfuse
low noise	PrOpt, Pg, PgIBF, CombMNZ	AvIBF	MeIBF	Pg, MeIBF	MeIBF
	less misinformation				more misinformation

(a) precision performance measure

high noise	Av AvADJ PgIBF	Av AvADJ PgADJ, PgIBF	Av AvADJ PgIBF	Av AvADJ PgIBF	Av AvADJ PgIBF
	Av AvADJ PgIBF	Av AvADJ PgADJ, PgIBF	Av AvADJ PgADJ, PgIBF	Av AvADJ PgADJ, PgIBF	Av AvADJ PgADJ, PgIBF
	Pg PgADJ PgIBF	Pg PgADJ PgIBF	Pg, PgADJ PgIBF AvIBF	Pg PgADJ PgIBF	Pg PgADJ PgIBF
	Av, Pg CombMNZ	Av, Pg PgIBF, CombMNZ	Pg Av	Av, Pg AvIBF, PgIBF	Av, Pg PgIBF, CombMNZ
low noise	PrOpt, AvIBF, PgIBF	AvIBF, PgADJ	PgADJ, PgIBF	AvADJ, PgADJ	PgADJ, AvIBF
	less misinformation				more misinformation

(b) Kendall-tau performance measure

Figure 7.18: Two factors are negatively correlated for the ground truth with $\sigma_f = -0.60$. Refer to Figure 7.1 for the full description of these tables.

	less misinformation				more misinformation
high noise	MeIBF AvIBF, PgIBF, RndIBF PrOpt, PgADJ, CombMNZ	RndIBF AvIBF, MeIBF PgIBF	AvIBF, MeIBF, RndIBF PgIBF PrOpt, PgADJ, CombMNZ	PrOpt, PgADJ, CombMNZ Pg, *IBF Cfuse	Pg PrOpt, PgADJ, CombMNZ PgIBF
	PrOpt, CombMNZ Pg, PgADJ, MeIBF PgIBF, RndIBF	MeIBF AvIBF, RndIBF PgIBF	MeIBF, RndIBF AvIBF, PgIBF PrOpt, CombMNZ	Av , Pg PrOpt, PgADJ, AvADJ, CombMNZ PgIBF	Av AvADJ Pg
	PrOpt, Av*, Pg*, CombMNZ MeADJ, MeIBF, Cfuse Me, RndIBF	MeIBF PrOpt, PgADJ, CombMNZ Cfuse	Me MeADJ Cfuse	Av Pg AvADJ	Av AvADJ Pg, Me
	Av PrOpt, Pg*, Cfuse, CombMNZ, *ADJ, AvIBF, MeIBF	PrOpt, PgADJ, MeADJ, MeIBF, Cfuse, CombMNZ PgIBF	Me MeADJ	Av Pg	Av AvADJ
low noise	Me	Me	PrOpt	AvADJ	RndIBF

(a) precision performance measure

	less misinformation				more misinformation
high noise	PrOpt, CombMNZ MeIBF Pg	MeIBF RndIBF PrOpt, CombMNZ	MeIBF RndIBF AvIBF	MeIBF RndIBF PrOpt, Pg	Pg PgADJ PrOpt, CombMNZ
	PrOpt, CombMNZ Pg PgADJ	MeIBF, RndIBF PgADJ PgIBF, AvIBF	MeIBF RndIBF AvIBF	Pg PrOpt, CombMNZ Av	Av Pg AvADJ
	Pg Av CombMNZ	MeIBF RndIBF PgADJ, MeADJ	MeADJ Me MeIBF	Av Pg PrOpt	Av PrOpt CombMNZ
	PrOpt, Pg*, CombMNZ, Cfuse, Av*, MeADJ, MeIBF Me, RndIBF	MeIBF MeADJ Cfuse	Me, MeADJ MeIBF Cfuse	Av Pg CombMNZ	PrOpt CombMNZ Av
low noise					

(b) Kendall-tau performance measure

Figure 7.19: Positive correlation between errors performed by rankers for two objects for the same factor with $\sigma_n^* = \langle 0.60, 0.60, 0.0, 0.0, 0.0 \rangle$. Refer to Figure 7.1 for the full description of these tables.

high noise	PrOpt, PgADJ, PgIBF, MeIBF, CombMNZ Pg, RndIBF AvIBF	PrOpt, Pg*, *IBF, CombMNZ Cfuse AvADJ, MeADJ	MeIBF, RndIBF PrOpt, Pg*, AvIBF, CombMNZ Cfuse	PrOpt, PgADJ, Pg, CombMNZ PgIBF MeIBF, RndIBF	Pg PrOpt, PgADJ, CombMNZ PgIBF
	PrOpt, CombMNZ Pg, PgADJ PgIBF, MeIBF	PrOpt, PgIBF, MeIBF Pg, PgADJ, AvIBF, RndIBF, CombMNZ Cfuse	MeIBF, RndIBF PgIBF AvIBF	Av Pg, AvADJ PgADJ	Av AvADJ Pg
	PrOpt, Av*, Pg*, CombMNZ MeADJ, Cfuse MeIBF	MeIBF, PgADJ, Cfuse PrOpt, PgADJ, PgIBF, RndIBF CombMNZ AvIBF	Me MeADJ Cfuse	Av Pg AvADJ	Av AvADJ Pg, Me
	Av PrOpt, Pg*, Cfuse, CombMNZ, *ADJ, AvIBF, MeIBF	PrOpt, PgADJ, MeADJ, MeIBF, Cfuse, CombMNZ PgIBF	Me MeADJ	Av Pg	Av AvADJ
low noise	Me	Me	PrOpt	AvADJ	RndIBF
	less misinformation				**more misinformation**

(a) precision performance measure

high noise	PrOpt, CombMNZ Pg PgADJ	PrOpt, CombMNZ Pg PgADJ	PrOpt, CombMNZ Pg, MeIBF PgADJ, RndIBF	Pg PrOpt, CombMNZ PgADJ	Pg PrOpt, CombMNZ PgADJ
	PrOpt, CombMNZ Pg PgADJ	PgADJ MeIBF Pg, RndIBF	MeIBF PgIBF, RndIBF AvIBF	Pg Av PrOpt, CombMNZ	Av Pg AvADJ
	Pg Av CombMNZ	MeIBF RndIBF MeADJ	Me, MeADJ MeIBF RndIBF	Av Pg PrOpt	Av PrOpt CombMNZ
	PrOpt, Pg*, CombMNZ, Cfuse, Av*, MeADJ, MeIBF Me, RndIBF	MeIBF, MeADJ Cfuse Me	Me MeADJ MeIBF	Av Pg CombMNZ	PrOpt CombMNZ Av
low noise					
	less misinformation				**more misinformation**

(b) Kendall-tau performance measure

Figure 7.20: Positive correlation amongst the rankers with $\sigma_R^* = 0.10$. Refer to Figure 7.1 for the full description of these tables.

7.2 Approximation Algorithms

We evaluate the performance of the rank aggregation and MFAS algorithms within the statistical framework. The list of algorithms used in our study are given in Table 7.1. The notation r to denote the ground truth, $r_1, \ldots, r_5$ to denote the input rankers and r_A to denote the result of any one of the algorithms in our study. Note that for graph algorithms, a topological sort of the aggregators is performed to find an ordering as discussed earlier.

Kendall-tau Lower Bound. We can compute a straightforward lower bound on the Kendall-tau which we will compare the algorithms to. Let n_{ij} be the number of rankers which rank object o_i above object o_j. Then $\mathcal{E}_{av} \geq min_\tau = \sum_{i<j} \min(n_{ij}, n_{ji})$. Recall that we define $\mathcal{E}_{av}$ in Section 4.1 for rank aggregation algorithms and Section 5.2.3 for the SubIBF algorithm. For each aggregation algorithm, an upper bound on the relative approximation ratio is as follows,

$$optz(r_A) = \frac{\mathcal{E}_{av} - min_\tau}{min_\tau}.$$

In the case if $min_\tau = 0$ when we consider low noise and less misinformation situations, we reassign $min_\tau = 1$ to avoid a division by zero. This approximation ratio addresses the closeness of the error to the lower bound in which the objective is to have $optz(r_A)$ be close to 0. This is the measure we will use for the algorithmic performance of all our algorithms.

Statistical Evaluation of Aggregation. We also compare how well the aggregate ranker r_A compares to the "solution" of the aggregation problem by computing its Kendall-tau distance to the ground truth given, $\tau(r_A, r)$. We will call this error the *Statistical Error*.

We re-examine the 20 different test settings by varying the noise and misinformation levels in the aggregation scenario as before. Noise is introduced for each factor. The ground truth ranker uses weights $\frac{1}{15}, \frac{2}{15}, \frac{3}{15}, \frac{4}{15}, \frac{5}{15}$. As misinformation increases, we add more rankers with the reverse weights $\frac{5}{15}, \frac{4}{15}, \frac{3}{15}, \frac{2}{15}, \frac{1}{15}$. For each test setting, we generate 40,000 instances.

Algorithm	Max	Avg
Av	0.525	0.311
Me	0.506	0.347
Pg	0.433	0.265
AvADJ	0.42	0.242
MeADJ	0.458	0.269
PgADJ	0.369	0.218
AvIBF	0.366	0.218
MeIBF	0.353	0.212
PgIBF	0.368	0.218
RndIBF	0.354	0.212
PrOpt	0.626	0.289
Cfuse	0.345	0.205
CombMNZ	0.571	0.282
SubIBF	1.33	0.44
AvSubIBF	0.44	0.272
MeSubIBF	0.482	0.277
PgSubIBF	0.369	0.219
Greedy	1.47	0.481
CUT	0.358	0.213
AvCUT	0.358	0.213
MeCUT	0.359	0.213
PgCUT	0.359	0.213

Table 7.2: Error to the lower bound (*optz*)

7.2.1 Kemeny Optimization Results

We first report on the Kemeny optimization performance of the algorithms as measured by *optz*. The results given in Table 7.2 which shows the average and worst case relative error with respect to the lower bound. This average and worst case performance are computed over all the 800,000 instances. In the average case, we observe that xADJ, xIBF, xSubIBF and xCUT produced a better lower bound with respect to the Av, Me and Pg aggregators. Cfuse has the best result with an average of 20% over the lower bound over all misinformation and noise settings. The xIBF and xCUT aggregators are relatively unaffected by the initial ranking while xADJ and xSubIBF are. Due to the local optimizing nature of ADJ and SubIBF, the reduction of Kendall-tau is restricted, which produces the higher *optz* results. With regard to the maximum relative error, Greedy and SubIBF have the highest

high noise	Cfuse MeIBF RndIBF	Cfuse MeIBF RndIBF	MeIBF Cfuse RndIBF	Cfuse MeIBF RndIBF	Cfuse MeIBF RndIBF
	PgSubIBF CUT PgCUT	MeCUT CUT PgCUT	MeIBF RndIBF AvCUT	Cfuse MeIBF CUT	Cfuse AvCUT MeCUT
	PgADJ AvADJ AvCUT	SubIBF PgSubIBF CUT	SubIBF MeIBF RndIBF	Cfuse MeIBF RndIBF	Cfuse PgSubIBF PgCUT
low noise	PgADJ AvADJ MeCUT	SubIBF PgSubIBF Cfuse	SubIBF Cfuse MeIBF	Cfuse MeIBF RndIBF	PgSubIBF Cfuse AvCUT
	less misinformation				more misinformation

Figure 7.21: Best aggregate rankers w.r.t. lower bound. Refer to Figure 7.1 for the full description of these tables.

optz since the bi-connected components may be contain a unique ordering of the objects.

For the twenty variations according to the noise and misinformation, we show the best few algorithms in Figure 7.21. We display the best three aggregate rankers. PgSubIBF, MeIBF and RndIBF appear to dominate the results for the majority of settings, which is probably because the other algorithms give greater weight to misleading data. The greedy algorithm appears to have poor performance, indicating that it may remove edges suboptimally. The definition of an edge weight sets the graph based algorithms apart from the rank aggregation algorithms. The graph based algorithms perform a *ranker* aggregation in which the edge weight captures the dominant ordering for a pair of objects. The rank aggregation algorithms, on the other hand, focus on the ranks for the pair of objects and use more information in order to determine the aggregate ranker. The graph based algorithms do not use the ranks; hence there is information loss which causes these algorithms not to produce competitive aggregate rankings.

Algorithm	CPU (sec)	Stat. Error
Av	0.002	33.612
Me	0.001	37.974
Pg	0.006	31.645
AvADJ	0.009	33.377
MeADJ	0.009	35.541
PgADJ	0.013	31.816
AvIBF	0.097	32.266
MeIBF	0.097	32.392
PgIBF	0.092	32.076
RndIBF	0.111	32.438
PrOpt	0.002	31.752
Cfuse	0.008	34.607
CombMNZ	0.001	31.748
SubIBF	0.172	32.149
AvSubIBF	0.148	34.412
MeSubIBF	0.184	36.289
PgSubIBF	0.329	32.804
Greedy	0.034	32.419
CUT	0.224	32.253
AvCUT	0.224	32.261
MeCUT	0.223	32.261
PgCUT	0.368	32.26

Table 7.3: Runtime and average error to the ground truth

Computational Cost. In Table 7.3, we display run time of each algorithm. As can be expected the simplest algorithms are quickest, and as the algorithms become more complex, the run time may increase by as much as two orders of magnitude. From the graph based algorithms, Greedy is the only algorithm that is competitive to the most of the rank aggregation algorithms.

7.2.2 Statistical Performance Results

The average statistical error for each algorithms is given in Table 7.3, and Figure 7.22(a) shows the best few performers depending on the particular aggregation scenario parameterized by noise and misinformation. In Table 7.3, we see that the error from the statistical framework varies from 31 to 38 disagreements. As expected, Me has a higher average number of disagreements since the algorithm

ignores (some correctly ordered) rankers. In general, the graph based algorithms are more computationally expensive and do not significantly reduce the statistical error. Among the optimization methods, e.g. xADJ, xIBF, xSubIBF and xCUT, our xIBF algorithm seems to be the better choice with respect to the Kemeny optimization, cost and statistical error.

When only considering the rank aggregation algorithms as shown in Figure 7.22(b), the Av and Pg algorithms are amongst the best algorithms as misinformation increases. PrOpt and Pg are the best aggregate algorithms in the case of high noise. However, when we include the graph based algorithms, the Greedy and SubIBF algorithms outperform both Av and Pg when there is more misinformation. The Greedy and PgSubIBF only outperform the rank aggregation algorithms in highest noise and misinformation cases. Me seems to dominate when there is a balance of noise and misinformation in the middle of the table. The graph based algorithms outperform the other rank aggregation algorithms in our statistical framework because of their combination of global and local ordering of the objects. In the global ordering, the objects are grouped. The local ordering allows for the global relationships between objects to be removed. The objects in each partition can then be ordered only with respect to each other.

7.3 TREC Data Collection

We use the rankings submitted by the participants of the Text REtreival Conference (TREC) as input rankers to the rank aggregation algorithms. Three datasets (TREC-3, TREC-5 and TREC-9) are used each comprising of 50 queries, which was also used in [97]. Each participant devises a system, which retrieves 1000 documents, and returns a ranking of these documents for a particular query. In TREC, human evaluators are used to determine if a document is relevant or irrelevant. The relevant documents are then compared to these rankings of up to 1000 documents using different types of performance evaluators including the precision and TREC-style average precision as described previously.

For each query, we repeat the following 50 iterations: we select the top-K ($K = \{5, 10, 20, 50\}$) documents from a randomly set of 5 input rankers as input

high noise	PgSublBF	PrOpt, CombMNZ, PgSublBF	PrOpt, CombMNZ	Pg	Greedy
	PrOpt, CombMNZ	Pg	Pg, PgADJMeIBF	PrOpt, CombMNZ	Pg
	Pg	PgADJ	RndIBF	PgADJ	PgADJ
	PrOpt, CombMNZ Pg PgSublBF	PgSublBF PgADJ *CUT	MeIBF, PgIBF AvIBF, RndIBF *CUT	Greedy Pg PrOpt, CombMNZ	Greedy SublBF Av
	Pg Av CombMNZ	SublBF MeIBF *CUT, MeSubIBF	SublBF MeIBF MeADJ	Greedy Av SublBF	Greedy SublBF Av
	Av	SublBF	SublBF	Av	SublBF
	*ADJ, *IBF, Pg*,*CUT,*SublBF, Cfuse, PrOpt, CombMNZ	MeSubIBF	MeSubIBF	Greedy	Greedy
low noise	Me	MeIBF	Me	Pg	PrOpt
	less misinformation				**more misinformation**

(a) all algorithms

high noise	PrOpt, CombMNZ	PrOpt, CombMNZ	PrOpt, CombMNZ	Pg	Pg
	Pg	Pg	Pg, PgADJ, MeIBF	PrOpt, CombMNZ	PgADJ
	PgADJ	PgADJ	RndIBF	PgADJ	PrOpt, CombMNZ
	PrOpt, CombMNZ	PgADJ	MeIBF, PgIBF	Pg	Av
	Pg	Pg, PgIBF, MeIBF, RndIBF	AvIBF. RndIBF	PrOpt, CombMNZ	Pg
	PgADJ	AvIBF, CombMNZ	PgADJ	Av	AvADJ
	Pg Av CombMNZ	MeIBF RndIBF PgADJ	MeADJ Me MeIBF	Av Pg PrOpt	Av PrOpt CombMNZ
	PrOpt, Av, Pg	MeIBF	Me	Av	PrOpt
	*ADJ, *IBF, Cfuse, CombMNZ	MeADJ	MeADJ	Pg	CombMNZ
low noise	Me	Cfuse	MeIBF	CombMNZ	Av
	less misinformation				**more misinformation**

(b) rank aggregation algorithms only

Figure 7.22: Best aggregate rankers w.r.t. ground truth using the Kendall-tau performance measure. Refer to Figure 7.1 for the full description of these tables.

to the rank aggregation methods. We compute the TREC-style average precision comparing the aggregate ranker to the relevant documents determined by human evaluators. For each query, we find the mean average precision over the 50 iterations for each value of K and aggregator. We display the mean average precision over all queries for each aggregator.

	top-5	top-10	top-20	top-50	Best Overall
Av	0.2769	0.1805	0.1109	0.0548	0.6231
Me	0.251	0.1697	0.1051	0.0525	0.5783
Pg	**0.2884**	**0.1882**	**0.1147**	**0.0562**	**0.64751**
AvIBF	0.2835	0.1857	0.1126	0.0548	0.6366
MeIBF	0.284	0.1863	0.1131	0.0549	0.6383
PgIBF	**0.2862**	0.1865	0.1132	0.055	**0.6409**
AvADJ	0.2755	0.1815	0.1111	0.0546	0.6227
MeADJ	0.2582	0.172	0.1058	0.0527	0.5887
PgADJ	0.2516	0.1876	0.1112	0.0559	0.6063
RndIBF	0.2724	0.1759	0.1094	0.0548	0.6125
PrOpt	0.2637	0.1806	0.1113	**0.0562**	0.6118
CFuse	0.284	0.1858	0.1106	0.0531	0.6335
CombMNZ	0.2697	**0.1877**	**0.1147**	**0.0562**	0.6283

Table 7.4: TREC-3 Results

	top-5	top-10	top-20	top-50	Best Overall
Av	0.2289	0.1428	0.0828	0.0387	0.4932
Me	0.2012	0.1274	0.0758	0.0357	0.4401
Pg	**0.2346**	**0.1463**	**0.0844**	**0.0388**	**0.5041**
AvIBF	**0.232**	0.1453	0.0835	0.0385	**0.4993**
MeIBF	0.2251	0.1458	**0.0837**	0.0375	0.4921
PgIBF	0.2135	0.1405	0.0836	0.0386	0.4762
AvADJ	0.218	0.1391	0.0828	0.0385	0.4784
MeADJ	0.2078	0.1302	0.0771	0.036	0.4511
PgADJ	0.196	0.1459	0.0841	**0.0388**	0.4648
RndIBF	0.2036	0.1454	0.0807	0.0384	0.4681
PrOpt	0.2139	**0.1458**	0.0843	0.0387	0.4827
CFuse	0.2216	0.1444	0.0828	0.0382	0.487
CombMNZ	0.2077	0.1408	0.0818	0.0376	0.4679

Table 7.5: TREC-5 Results

We show in Tables 7.4- 7.6 the mean average precision over the 50 queries in

	top-5	top-10	top-20	top-50	Best Overall
Av	0.176	0.1033	0.0599	0.0272	0.3664
Me	0.1418	0.0888	0.0531	0.0247	0.3084
Pg	**0.1808**	**0.1065**	**0.0608**	0.0272	**0.3753**
AvIBF	0.1741	0.105	0.0606	0.0276	0.3673
MeIBF	0.1786	0.1057	0.0607	0.0274	0.3724
PgIBF	**0.1791**	0.1052	0.0607	0.0275	**0.3725**
AvADJ	0.1764	0.1025	0.0602	**0.0276**	0.3666
MeADJ	0.1503	0.0916	0.054	0.0251	0.321
PgADJ	0.1569	0.1062	**0.061**	**0.0277**	0.3518
RndIBF	0.1761	0.1053	0.0605	0.0263	0.3682
PrOpt	0.1781	0.1063	0.0606	0.027	0.372
CFuse	0.1723	0.1051	0.055	0.0265	0.3589
CombMNZ	0.1768	**0.1065**	0.0606	0.02591	0.3698

Table 7.6: TREC-9 Results

each TREC data collection. We bold the best two aggregators for each column. The last column sums the error from each top-K for each aggregator to determine which are the best aggregators overall. We see throughout these results that Pg produces the best aggregators for the majority of values of K. The CombMNZ and Condorcet-fuse aggregators only perform well in Table 7.4 for most top-K values; however these methods are not one of the best overall. The xIBF, xADJ and PrOpt are among the aggregators that perform second best. We observe that certain aggregators perform best under different circumstances. Our xIBF aggregators tend to give a higher mean average precision than the standards CombMNZ and Condorcet-fuse.

7.4 Real Data

We have tested five aggregation methods on a small sample (6 queries) using the following search engines for rankers:

queries	computer viruses, death penalty, mining coal gold silver, photography, wireless communications
rankers	Altavista, Clusty, Dogpile, Excite, Google, Looksmart, Metacrawler, MSN, Search, Teoma, Yahoo
aggregators	Average, Median, Median with IBF optimization, PageRank, Precision Optimal

For each query and every pair of aggregation methods, we determined manually which aggregator was superior. To do this, we used the pair of objects with highest rank discrepancy in the two aggregate rankings, and then manually determined which object was more relevant to the query. The results between every pair of rankers were averaged over queries. The relative performance of these aggregators on this set of queries is summarized in the figure below.

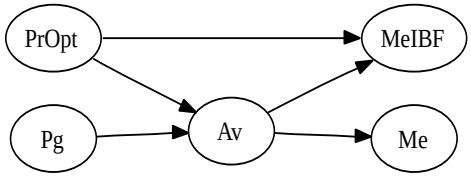


Figure 7.23: small sample real dataset results

As can be observed in Figure 7.23, the optimal aggregator is Precision Optimal and PageRank aggregators. The average variance in the rank of an object over the rankers was high and a clustering of the rankings provided by the different rankers revealed only one significant cluster, indicating little asymmetry between the rankers. To find a cluster among the rankers, we constructed a graph where each ranker is a vertex and each edge has a weight that corresponds to the average Kendall-tau distance between the two rankers for the above studied queries. Assuming the rankers are not all using the incorrect weights, we can assume that there is little misinformation in this setting. As a result, we conclude that the we are in the high noise low misinformation aggregation scenario, and according to our results within the statistical framework, the optimal aggregator should be Precision Optimal and PageRank, which agrees with what was determined empirically.

Our results indicate that through a somewhat qualitative analysis of the ranker results, the level of noise and misinformation can be roughly determined and leads to the correct choice of aggregator from within the statistical framework.

Our experimental results reveal which aggregation methods work well in which aggregation scenarios. In a practical setting, one does not know *apriori* which aggregation scenario one is in, hence it is not easy to determine which method to use. We give some results using the results from search engine queries and based on human evaluation of the aggregator outputs. Based on these (small sample) results, we conclude that the Precision Optimal or PageRank aggregation is optimal with the context of our limited sample size.

Based on a simplistic analysis of the ranker data, we conclude that the aggregation scenario is the high-noise, small misinformation case, where our statistical framework indicates that the optimal aggregator should be the Precision Optimal aggregator, which is in accordance with our results.

CHAPTER 8
DETERMINING MISINFORMATION AND NOISE

8.1 Introduction

In the previous chapters, we have shown that the choice of the optimal ranker depends on two main factors: the level of noise and misinformation the rankers bring with respect to the ground truth. In most real life situations, the ground truth is not known. In this chapter, we develop methods to identify the specific noise and misinformation scenario for input rankers without foreknowledge of the ground truth. These methods allow us to choose the best ranker for each scenario. We expect that by dynamically adapting the rank aggregation strategy, we will be able to outperform any static aggregation method. Note that the method used for predicting the best aggregation method should be easy to compute and should not require large amounts of memory. We show that our methods satisfy both criteria.

In order to estimate the level of misinformation, we cluster the rankers with respect to some measure of closeness such as precision or Kendall-tau. Each cluster is meant to represent a different opinion. We then compute the quality of clusters by comparing distances within and across clusters. The higher the quality of the clusters, the higher the amount of asymmetry between the rankers, which we interpret as misinformation. Next, we develop a measure of noise by finding the variance of ranks for the same object across different rankers. We then show that using these two measures, we are able to find the the correct level of misinformation and noise of rankers for a specific query. We illustrate the benefit of this approach using the statistical model.

8.2 Clustering of Rankers

Given a set of rankers, $r_1, \ldots, r_s$, a greedy approach is used to cluster the rankers into C disjoint clusters. The objective is to cluster rankers with the highest similarity together. Assume that for every pair of rankers (r_i, r_j), $sim(i, j)$ refers to the given measure of similarity between the rankers. We assume $sim(i, j)$ is a

normalized measure. In the following, we use average precision as our similarity measure. However, it is also possible to use $(1-\tau')$ for measure of similarity as well where τ' is a normalized version of Kendall-tau. To cluster, we place each ranker in a different cluster. We then merge clusters with the highest similarity until there are exactly C clusters. The similarity between two clusters C_1 and C_2 is given by the average of all pairwise similarity computations.

$$sim_{AVG}(C_1, C_2) = \frac{1}{|C_1| * |C_2|} \sum_{r_i \in C_1, r_j \in C_2} sim(i, j) \tag{8.1}$$

Note that it is also possible to compute the similarity by finding the maximum similarity between any two rankers in the given two clusters:

$$sim_{MAX}(C_1, C_2) = max\{sim(i, j) | r_i \in C_1, r_j \in C_2\} \tag{8.2}$$

In the following, we test both methods of computing similarity.

Clustering Algorithm. The input to the function *makeClusters* is the set of rankers, $r_1, \ldots, r_s$, similarity function, *sim*, and the target number of clusters, C. The output is the C clusters where each ranker is assigned a cluster identifier. This algorithm allows clusters of size 1. This algorithm makes use of the similarity function, as described above, which could be either the maximum or average similarity.

```
1: function makeClusters({r_1, ..., r_s}, sim, numclusters) returns clusters
2: for i=1 to s do
3:    clusters(i) = i /* mark each ranker with a different cluster */
4: total = s /* number of clusters */
5: while total > numclusters do
6:    bestSim ← 2
7:    idx1 ← 0, idx2 ← 0 /* merge candidates */
8:    for i=1 to total-1 do
9:       for j=i+1 to total do
10:         /* retrieve from clusters those rankers labeled by i or j respectively */
11:         if sim(C_i, C_j) < bestSim then
```

12: $idx1 = i, idx2 = j$

13: $bestSim \leftarrow sim(C_i, C_j)$

14: $k \leftarrow getRankers(idx1)$ /* get all rankers labeled as $idx1$ */

15: $clusters \leftarrow labelRankers(k, idx2)$ /* re-label k as $idx2$

16: total = total - 1

17: **return** *clusters*

8.3 Identification of Misinformation and Noise

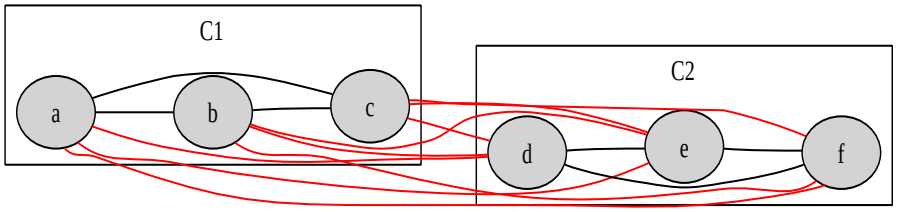


Figure 8.1: width of cluster $C1$ and $C2$ (in black), distance between clusters $C1$ and $C2$ (in gray)

Once the rankers have been grouped into C clusters, we now must evaluate the quality of these rankers as represented by the clusters. We describe two approaches to determine misinformation and noise, which are (1) cluster quality, and (2) variance of ranks. We determine the threshold for misinformation and noise levels using cluster quality and variance.

Our clustering methods places similar rankers in the same cluster. We need to measure how close the clustered objects are. To accomplish this, we compute two values: inter-cluster width and intra-cluster distance. The inter-cluster width computes how far away the rankers in a single cluster are. The smaller the width, the better quality is that specific cluster. The intra-cluster distance measure how far away a pair of clusters are. The larger the distance, the better the cluster quality. We combine these two measures to compute overall quality of clusters as discussed below.

Distance within clusters (width w). Let r_i, r_j be two rankers. Let C denote the set of rankers grouped together in a cluster and $|C|$ represent the size of this

cluster. We compute for all distinct pairs of rankers ($r_i, r_j \in C$), the width of the cluster as follows:

$$width_C = \frac{\sum_{i,j=1,i\neq j}^{|C|}(1 - sim(r_i, r_j))}{\frac{|C|*|C-1|}{2}} \tag{8.3}$$

Distance across clusters (distance *dist*). Let C_1 and C_2 be two clusters. Let $r_L \in C_1$ be the set of rankers that are grouped in cluster C_1 and $r_R \in C_2$ be the set of rankers that are grouped in cluster C_2. We compute for all pairs of rankers from the two clusters (r_L, r_R), the average distance across two clusters as follows:

$$dist_{LR} = \frac{\sum_{L=1}^{|C_1|}\sum_{R=1}^{|C_2|}(1 - sim(r_L, r_R))}{|C_1| * |C_2|} \tag{8.4}$$

The above width and distance equations could be modified to be the maximum width or distance across clusters instead of the average width or distance. In our experimentation, we consider both the maximum and average precision.

Cluster quality. Suppose there are C clusters. Let w_i, w_j be the width of two clusters C_i, C_j, respectively, and $dist_{ij}$ be the distance between these clusters. We can now compute the similarity every distinct pair of clusters, denoted as cluster quality Q, as follows:

$$Q(C) = \sum_{i=1}^{|C|}\sum_{j=i+1}^{|C|}\frac{w_i + w_j}{2 * dist_{ij}} \tag{8.5}$$

Both the width and distance are normalized on a scale [0,1], where values closer to 0 denotes higher similarity. For example, when all rankers are identical and there are two clusters, the width of each cluster is 0 and the distance between each these clusters is 0. In order to handle the zero case (when distance = 0 or width = 0), we assign the insignificant $\epsilon = 0.0001$. The epsilon removes the divide by zero case. When both width and distance are zero, the cluster quality becomes 1, which is the worst case scenario. The cluster quality is considered high if the Q value is low. In addition, epsilon is used for any width zero which may happen for clusters of size greater than 1.

rank	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	r13	r14	r15
1	16	16	16	16	16	16	16	16	6	6	6	6	6	6	6
2	96	96	96	96	96	96	96	96	8	8	8	8	8	8	8
3	8	8	8	8	8	8	8	8	79	79	79	79	79	79	79
4	6	6	6	6	6	6	6	6	95	95	95	95	95	95	95
5	79	63	79	79	79	63	79	63	100	100	100	100	100	100	100
6	63	79	63	63	63	79	63	79	16	16	5	16	16	16	16
7	38	38	38	38	38	38	38	38	5	5	16	5	5	5	5
8	95	95	95	95	95	95	95	95	99	99	99	99	99	99	99
9	92	92	92	92	92	92	92	92	96	96	96	96	96	96	96
10	1	1	45	1	1	1	1	1	62	62	62	62	62	62	62

Table 8.1: top-10 objects from 15 rankers

	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	r13	r14	r15
r1	0	10	9	10	10	10	10	10	6	6	6	6	6	6	6
r2	10	0	9	10	10	10	10	10	6	6	6	6	6	6	6
r3	9	9	0	9	9	9	9	9	6	6	6	6	6	6	6
r4	10	10	9	0	10	10	10	10	6	6	6	6	6	6	6
r5	10	10	9	10	0	10	10	10	6	6	6	6	6	6	6
r6	10	10	9	10	10	0	10	10	6	6	6	6	6	6	6
r7	10	10	9	10	10	10	0	10	6	6	6	6	6	6	6
r8	10	10	9	10	10	10	10	0	6	6	6	6	6	6	6
r9	6	6	6	6	6	6	6	6	0	10	10	10	10	10	10
r10	6	6	6	6	6	6	6	6	10	0	10	10	10	10	10
r11	6	6	6	6	6	6	6	6	10	10	0	10	10	10	10
r12	6	6	6	6	6	6	6	6	10	10	10	0	10	10	10
r13	6	6	6	6	6	6	6	6	10	10	10	10	0	10	10
r14	6	6	6	6	6	6	6	6	10	10	10	10	10	0	10
r15	6	6	6	6	6	6	6	6	10	10	10	10	10	10	0

Table 8.2: precision similarity matrix

Example. Let's look at an example when $n_{MI} = 7$ and $\sigma^2 = 0.10$, meaning 7 of the 15 rankers use the reverse weight function. In Table 8.1, the first 10 objects of the 15 rankers are shown and Table 8.2 displays the precision between each pair of rankers. In the case of two clusters, cluster1 (C_1) groups rankers r1-r8 and cluster2

(C_2) groups r9-r15. The widths, distance and cluster quality are:

$$\begin{aligned} w_1 &= 0.10 \\ w_2 &= 0.0001 \\ dist_{12} &= 0.3625 \\ Q(C_1, C_2) &= \frac{0.1+0.0001}{2*0.3625} = 0.1379 \end{aligned}$$

Variance of Ranks. While the cluster quality is used to measure the misinformation, we introduce the variance of ranks measure to compute the amount of noise in the rankers. Let n be the distinct number of objects in the input rankers. Let $rank(r_j, o_i)$ be the rank of o_i in ranker r_j. For each cluster C, we compute the mean of the variance in the ranks over all objects. Note that if an object is not ranked by a ranker then we assign it the default rank of K+1 where K is the current retrieval size. If two rankers are missing the same objects, the objects would reduce the variance measure, indicating an agreement between the two rankers in line with the intended meaning of this measure.

1: $ranks \leftarrow$ empty /* n-by-s */
2: $vars \leftarrow$ empty /* 1-by-n */
3: $Var \leftarrow$ empty
4: **for** i=1 to n **do**
5: **for** j=1 to s **do**
6: $ranks(i, j) = rank(o_i, r_j)$
7: $vars(i) = var(ranks(\mathbf{i}))$ /* variance of ranks for o_i */
8: $Var = mean(\mathbf{vars})$ /* avg. variance over all objects */
9: **return** Var

8.4 Experimental Evaluation

In this section, we describe experiments that illustrate the validity of our approach for determining noise and misinformation. Our aim is to use the statistical model with different noise and misinformation settings and determine whether we can identify boundary conditions using our methods for different scenarios. We will

	$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$	$n_{MI} = 8$	$n_{MI} = 13$
$\sigma^2 = 7.5$	PrOpt, CombMNZ Pg PgADJ	PrOpt, CombMNZ Pg PgADJ	PrOpt, Pg, CombMNZ PgADJ PgIBF, Cfuse	PrOpt, Pg, CombMNZ PgADJ Cfuse	Pg, CombMNZ, PrOpt, Cfuse PgADJ PgIBF
$\sigma^2 = 5.0$	PrOpt CombMNZ Pg	PrOpt, CombMNZ Pg PgADJ	Cfuse PrOpt, CombMNZ Pg	Cfuse Pg PrOpt, CombMNZ	Av AvADJ Pg
$\sigma^2 = 1.0$	CombMNZ Pg*, PrOpt AvADJ	PgADJ, PrOpt, Cfuse, CombMNZ Pg, PgIBF, MeIBF RndIBF	Me MeADJ PrOpt	Av Pg CombMNZ	Av AvADJ Me
$\sigma^2 = 0.10$	Av, AvIBF PgADJ, MeADJ, CombMNZ AvADJ, PgIBF, PrOpt, Cfuse	PrOpt, Cfuse PgADJ, CombMNZ Pg, PgIBF, MeIBF	Me PrOpt MeADJ	Av Pg CombMNZ	Av AvADJ AvIBF

(a) precision results

	less misinformation				more misinformation
high noise	PrOpt, CombMNZ Pg MeIBF	PrOpt, CombMNZ Pg MeIBF, RndIBF	MeIBF RndIBF PrOpt, CombMNZ	MeIBF RndIBF PrOpt, Pg, CombMNZ	Pg PgADJ, MeIBF PrOpt, RndIBF, CombMNZ
	Pg CombMNZ PrOpt	PgADJ PgIBF Pg	PgADJ, MeIBF PgIBF RndIBF	Pg PrOpt, CombMNZ PgADJ	Av Pg, AvADJ PrOpt
	Pg Av AvIBF, PgIBF	Cfuse PgADJ MeIBF	MeADJ Me Cfuse	Av Pg CombMNZ	Av PrOpt CombMNZ
low noise	Av, Pg *ADJ, *IBF, PrOpt, Cfuse, CombMNZ Me	Cfuse MeIBF MeADJ	Me MeADJ Cfuse	Av Pg CombMNZ	PrOpt CombMNZ Av

(b) Kendall-tau results

Figure 8.2: Summary of results for the baseline case with 15 input rankers. Refer to Figure 7.1 for the full description of these tables.

also use these boundary conditions to decide which rank aggregation method to use. We assume that queries in our study will consist of mostly correct information, in which the number of rankers using the reverse weight function is less than the number of rankers using the true weight function. In our setup, we keep the number of factors set to 5 and the noise (σ^2) vary between $\langle 0.10, 1.0, 5.0, 7.5 \rangle$. We increase the number of rankers to 15. The 15 input rankers gives us ample information to classify both noise and misinformation. We also changed the misinformation levels to be $n_{MI} = \langle 0, 2, 7 \rangle$, where $n_{MI} = 2$ means two rankers are using the reverse weight function. As a reminder, the (ground truth) weight function is $\mathbf{w} = \langle \frac{1}{15}, \frac{2}{15}, \frac{3}{15}, \frac{4}{15}, \frac{5}{15} \rangle$ and the reverse weight function is $\mathbf{w}' = \langle \frac{5}{15}, \frac{4}{15}, \frac{3}{15}, \frac{2}{15}, \frac{1}{15} \rangle$.

We investigate the impact of our approach by varying K (the number of objects returned by the rankers) $= \langle 10, 20, 30 \rangle$. Due to this change in the number of input rankers and varying of K, we execute our baseline experiments again (e.g. no correlation and no missing information for the 13 aggregators). We want to see whether or not there is a difference in the top aggregators across different selections of K and number of rankers. We also vary the number of clusters C in our experiments, which we assign $C = \langle 2, 3 \rangle$. When $C = 2$ and low noise level, there is a distinct partitioning of the rankers. However when $C = 3$, there are two clusters that are nearly identical with the remaining cluster equi-distance from the other two clusters. Since we vary both K and C, we have six sets of experiments for deriving the ranker information.

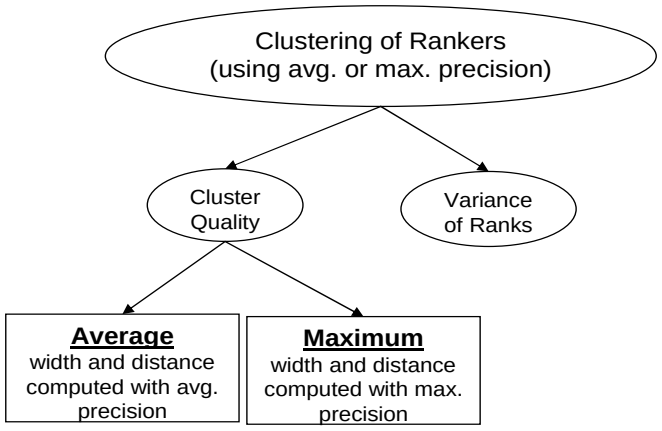


Figure 8.3: Flow chart of misinformation and noise evaluation

We compute the cluster quality for each combination of clusters and variance of ranks for each cluster. We show a flow chart in Figure 8.3 to identify the different types of evaluation methods used. We begin by clustering the rankers. As mentioned previously, these clusters are selected using either the average or maximum precision. Now the cluster quality can be calculated by computing the widths and distances using either the average or maximum precision. As we form clusters using the maximum or average precision, we also compute the cluster quality using the maximum or average precision. Also, the rank variance for each cluster can be computed, which is independent of the width and distance calculations. Given each K and C, there are six result sets, three result sets using average precision to cluster and three result sets using maximum precision to cluster.

$\sigma^2 = 7.5$	(min)	0.2517483	0.2142998	0.241482
	(mean)	0.6339355	0.6384475	0.7126436
	(max)	0.9504219	0.9529506	0.9458219
$\sigma^2 = 5.0$	(min)	0.1873687	0.0985956	0.2334417
	(mean)	0.5792157	0.5564963	0.5867503
	(max)	0.9281250	0.9049145	0.9067086
$\sigma^2 = 1.0$	(min)	0.0005000	0.0001000	0.0001111
	(mean)	0.4100831	0.1329626	0.1334516
	(max)	1.0000000	0.6666709	0.5804870
$\sigma^2 = 0.10$	(min)	0.0010000	0.0001000	0.0001000
	(mean)	0.7756248	0.0157210	0.0158917
	(max)	1.0000000	0.4231538	0.3100234
		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$

Table 8.3: Average cluster quality of top-10 objects, 2 clusters, using average precision to cluster. The x-axis denotes the misinformation (n_{MI}) when there are 15 input rankers. The y-axis denotes the noise (σ^2). Each misinformation and noise case displays the minimum, mean and maximum cluster quality observed in the 40,000 datasets.

We then report the minimum, maximum and mean values for the cluster quality and rank variance. We display in Tables 8.3 - 8.6 the results from taking the top-10 objects and forming two and three clusters using average precision. The general trend we observe shows that it is hard to distinguish between $n_{MI} = 2$ and $n_{MI} = 7$ cases using this measure. As expected, the measure does not depend

$\sigma^2 = 7.5$	(min)	0.000000	0.000000	0.000000
	(mean)	5.298782	6.464934	6.165487
	(max)	22.133333	22.312500	21.937500
$\sigma^2 = 5.0$	(min)	0.000000	0.000000	0.000000
	(mean)	3.514492	5.281215	4.565962
	(max)	20.933333	22.333333	18.428570
$\sigma^2 = 1.0$	(min)	0.000000	0.000000	0.000000
	(mean)	0.639310	0.746333	0.718639
	(max)	7.750000	11.000000	6.853896
$\sigma^2 = 0.10$	(min)	0.000000	0.000000	0.000000
	(mean)	0.043294	0.0513666	0.051265
	(max)	1.727273	3.218182	1.282468
		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$

Table 8.4: Variance of ranks of top-10 objects, 2 clusters, using average precision to cluster. The x-axis denotes the misinformation (n_{MI}) when there are 15 input rankers. The y-axis denotes the noise (σ^2). Each misinformation and noise case displays the minimum, mean and maximum rank variance observed in the 40,000 datasets.

$\sigma^2 = 7.5$	(min)	0.0001000	0.0001000	0.0001000
	(mean)	0.5741046	0.5233971	0.6221042
	(max)	0.9523810	0.9600000	0.9600000
$\sigma^2 = 5.0$	(min)	0.0001000	0.0001000	0.0001000
	(mean)	0.5128798	0.4105165	0.5046919
	(max)	0.9339975	0.9459459	0.9411765
$\sigma^2 = 1.0$	(min)	0.0002500	0.0001000	0.0001000
	(mean)	0.3134411	0.1627706	0.1850388
	(max)	1.0000000	1.0000000	1.0000000
$\sigma^2 = 0.10$	(min)	0.0005000	0.0001000	0.0001000
	(mean)	0.8299089	0.2407070	0.2214130
	(max)	1.0000000	1.0000000	1.0000000
		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$

Table 8.5: Average cluster quality of top-10 objects, 3 clusters, using average precision to cluster. Refer to Table 8.3 for a full description of the table.

$\sigma^2 = 7.5$	(min)	0.000000	0.000000	0.000000
	(mean)	5.828739	5.857973	6.653365
	(max)	22.333333	22.500000	22.50000
$\sigma^2 = 5.0$	(min)	0.000000	0.000000	0.000000
	(mean)	3.604112	3.751862	4.366549
	(max)	22.250000	21.571430	21.500000
$\sigma^2 = 1.0$	(min)	0.000000	0.000000	0.000000
	(mean)	0.550212	0.580480	0.642011
	(max)	8.909091	13.090910	8.750000
$\sigma^2 = 0.10$	(min)	0.000000	0.000000	0.000000
	(mean)	0.018619	0.021374	0.033739
	(max)	2.181818	1.000000	1.316667
		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$

Table 8.6: Variance of ranks of top-10 objects, 3 clusters, using average precision to cluster. Refer to Table 8.4 for a full description of the table.

on the size of the clusters but the degree of agreement and disagreement between rankers. As the two different types of rankers have the same weights in both cases, the cluster quality should be the same in both cases. However, there is a noticeable difference between $n_{MI} = 0$ case and the other cases. Hence, the measure is useful for detecting the presence of asymmetry between rankers. We note however that the cluster quality becomes more or less the same for all misinformation cases as the noise increases. In these cases, noise masks the misinformation. As noise increases, the number of common objects among the rankers decrease and hence there is less rank information to use when clustering objects and the rank values are less reliable. As a result, it is harder to use cluster quality as a way to distinguish between different misinformation cases.

One thing we notice is that for $n_{MI} = 0$, the cluster quality drops for moderate noise levels. This is probably due to grouping high and low noise rankers in two different clusters. However, the cluster quality still remains a decisive factor for determining whether misinformation exists. As noise increases further, all rankers appear the same and hence the cluster quality increases again.

Now, we discuss particular influences of imposing different clustering approaches, different retrieval sizes (K) and different cluster sizes (C).

Effect of average precision vs. maximum precision in clustering. When using max precision, we decide to include a ranker in a cluster if it is very similar to one of the rankers. This may result in clusters where some rankers are not very similar to each other. In average precision, we include a new ranker in a cluster based on the average distance to the existing rankers. In this case, two different clusters may appear the same with respect to this measure due to the averaging, making the clusters more uncertain. When using the maximum precision in deciding the clusters, the average cluster quality and rank variances increase slightly. The exception to this generalization is when $\sigma^2 = 0.10$, in which the cluster quality and rank variance are unaffected. When $\sigma = 1.0$, the differences are very low, e.g. 2% in cluster quality and less than 1 rank position in rank variance. As expected, low noise does not significantly impose wrong ordering of the objects; thus the cluster quality and rank variance does not vary significantly. For $n_{MI} = 0$, the cluster quality and rank variance is the most affected as σ^2 increases. The largest difference between using the average precision to cluster and using the maximum precision to cluster is approximately a 20% change in the cluster quality. Since there is no misinformation, the misordering of objects is a direct result of the noise level, which is observed in this large discrepancy. For the remaining cases, $n_{MI} = 2, 7$ and $\sigma^2 = 5.0, 7.5$, less misinformation ($n_{MI} = 2$) has lower differences than in more misinformation ($n_{MI} = 7$) as well as lower noise ($\sigma^2 = 5.0$) has lower differences than in the high noise ($\sigma^2 = 7.5$).

Effect of retrieval size K. When increasing K, the rank variance increases since the number of possible ranks increase. For two clusters when $n_{MI} = 0$ and $\sigma^2 = 0.10$, the cluster quality decreases signifying that there are decreasing number of identical rankers. Recall that we set $\delta = 5.0$ and $\beta = 0.01$, which means that there are greater errors added to the factors as we proceed down the ranker. With the increase of K, the δ and β are contributing to the ordering of the objects as well as the noise level, hence the cluster quality does not reach 1. In the other misinformation and noise levels, the cluster quality increases by a small percentage of approximately 2% for each increase in K. For three clusters, we observe the decrease in cluster

quality when $\sigma^2 = 0.10$ in all misinformation levels. The largest decrease occurs when $n_{MI} = 0$ and there are 3 clusters, which has the average cluster quality of 0.829 (top-10), 0.681 (top-20) and 0.569 (top-30). For the higher noise levels, the cluster quality increases. The largest increase occurs for 3 clusters when $n_{MI} = 0$ and $\sigma^2 = 1.0$, which has the cluster quality of 0.31 (top-10), 0.40 (top-20) and 0.45 (top-30).

Effect of cluster size C. In our tests, there are only two true clusters. When we try to find three clusters in low noise cases, two clusters contain more or less identical items. For cluster quality, we see that the mean values decrease from two clusters to three clusters for the higher noise cases. The range in cluster quality for three clusters is larger than with two clusters. The rank variance decreases from two clusters to three clusters for all misinformation and noise cases, except for $\sigma^2 = \langle 5.0, 7.5 \rangle, n_{MI} = 0$. In those cases, the increase in rank variance is small, e.g. less than one rank position.

Given change in the number of input rankers, we repeat an experiment for the best aggregator using 15 input rankers and top-10 objects. The results of these experiments are shown in Figure 8.2. We consider the first 3 columns of each table indicating that the majority of the input rankers are applying the true weight function. For both precision and Kendall-tau, we calculate the best aggregator over these 12 cells. However, in the case where there is more than one aggregator for a rank, the next ranked aggregator received the next available rank. For instance when considering the Kendall-tau results for $\sigma^2 = 7.5$ and $n_{MI} = 2$, both PrOpt and CombMNZ have rank 1, Pg has rank 3, MeIBF and RndIBF have rank 4 and the remaining aggregators have rank 6. For precision, the top aggregators over the 12 cells are CombMNZ, PrOpt and PgADJ. For Kendall-tau, the top aggregators over the 12 cells are Pg, MeIBF and CombMNZ. We use these aggregators as the static aggregators when trying to choose the best aggregator. We want to prove that using the best aggregator in each cell is better than using one of these static aggregators in terms of precision and Kendall-tau. We select the best aggregator for each class, which we show in Figure 8.4.

$\sigma^2 = 7.5$	CombMNZ	PrOpt	PrOpt
$\sigma^2 = 5.0$	PrOpt	PrOpt	Cfuse
$\sigma^2 = 1.0$	CombMNZ	PgADJ	Me
$\sigma^2 = 0.1$	Av	PrOpt	Me
	$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$

(a)

$\sigma^2 = 7.5$	PrOpt	PrOpt	MeIBF
$\sigma^2 = 5.0$	Pg	PgADJ	PgADJ
$\sigma^2 = 1.0$	Pg	Cfuse	MeADJ
$\sigma^2 = 0.1$	Av	Cfuse	Me
	$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$

(b)

Figure 8.4: (a) Best aggregators by class using precision results, (b) Best aggregators by class using Kendall-tau results

Choosing the best aggregator. In order to choose the best method to aggregate the input rankers, we apply the naive Bayesian supervised learning model. This model has two phases. The first phase is to determine the probability of a query or a set of input rankers belong to a specific misinformation-noise case. Also, we have to consider the penalty associated with misclassifying a query. The second phase applies the computations of the first phase to new queries.

We call each misinformation-noise case a *class*, we have 12 classes. We identify a numerical class identifier in the table below.

Within each class, we generate 40,000 datasets. We compute and record the cluster quality and rank variance from each dataset, which we call a 2-d point (x_1, x_2). For each class with 40,000 points, we compute the mean and covariance of (x_1, x_2). We will assume that each class of data sets can be presented as a 2-d Gaussian with the given mean and covariance for this class. We must also understand the penalty of misclassifying a data point. To illustrate this, suppose

$\sigma^2 = 7.5$	4	8	12
$\sigma^2 = 5.0$	3	7	11
$\sigma^2 = 1.0$	2	6	10
$\sigma^2 = 0.1$	1	5	9
	$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$

we misclassify a set of rankers with true class c_i in another class c_j. Then, there is a cost involved with this decision. In this case, instead of using the top aggregator in c_i, we will end up using the top aggregator A_j in c_j. We have to measure how bad the aggregator A_j would do in class c_i which is the true class for this data set. We use a cost matrix (12x12 matrix) that works as follows. For each class c_i, c_j, we compute the penalty of misclassification

1. if the true class of the point is c_i and
2. if the selected class of the point is c_j.

We can apply any performance measure to use as the penalty cost. Let aggregator A_i be the top aggregator for class c_i and aggregator A_j be the top aggregator for class c_j. In the case of Kendall-tau, we compute $\tau_i(A_i)$ which is the average Kendall-tau of aggregator A_i in class c_i, and $\tau_i(A_j)$ which is the average Kendall-tau of aggregator A_j in class c_i. Each entry in the cost matrix is $Cost(c_i, c_j) = \tau_i(A_j) - \tau_i(A_i)$ and then normalized such that $\sum_j Cost(c_i) = 1$. In the case of precision, we compute $pr_i(A_i)$ which is the average precision of aggregator A_i in class c_j and $pr_i(A_j)$ which is the average precision of aggregator A_j in class c_i. The cost matrix has entries of $Cost(c_i, c_j) = pr_i(A_i) - pr_i(A_j)$.

For each new query (dataset), we perform the following:

1. Generate a ground truth ranker and input rankers and compute the new cluster quality and rank variance, x_1^*, x_2^*
2. Compute the best class for x_1^*, x_2^* by applying the naive Bayes theorem
3. Select the lowest cost computation
4. Aggregate using the top aggregator for the selected class
5. Compare the performance of the best class aggregator with each static aggregator in terms of both precision and Kendall-tau performance measures

We first use the statistical model to produce the ground truth ranker and input rankers. We cluster the rankers and compute the cluster quality and rank variance.

Next, we use the naive Bayes theorem to model each class's discrete conditional probability distribution. Since we assume continuous values, we use a conditional Gaussian distribution which is estimated by computing the mean (μ_i) and covariance (Σ_i) from the first phase for each class c_i. We compute the probability of x_1^*, x_2^* belonging to class c_i using the formula

$$prob(x_1^*, x_2^*|c_i) = \frac{1}{2\pi * \det \Sigma_i^{1/2}} e^{-\frac{1}{2}(x-\mu_i)'\Sigma_i^{-1}(x-\mu_i)}$$

We are in fact interested in $prob(c_i|x_1^*, x_2^*)$. By Bayesian theorem, we can write this as:

$$prob(c_i|x_1^*, x_2^*) = prob(x_1^*, x_2^*|c_i) * \frac{prob(c_i)}{prob(x_1^*, x_2^*)}$$

As $prob(x_1^*, x_2^*)$ is common to all the classes, it is simply a scaling factor. Similarly, we assume that each class is equally likely in our simulation. Hence, $prob(c_i) = 1/12$. As a result, we can simply write:

$$prob(c_i|x_1^*, x_2^*) \propto prob(x_1^*, x_2^*|c_i)$$

As we want to find the most likely class, we are simply using the $prob(x_1^*, x_2^*|c_i)$ values to order the classes. As a result, we do not need to compute the real value of $prob(c_i|x_1^*, x_2^*)$. Hence, we use $prob(x_1^*, x_2^*|c_i)$ in our class decision computations instead of $prob(c_i|x_1^*, x_2^*)$. We can compute the best class for point (x_1^*, x_2^*) for each class using the decision cost given below:

$$DecisionCost(c_i) = \sum_j p(x_1^*, x_2^*|c_i) * Cost(c_j, c_i)$$

The lowest decision cost identifies the estimated class in which that class's aggregator is executed to determine the error.

We compute the performance of the static and top aggregators with respect to the ground truth r. Let A denote the selected top aggregator and B denote any static aggregator. For the precision performance measure, we compute the

performance using

$$average(\frac{pr(r,A) - pr(r,B)}{pr(r,B)})$$

and the performance evaluation for Kendall-tau is slightly modified and becomes

$$average(\frac{\tau(r,B) - \tau(r,A)}{\tau(r,B)})$$

. In terms of precision, we want to maximize while in Kendall-tau we want to minimize. In the case where the performances are equivalent or the benefit ratio has a denominator is zero, the performance should be fixed to 0 (for precision) and 1 (for Kendall-tau).

We also consider the accuracy of the classifier. The ideal situation would be to have all new points correctly classified into the associated classes but as we see with the cluster quality and rank variance values, this may be hard in some cases. However, we can identify how wrong the naive Bayes classifier performed. If the true and selected class are the same, then there is no misclassification. However, in the case where the true and selected class are in different classes, this may significantly degrade the performance.

8.5 Analysis and Discussion

We conduct four experiments in order to evaluate the performance improvements using Bayes theorem and the accuracy associated with this cost-based classification method. In the first experiment, we run a optimal classification approach in which we use the top aggregator in each class, as outlined in Figure 8.4. We compare this optimal classification approach to each of the static aggregators. This experiment shows the best performance achievable with respect to each static aggregator. In the second experiment, we run the Bayesian classification technique. We then compare this classification method to each of the static aggregators. We consider the minimum, maximum and average performance for each class. The minimum value refers to the best performance of the static aggregator. The maximum value refers to the best performance of the optimal classification or cost-based classification approach. The third experiment computes the average performance over all

classes, not individually as we do in the first two experiments. The last experiment examines the accuracy of the cost-based classification classifier. We now discuss the results of these experiments with respect to precision and Kendall-tau.

Optimizing performance using precision. We display in Figures 8.5-8.7 the optimal classification and cost-based classification performance for each static aggregator with the objective of maximizing the precision. As we compare each class's mean performance of both the optimal classification and cost-based classification approach, we show the how close the cost-based classification method reaches the optimal classification method. In Figure 8.5, we see that the optimal classification approach for class $\sigma^2 = 0.1, n_{MI} = 7$ is 0.227 while the cost-based classification approach reaches 0.223. In this case, the cost-based classification approach does not reach the best possible performance. When $\sigma^2 = 7.5, n_{MI} = 0$, the cost-based classification approach has the same performance as the optimal classification approach. In this case, CombMNZ is the designated rank aggregation method thus there is no performance improvement in the optimal classification approach, but the cost-based classification approach misclassifies the query into other classes, which actually makes better decisions.

In Figure 8.6, PrOpt is the designated rank aggregation in 4 of the 12 classes, which shows no performance improvements. When PgADJ is the static aggregator (Figure 8.7), we see some classes that show performance improvements using the cost-based classification approach but do not reach the performance achieved by applying the optimal classification method. In the table below, we display the average precision over the 12 classes. We find the average precision for each class and then compute the mean of the average precision. We observe that the cost-based classification approach performs close to the optimal classification approach. Also, the static aggregator PrOpt that rivals both the cost-based classification and optimal classification approaches. The cost-based classification approach does not strongly outperform PrOpt since PrOpt is the top aggregator in several classes.

When executing the cost-based classification approach, we also record the accuracy count for each class. Each row in the table below gives the number of

		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$
$\sigma^2 = 7.5$	min	0.000	-0.333	-0.500
	mean	0.000	0.000	0.000
	max	0.000	0.333	0.500
$\sigma^2 = 5.0$	min	-0.143	-0.167	-0.375
	mean	0.000	0.000	0.003
	max	0.250	0.167	0.750
$\sigma^2 = 1.0$	min	0.000	-0.125	-0.250
	mean	0.000	0.001	0.103
	max	0.000	0.250	0.667
$\sigma^2 = 0.1$	min	-0.100	-0.100	-0.100
	mean	0.000	0.000	0.227
	max	0.111	0.111	0.667

(a) optimal classification, best aggregator in each class

		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$
$\sigma^2 = 7.5$	min	-0.200	-0.250	-0.333
	mean	0.000	0.000	0.000
	max	0.333	0.333	0.333
$\sigma^2 = 5.0$	min	-0.167	-0.143	-0.200
	mean	0.000	0.000	0.000
	max	0.167	0.200	0.333
$\sigma^2 = 1.0$	min	-0.111	-0.111	-0.143
	mean	0.000	0.000	0.079
	max	0.125	0.125	0.667
$\sigma^2 = 0.1$	min	-0.100	-0.100	-0.111
	mean	0.000	0.000	0.223
	max	0.111	0.111	0.667

(b) cost-based classification

Figure 8.5: Precision performance results using CombMNZ as a static aggregator. The x-axis denotes the misinformation (n_{MI}) when there are 15 input rankers. The y-axis denotes the noise (σ^2). Each misinformation and noise case displays the minimum, mean and maximum precision performance improvement observed in the 40,000 queries. Part (a) displays the improvement when all queries are correctly classified in terms of misinformation and noise and part (b) displays the improvement when using Bayes Theorem.

		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$
$\sigma^2 = 7.5$	min	-0.250	0.000	0.000
	mean	0.000	0.000	0.000
	max	0.333	0.000	0.000
$\sigma^2 = 5.0$	min	0.000	0.000	-0.375
	mean	0.000	0.000	0.003
	max	0.000	0.000	0.750
$\sigma^2 = 1.0$	min	-0.111	-0.125	-0.250
	mean	0.000	0.001	0.022
	max	0.125	0.250	0.600
$\sigma^2 = 0.1$	min	-0.100	0.000	-0.100
	mean	0.000	0.000	0.003
	max	0.111	0.000	0.429

(a) optimal classification, best aggregator in each class

		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$
$\sigma^2 = 7.5$	min	-0.200	-0.333	-0.500
	mean	0.000	0.000	0.000
	max	0.250	0.250	0.200
$\sigma^2 = 5.0$	min	-0.143	-0.125	-0.143
	mean	0.000	0.000	0.000
	max	0.143	0.143	0.125
$\sigma^2 = 1.0$	min	-0.111	-0.100	-0.125
	mean	0.000	0.000	0.000
	max	0.125	0.111	0.125
$\sigma^2 = 0.1$	min	-0.100	-0.100	-0.200
	mean	0.000	0.000	0.000
	max	0.111	0.111	0.125

(b) cost-based classification

Figure 8.6: Precision performance results using PrOpt as a static aggregator. Refer to Figure 8.5 for a full description.

optimal classification	8.5518
cost-based classification	8.5364
CombMNZ	8.3385
PrOpt	8.5360
PgADJ	8.4466

		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$
$\sigma^2 = 7.5$	min	-0.333	-0.250	-0.500
	mean	0.006	0.005	0.005
	max	0.333	0.500	0.500
$\sigma^2 = 5.0$	min	-0.167	-0.222	-0.375
	mean	0.006	0.005	0.006
	max	0.286	0.333	0.750
$\sigma^2 = 1.0$	min	-0.111	0.000	-0.250
	mean	0.001	0.000	0.060
	max	0.125	0.000	0.667
$\sigma^2 = 0.1$	min	-0.100	-0.100	-0.100
	mean	0.000	0.000	0.077
	max	0.111	0.111	0.429

(a) optimal classification, best aggregator in each class

		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$
$\sigma^2 = 7.5$	min	-0.333	-0.333	-0.500
	mean	0.006	0.005	0.005
	max	0.333	0.500	0.500
$\sigma^2 = 5.0$	min	-0.222	-0.250	-0.250
	mean	0.006	0.005	0.005
	max	0.333	0.333	0.400
$\sigma^2 = 1.0$	min	-0.111	-0.125	-0.250
	mean	0.001	0.000	0.037
	max	0.125	0.250	0.800
$\sigma^2 = 0.1$	min	-0.100	-0.100	-0.111
	mean	0.000	0.000	0.073
	max	0.111	0.111	0.667

(b) cost-based classification

Figure 8.7: Precision performance results using PgADJ as a static aggregator. Refer to Figure 8.5 for a full description.

new queries that are classified in each class, e.g. the sum of each row is therefor 40,000. In the case of class 1, nearly all queries are correctly identified as being class 1. In the case of class 2 and 4, CombMNZ is the top aggregator and the majority of the cost-based classification is evenly divided. Since class 3, 5, 7, 8 and 12 have the same top aggregator (PrOpt), a majority of new queries are evenly distributed across these classes and are correctly identified. We notice that classes 6, 9 and

10 are never selected as a new query's class. These classes use the top aggregators of PgADJ or Me, which have a high penalty value (e.g. a high probability in the normalized cost matrix) with respect to the other aggregators. This means that the decision cost associated with these classes are too high to be selected.

	1	2	3	4	5	6	7	8	9	10	11	12
1	29985	904	1463	880	1548	0	1597	1587	0	0	382	1654
2	514	13883	2385	13672	2324	0	2281	2451	0	0	76	2414
3	0	5211	5885	5101	5873	0	5907	6063	0	0	0	5960
4	0	2110	7170	2189	7175	0	7118	7124	0	0	0	7114
5	0	13	7443	17	7597	0	7547	7745	0	0	1969	7669
6	0	452	7882	440	7827	0	7734	7733	0	0	37	7895
7	0	507	7893	501	7733	0	7845	7741	0	0	0	7780
8	0	1424	7311	1431	7543	0	7459	7494	0	0	0	7338
9	0	3	7726	2	7904	0	7845	7908	0	0	883	7729
10	0	181	8045	162	7892	0	8090	7824	0	0	10	7796
11	0	233	7993	236	7794	0	7947	7954	0	0	0	7843
12	0	603	7869	582	7744	0	7813	7782	0	0	0	7607

Optimizing performance using Kendall-tau. We show in Figures 8.8-8.10 performance improvements for the Kendall-tau performance measure. In Kendall-tau, we consider the number of disagreements, in which the values refer to the swap difference between the static and non-static aggregators.

We now look at the performance results for Kendall-tau. When looking at the mean, the negative values indicate that the static aggregator has better Kendall-tau count and by how much on the average. We should note that negative values can be misleading. We see in a majority of classes with the minimum value is much further from zero than the maximum value. A negative mean value are indicative of larger negative values and smaller positive values. The negative mean values are relatively small implying that the two aggregators under comparison are roughly equivalent.

In Figure 8.8 when $\sigma^2 = 7.5, n_{MI} = 2$, the optimal classification mean values has the same performance as the cost-based classification approach. In another case when $\sigma^2 = 0.1, n_{MI} = 7$, the optimal classification mean is 0.925 and cost-based classification mean is 0.915. Once again, the cost-based classification approach does not

		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$
	min	-1.000	-2.000	-1.267
$\sigma^2 = 7.5$	mean	0.002	-0.001	-0.001
	max	0.667	0.333	0.778
	min	0.000	-7.000	-1.000
$\sigma^2 = 5.0$	mean	0.000	0.015	0.038
	max	0.000	0.875	0.833
	min	0.000	-5.000	-0.692
$\sigma^2 = 1.0$	mean	0.000	0.369	0.622
	max	0.000	0.900	0.974
	min	-1.000	-2.000	0.000
$\sigma^2 = 0.1$	mean	0.000	0.416	0.926
	max	0.667	0.889	0.974

(a) optimal classification, best aggregator in each class

		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$
	min	-2.000	-3.000	-0.600
$\sigma^2 = 7.5$	mean	-0.002	-0.001	0.001
	max	0.500	0.750	0.500
	min	-4.000	-6.000	-1.000
$\sigma^2 = 5.0$	mean	-0.011	0.014	0.033
	max	0.833	0.889	0.750
	min	-5.000	-4.000	-0.556
$\sigma^2 = 1.0$	mean	-0.020	0.361	0.575
	max	0.750	0.900	0.973
	min	-1.000	-5.000	0.000
$\sigma^2 = 0.1$	mean	0.000	0.409	0.915
	max	0.500	0.889	0.974

(b) cost-based classification

Figure 8.8: Kendall-tau performance results using Pg as a static aggregator. The x-axis denotes the misinformation (n_{MI}) when there are 15 input rankers. The y-axis denotes the noise (σ^2). Each misinformation and noise case displays the minimum, mean and maximum Kendall-tau performance improvement observed in the 40,000 queries. Part (a) displays the improvement when all queries are correctly classified in terms of misinformation and noise and part (b) displays the improvement when using Bayes Theorem.

		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$
$\sigma^2 = 7.5$	min	-3.000	-4.000	0.000
	mean	0.023	0.005	0.000
	max	0.875	0.800	0.000
$\sigma^2 = 5.0$	min	-6.000	-5.000	-2.667
	mean	0.033	-0.002	-0.012
	max	0.917	0.875	0.500
$\sigma^2 = 1.0$	min	-5.000	-6.000	-12.000
	mean	0.014	-0.013	0.124
	max	0.857	0.875	0.941
$\sigma^2 = 0.1$	min	-2.000	-6.000	-8.000
	mean	0.000	0.002	0.202
	max	0.667	0.857	0.970

(a) optimal classification, best aggregator in each class

		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$
$\sigma^2 = 7.5$	min	-2.000	-6.000	-2.500
	mean	-0.005	-0.002	-0.001
	max	0.889	0.889	0.667
$\sigma^2 = 5.0$	min	-10.000	-7.000	-4.000
	mean	-0.012	0.052	0.033
	max	0.889	0.864	0.539
$\sigma^2 = 1.0$	min	-5.000	-7.000	-13.500
	mean	-0.028	0.582	0.522
	max	0.857	0.875	0.955
$\sigma^2 = 0.1$	min	-2.000	-5.000	-11.000
	mean	-0.001	0.698	0.893
	max	0.667	0.857	0.963

(b) cost-based classification

Figure 8.9: Kendall-tau performance results using MeIBF as a static aggregator. Refer to Figure 8.8 for a full description.

reach the full improvement associated with that class. When MeIBF and CombMNZ are considered static, we observe similar trends that the cost-based classification method shows improvements but does not reach the best achievable. In the table below, we show the average Kendall-tau over all 12 classes. We notice that the cost-based classification method clearly outperforms the static aggregators by producing a lower Kendall-tau.

		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$
$\sigma^2 = 7.5$	min	-0.500	-0.667	-1.429
	mean	0.000	-0.001	-0.002
	max	0.333	0.200	0.750
$\sigma^2 = 5.0$	min	-4.000	-6.000	-1.000
	mean	-0.005	0.052	0.039
	max	0.857	0.917	0.833
$\sigma^2 = 1.0$	min	-3.000	-5.000	-0.750
	mean	-0.013	0.591	0.575
	max	0.833	0.933	0.966
$\sigma^2 = 0.1$	min	-2.000	-1.000	-0.222
	mean	0.000	0.703	0.907
	max	0.667	0.923	0.971

(a) optimal classification, best aggregator in each class

		$n_{MI} = 0$	$n_{MI} = 2$	$n_{MI} = 7$
$\sigma^2 = 7.5$	min	-2.000	-2.000	-0.667
	mean	-0.005	-0.002	-0.001
	max	0.500	0.833	0.500
$\sigma^2 = 5.0$	min	-4.000	-6.000	-1.250
	mean	-0.012	0.052	0.033
	max	0.857	0.923	0.857
$\sigma^2 = 1.0$	min	-7.000	-5.000	-0.706
	mean	-0.028	0.582	0.522
	max	0.857	0.938	0.969
$\sigma^2 = 0.1$	min	-2.000	-5.000	-0.222
	mean	-0.001	0.698	0.893
	max	0.667	0.917	0.969

(b) cost-based classification

Figure 8.10: Kendall-tau performance results using CombMNZ as a static aggregator. Refer to Figure 8.8 for a full description.

optimal classification	18.869
cost-based classification	19.091
Pg	21.313
MeIBF	19.384
CombMNZ	22.115

As we did with the precision, we record the accuracy count for each each class. Each row in the table below gives the number of new queries that are classified in each class. Once again, class 1 correctly identify a new query's class in a majority of cases. Both class 2 and 3 have the top aggregator of Pg and for a majority of new queries, class 2 or 3 are correctly identified. When PrOpt is the top aggregator, classes 4 and 8, the cost-based classification approach misclassifies as being class 2 or 3. Both classes 5 and 6 uses the Cfuse rank aggregation method; however class 5 is typically misclassified as class 10 while class 6 makes the correct classification. Classes 7 and 11 are correctly classified using the PgADJ rank aggregation method. Classes 8 and 12 are misclassified as class 2 or 3. Class 9 is misclassified as class 10 and class 10 is misclassified as either class 5 or 6. We notice two classes, 9 and 12, in which a new query is never classified. The Me and MeIBF rank aggregation methods have a high penalty value as we observed for PgADJ and Me when using the precision performance measure. We see through this experiment that the cost-based classification method does not always identify the correct class for the queries but the the substitution of another aggregation method due to misclassification does not degrade the performance dramatically.

	1	2	3	4	5	6	7	8	9	10	11	12
1	29980	841	860	0	413	418	76	0	0	7341	71	0
2	486	14116	14364	0	3977	4146	1314	0	0	346	1251	0
3	0	17521	17443	77	20	28	2456	78	0	0	2377	0
4	0	17493	17504	1623	0	0	934	1616	0	0	830	0
5	0	6	13	0	3801	3842	43	0	0	32252	43	0
6	0	621	628	0	18103	18438	1064	0	0	81	1065	0
7	0	4949	5010	8	197	197	14875	11	0	0	14753	0
8	0	17044	17234	1201	6	4	1641	1170	0	0	1700	0
9	0	3	2	0	3570	3469	5	0	0	32944	7	0
10	0	250	262	0	19211	19061	626	0	0	5	585	0
11	0	2411	2316	5	0	0	17720	4	0	0	17544	0
12	0	16525	16499	586	0	0	2930	612	0	0	2848	0

Real setting application. We have shown the possibility of using cluster quality and rank variance to find which class a new set of rankers belongs. We have also shown that if there is noise and misinformation, the dynamic methods may perform

better than a static aggregator. However, in our statistical framework, we have perfect knowledge about the misinformation and noise of past data sets. In a real life scenario, this is not possible. Another unknown in our approach is to identify the best aggregator for each class. Again, in real world, it is not possible to know the ground truth and hence we can not identify the best aggregator with respect to the ground truth. However, a more limited set of experiments can be run.

We can replicate our experiments in the real world. We must first define the query space. Given the queries posed to search engines and the associated search results, we can organize the queries with respect to misinformation and noise. These queries and their search results were our motivation for the statistical framework presented in Chapter 6. For each query and its rankings of search results, we can cluster the rankings to obtain cluster quality or misinformation and compute the variance of the ranks to define the noise. We can then find the similarity between queries based on the cluster quality and rank variance. The similarity amongst queries can be a function of the closeness between the cluster qualities and rank variance for every distinct pair of queries. The clustering of the query space will give us the different behavior of rankers with respect to different queries. We can use these regions to represent different noise and misinformation cases.

Once the different noise and misinformation classes have been identified, we then need to find the best aggregators for each case. We can compute the top aggregator by running the evaluation discussed in Chapter 4. We should note that a small number of pairs may be needed to learn the ordering of rankers in each class. As we are not trying to learn the exact ordering of all objects, but only the ordering of rankers, hence we do not need a large volume of training data.

Since there is no ground truth in web searching, we have the option of using relevance feedback [5, 17, 24], expert judgments [7, 13] or clicked search results [75] to represent a perceived ground truth. The ground truth equivalent and the top aggregators for misinformation and noise can be evaluated using the measures described in Chapter 3. When there is a new query, it can be classified using precision or Kendall-tau based on its ranker characteristics in which a dynamic aggregator can be used in lieu of a static aggregator.

CHAPTER 9
SUMMARY AND CONCLUSIONS

As a subset of the rank aggregation problem, we examine how the trustworthiness of the input rankers (misinformation) and the degree of noise in the input rankers (such as spam) contribute in influencing the choice in an aggregation method. In prior research, spam has been identified as a factor in varying performance of rank aggregation methods. However, this work does not provide a clear model of what constitutes spam. Our work provides the first principled study of noise in general and spam in particular in rank aggregation. We also introduce another factor, misinformation, to our study which models how well rankers act with respect to the ground truth. In an adversarial scenario, rankers cannot be expected to perform well uniformly for all queries. It is possible that certain rankers provide bad results due to different valuation systems or by purposefully mislead the rank aggregation systems.

In a real setting where the noise level and misinformation of rankers may vary greatly, the choice of an optimal aggregator depends greatly on how it handles noise and misinformation. We conclude that there is not a single aggregation method that performs well across each misinformation and noise case. As a result, we must develop methods to customize the rank aggregation methods to reflect the fluctuations in misinformation and noise. In this thesis, we have prove our first claim and then develop customization methods for dynamic rank aggregation.

We set out to investigate this problem by first designing a benchmarking system that we use to compare the aggregation methods. Our statistical framework models the relationship between rankers and the ground truth, the true ranking. It then evaluates the rank aggregation methods with respect to the ground truth. This method is a divergence from the previous work in this area which uses real data sets. By using a statistical framework, we are able to control the test cases and generate statistically significant number of tests for each case. We model noise, spam as a special type of noise and misinformation in this model. We can also model various

types of correlations that may exist between the rankers, objects and the errors that the rankers make for these objects.

We test well-known rank aggregation methods as well as a number of novel methods that we introduce using this statistical model. We also verify our results using real data sets. We introduce a rank aggregation algorithm called PrOpt (precision optimal) which is a simpler variation of the well-known CombMNZ method, and the IBF (iterative best flip) optimization method for optimizing the Kendall-tau error between input rankers. We also investigate other optimization methods for the Kendall-tau error for the equivalent graph problem of minimum feedback arc set (MFAS). We study three approximation algorithms based on a greedy approach (Greedy), sublist iterative best flip (SubIBF) and cut partitioning (CUT). The algorithms studied in this thesis present the most comprehensive study to date in this area. We study the effect of number of rankers, noise and misinformation and various correlations between the rankers.

Through the experimental evaluation, we show that the best aggregation method changes with respect to the misinformation and noise in the rankers. Rank aggregation methods that are more resistant to noise tend to disregard information available in the rankers. These rankers are not optimal in the case of low noise. Rank aggregation methods that tend to treat all rankers uniformly do well if there is no misinformation available in the rankers. However, if there is asymmetry between the performance of the rankers, then a ranker that is robust with respect to outliers is needed. However, as the noise and misinformation increases, there is less information available in each ranker. As a result, there is a need to incorporate more information from each ranker. In this case, the most robust rankers are not optimal. Table 9.1 provides a listing of the algorithms studied in this thesis and the specific observations made about these algorithms. We validate our findings with two real datasets. The first data set is a collection of rankings from real life search engines. We use this dataset to validate our results. We find the best rank aggregation methods with a user study and decide on noise and misinformation levels empirically. The ordering of rankers support our original findings. We also conduct a study using the TREC data set with known relevance labels. We verify our results

in this setting as well.

Algorithm	Observation
Av	Considers most information of all the rankers provided, performs best when there is no noise or very high misinformation
Me	Considers the median ranker, robust to outliers, performs best when there is little noise and some misinformation
Pg	Approximates Kendall-tau optimal ranking, robust to outliers, performs best in noisy but low misinformation cases
PrOpt	Disregards most rank information in noisy data, highly robust in high noise cases
CombMNZ	Similar to PrOpt, but considers rank information to a higher degree. PrOpt outperforms CombMNZ in very high noise cases
Cfuse	A greedy Kendall-tau optimizer that disregards most rank information, considers less information than Pg. Best in noisy cases with some misinformation
xADJ	A somewhat local optimizer for Kendall, it is good in cases that needs some robustness
xIBF	A global Kendall-tau optimizer, best in cases needing high robustness
Greedy	Very susceptible to wrong decisions and generally not competitive
xSubIBF	Similar to IBF, but more efficient
xCUT	Global optimizer similar to IBF, generally a better optimizer than IBF resulting in more robust rankings but more costly

Table 9.1: Synopsis of Algorithms

Research Extensions. Even though spam is one of the main problems facing search engines today, it is not the only one. Search engines are designed for the general public hence serves the most common interests of users. However, this may not always provide the most useful search results for a specific user or query. For example, when we are searching for a historic document on a topic, the freshness of information may not be very important for our search. Furthermore, search engines may have noisy estimates of this factor based on the frequency of visits to each site. Similarly, when searching for two keywords, we might be more interested in one

term than the other. When searching a topic that has more than one established viewpoint on the Web, we might be interested in one perspective over the others. Hence, we might want to tailor the search results to match our specific interests while retaining the obvious benefits of meta-search engines.

This thesis work can be extended in several ways. One extension would consider using scores instead of ranks. The features of image and video data consider colors, textures and shapes in which accuracy of this metadata is required to determine the best match for a request. If we were to use ranks, the granularity of the metadata would be compromised. Instead of considering only the first K, we may want to consider all the feature information that is above a certain threshold since scores are not always monotonic. Another extension of this research would be incorporating the customization of aggregation to search results. We would need to estimate, as accurately as we can, the search query space through the use of clustering, human evaluators or relevance feedback. Next, we can use a classification method to determine misinformation and noise and identify the best aggregators. Then we can adapt the search results to reflect the query's misinformation and noise by predicting the appropriate aggregation method. Lastly, a long-term extension involves changing the search results based on the previous clicked data. This different searching system would not simply include new results when querying the same information at different intervals. The searching system could use or ignore the previous selections, identified by the current user, to adapt the search results dynamically. The static search results retrieval would make way for more dynamic and interactive searching experience.

CITED LITERATURE

[1] M. Abidi and R. Gonzalez. *Data Fusion in Robotics and Machine Intelligence*. Academic Press, Inc., 1992.

[2] E. Agichtein, S. Lawrence, and L. Gravano. Learning search engine specific query transformations for question answering. In *World Wide Web*, pages 169–178, 2001.

[3] R. Agrawal and E. Wimmers. A framework for expressing and combining preference. In *Proceedings of ACM SIGMOD*, pages 297–306, 2000.

[4] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 684–693, 2005.

[5] J. Allan. Incremental relevance feedback for information filtering. In *Proceedings of ACM SIGIR*, pages 270–278, 1996.

[6] N. Alon. Ranking tournaments. *SIAM Journal of Discrete Mathematics*, 20(1):137–142, 2006.

[7] B. Amento, L. Terveen, and W. Hill. Does authority mean quality? predicting expert quality ratings of web documents. In *Proceedings of ACM SIGIR*, pages 296–303, 2000.

[8] P. Andritsos, J. R. J. Miller, and P. Tsaparas. Information-theoretic tools for mining database structure from large data sets. In *Proceedings of ACM SIGMOD*, pages 731–742, 2004.

[9] J. A. Aslam and M. Montague. Models of metasearch. In *Proceedings of ACM SIGIR*, pages 276–284, 2001.

[10] B. Babcock and C. Olston. Distributed top-k monitoring. In *Proceedings of ACM SIGMOD*, pages 28–39, 2003.

[11] J. Bar-Ilan. Search engine results over time–a case study on search engine stability. *Cybermetrics*, 2/3(1), 1998/9.

[12] J. Bar-Ilan. Methods for measuring search engine performance over time. *Journal of the American Society for Information Science and Technology*, 53(4):308–319, 2002.

[13] J. Bar-Ilan, M. Levene, and M. Mat-Hassan. Dynamics of search engine rankings - a case study. In *International Workshop on Web Dynamics*, 2004.

[14] J. J. Bartholdi, C. A. Tovey, and M. A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

[15] I. Bartolini, P. Ciaccia, V. Oria, and M. T. Ozsu. Integrating the results of multimedia sub-queries using qualitative preferences. In *International Workshop on Multimedia Information Systems*, pages 66–75, 2004.

[16] M. M. S. Beg and N. Ahmad. Soft computing techniques for rank aggregation on the world wide web. *World Wide Web: Internet and Web information Systems*, 6(1):5–22, 2003.

[17] S. M. Beitzel, E. C. Jensen, O. Frieder, D. D. Lewis, A. Chowdhury, and A. Kolcz. Improving automatic query classification via semi-supervised learning. In *Proceedings of the IEEE International Conference on Data Mining (ICDM 2005)*, pages 42–49, 2005.

[18] A. Benczur, K. Csalogany, T. Sarlos, and M. Uher. Spamrank - fully automatic link spam detection. In *International Workshop on Adversarial Information Retrieval on the Web*, 2005.

[19] M. K. Bergman. The deep web: Surfacing the hidden value. http://www.brightplanet.com/pdf/deepwebwhitepaper.pdf, 2001.

[20] E. Bertino, D. Montesi, and A. Trombetta. Fuzzy and presentation algebras for web and multimedia data. In *Proceedings of IEEE International Symposium on Database Engineering and Applications*, pages 134–144, 2000.

[21] K. Bharat and M. Hensinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of ACM SIGIR*, pages 104–111, 1998.

[22] K. Bharat and G. Mihaila. When experts agree: Using non-affliated experts to rank popular topics. *ACM Transactions on Information Systems*, 20(1):47–58, 2002.

[23] M. Bianchini, M. Gori, and F. Scarselli. Inside pagerank. *ACM Transactions on Internet Technology*, 5(1):92–128, 2005.

[24] A. Bifet, C. Castillo, P. A. Chirita, and I. Weber. An analysis of factors used in search engine ranking. In *Proceedings of the International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2005.

[25] J. C. Borda. Mémoire sur les élections au scrutin. In *Histoire de l'Académie Royale des Sciences*, 1781.

[26] A. Borodin, G. Roberts, J. Rosenthal, and P. Tsaparas. Link analysis ranking: Algorithms, theory, and experiments. *ACM Transactions on Internet Technology*, 5(1):231–297, 2005.

[27] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 421–430, 2001.

[28] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of ACM WWW*, pages 107–117, 1998.

[29] N. Bruno and S. Chaudhuri. Conditional selectivity for statistics on query expressions. In *Proceedings of ACM SIGMOD*, pages 211–322, 2004.

[30] N. Bruno, S. Chaudhuri, and L. Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Transactions on Database Systems*, 27(2):153–187, 2002.

[31] C. Buckley and E. M. Voorhees. Retrieval evaluation with incomplete information. In *Proceedings of ACM SIGIR*, pages 25–48, 2004.

[32] P. Calado, N. Z. Berthier Ribeiro-Neto, E. Moura, and I. Silva. Local versus global link information in the web. *ACM Transactions on Information Systems*, 21(1):42–63, 2003.

[33] K. Chakrabarti, S. Chaudhuri, and S. won Hwang. Automatic categorization of query results. In *Proceedings of ACM SIGMOD*, pages 755–766, 2004.

[34] S. Chakrabarti. *Mining the Web: Discovering Knowlegde from Hypertext Data*. Morgan Kaufmann Publishers, 2003.

[35] C.-Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified computation of skylines with partially-ordered domains. In *Proceedings of ACM SIGMOD*, pages 203–214, 2005.

[36] K. C.-C. Chang and S. won Hwang. Minimal probing: Supporting expensive predicates for top-k queries. In *Proceedings of ACM SIGMOD*, pages 346–357, 2002.

[37] Y.-C. Chang, L. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith. The onion technique: Indexing for linear optimization queries. In *Proceedings of ACM SIGMOD*, pages 391–402, 2000.

[38] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic ranking of database query results. In *Proceedings of ACM VLDB*, pages 888–899, 2004.

[39] S. Chaudhuri, L. Gravano, and A. Marian. Optimizing top-k selection queries over multimedia repositories. *IEEE Transactions on Knowledge and Data Engineering*, 16(8):992–1009, 2004.

[40] F. Y. L. Chin, X. Deng, Q. Fang, and S. Zhu. Approximate and dynamic rank aggregation. *Theoretical Computer Science*, 325(3):409–424, 2004.

[41] J. Cho and S. Roy. Impact of search engines on page popularity. In *Proceedings of ACM WWW*, pages 20–29, 2004.

[42] J. Cho, S. Roy, and R. E. Adams. Page quality: In search of an unbiased web ranking. In *Proceedings of ACM SIGMOD*, pages 551–562, 2005.

[43] J. Chomicki. Preference formulas in relational queries. *ACM Transactions on Database Systems*, 28(4):427–466, 2003.

[44] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. *Journal of Artificial Intelligence Research (JAIR)*, 10:243–270, 1999.

[45] D. Coppersmith, L. Fleischer, and A. Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 776–782, 2006.

[46] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms (Second Edition)*. McGraw-Hill, 2001.

[47] L. F. Cranor and B. A. LaMacchia. Spam! *Communications of the ACM*, 41(8):74–83, 1998.

[48] N. Craswell, F. Crimmins, D. Hawkings, and A. Moffat. Performance and cost tradeoffs in web search. In *Proceedings of the 15th Australasian Database Conference*, pages 161–169, 2004.

[49] B. Davison. Recognizing nepotistic links on the web. In *Workshop on Artificial Intelligence for Web Search*, pages 23–28, 2000.

[50] P. Diaconis. Group representation in probability and statistics. In *IMS Lecture Series 11*, 1988.

[51] P. Diaconis and R. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B*, 39(2):262–268, 1977.

[52] D. Donjerkovic and R. Ramakrishnan. Probabilistic optimization of top n queries. In *Proceedings of ACM VLDB*, pages 411–422, 1999.

[53] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of ACM WWW*, pages 613–622, 2001.

[54] N. Eiron, K. McCurley, and J. Tomlin. Ranking the web frontier. In *Proceedings of ACM WWW*, pages 309–318, 2004.

[55] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. *SIAM J. Discrete Mathematics*, 17(1):134–160, 2003.

[56] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proceedings of ACM SIGMOD*, pages 301–312, 2003.

[57] R. Fagin and E. L. Wimmers. A formula for incorporating weights into scoring rules. *Theoretical Computer Science*, 239(2):309–338, 2000.

[58] P. Ferragina and A. Gulli. A personalized search engine based on web-snippet hierarchical clustering. In *Proceedings of ACM WWW*, pages 801–810, 2005.

[59] D. Fetterly, M. Manasse, and N. Najork. Spam, damn spam, and statistics: using statistical analysis to locate spam web pages. In *Proceedings of the International Workshop on the Web and Databases*, pages 1–6, 2004.

[60] J. Fox and E. Shaw. Combination of multiple sources: The trec-2 interactive track matrix experiment. In *Proceedings of ACM SIGIR*, 1994.

[61] Y. Freund, R. Iyer, R. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4(6):933–969, 2004.

[62] L. Gravano and H. Garcia-Molina. Merging ranks from heterogeneous internet sources. In *Proceedings of ACM VLDB*, pages 196–205, 1997.

[63] U. Guntzer, W.-T. Blake, and W. Kiessling. Optimizing multi-feature queries for image databases. In *Proceedings of ACM VLDB*, pages 419–428, 2000.

[64] U. Guntzer, W.-T. Blake, and W. Kiessling. Toward efficient multi-feature queries in hetergeneous environments. In *IEEE International Conference on Information Technology:Coding and Computing*, 2001.

[65] T. Haveliwala. Topic-sensitive pagerank. In *Proceedings of ACM WWW*, pages 517–526, 2002.

[66] S. Holland, M. Ester, and W. Kiessling. Preference mining: A novel approach on mining user preferences for personalized applications. In *7th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 204–216, 2003.

[67] V. Hristidis, N. Koudas, and Y. Papakonstantinou. Prefer: A system for the efficient execution of multi-parametric ranked queries. In *Proceedings of ACM SIGMOD*, pages 259–270, 2001.

[68] V. Hristidis and Y. Papakonstantinou. Algorithms and application for answering ranked queries using ranked views. *VLDB Journal*, 13:49–70, 2004.

[69] http://trec.nist.gov/. Text retrieval conference (trec), 1999.

[70] I. Ilyas, W. Aref, and A. Elmagarmid. Joining ranked inputs in practice. In *Proceedings of ACM VLDB*, pages 950–961, 2002.

[71] I. Ilyas, R. Shah, W. Aref, J. S. Vitter, and A. Elmagarmid. Rank-aware query optimization. In *Proceedings of ACM SIGMOD*, pages 203–214, 2004.

[72] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k *join* queries in relational databases. *VLDB Journal*, 13:207–221, 2004.

[73] B. Jansen, A. Spink, J. Bateman, and T. Saracevic. Real life information retrieval: A study of user queries on the web. *ACM SIGIR Forum*, 32(1):5–17, 1998.

[74] K. Jarvelin and J. Kekalainen. Ir evaluation methods for retrieving highly relevant documents. In *Proceedings of ACM SIGIR*, pages 41–48, 2000.

[75] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of ACM SIGKDD*, pages 133–142, 2002.

[76] J. Kemeny and J. Snell. Mathematical models in the social sciences, 1962.

[77] J. G. Kemeny. Mathematics without numbers. *Daedalus*, 88:571–591, 1959.

[78] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49(1):291–307, 1970.

[79] R. Khoussainov and N. Kushmerick. Specialisation dynamics in federated web search. In *Proceedings of the ACM CIKM International Workshop on Web Information and Data Management (WIDM)*, pages 112–119, 2004.

[80] W. Kiessling. Foundations of preferences in database systems. In *Proceedings of ACM VLDB*, pages 311–322, 2002.

[81] W. Kiessling. Preference queries with sv-semantics. In *International Conference of Management of Data*, pages 25–26, 2005.

[82] L. A. Klein. *Sensor and Data Fusion: A tool for Information Assessment and Decision Making*. The International Society for Optical Engineering, 1999.

[83] R. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[84] G. Koutrika and Y. Ioannidis. Personalized queries under a generalized preference model. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, pages 841–852, 2005.

[85] G. Lebanon and J. Lafferty. Cranking: Combining rankings using conditional probability models on permutations. In *Proceedings of the International Conference on Machine Learning*, pages 363–370, 2002.

[86] J. H. Lee. Analyses of multiple evidence combination. In *Proceedings of ACM SIGIR*, pages 267–276, 1997.

[87] R. Lempel and S. Moran. Salsa: The stochastic approach for link-structure analysis. *ACM Transactions on Information Systems*, 19(2):131–160, 2001.

[88] R. Lempel and S. Moran. Predictive caching and prefetching of query results in search engines. In *Proceedings of ACM WWW*, pages 19–28, 2003.

[89] R. Lempel and S. Moran. Rank-stability and rank-similarity of link-based web ranking algorithms in authority-connected graphs. *Information Retrieval*, 8(2):245–264, 2005.

[90] A. Lotem, R. Fagin, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66:614–656, 2003.

[91] Y. Lu, W. Meng, L. Shu, C. Yu, and K.-L. Liu. Evaluation of result merging strategies for metasearch engines. In *Proceedings of the 6th International Conference on Web Information Systems Engineering (WISE)*, 2005.

[92] C. Manning and H. Schútze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[93] A. Marian, N. Bruno, and L. Gravano. Evaluating top-k queries over web-accessible databases. *ACM Transactions on Database Systems*, 29(2):319–362, 2004.

[94] I. McLean and A. B. Urken. *Classics of Social Choice*. The University of Michigan Press, 1995.

[95] W. Meng, Z. Wu, C. Yu, and Z. Li. A highly scalable and effective method of methasearch. *ACM Transactions on Information Systems*, 19(3):310–335, 2001.

[96] W. Meng, C. Yu, and K.-L. Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89, 2002.

[97] M. Montague and J. A. Aslam. Condorcet fusion for improved retrieval. In *Proceedings of ACM CIKM*, pages 538–548, 2002.

[98] A. Motro, P. Anokhin, and A. C. Acar. Utility-based resolution of data inconsistencies. In *Proceedings of the ACM International Workshop on Information Quality in Information Systems (IQIS)*, pages 35–43, 2004.

[99] A. Natsev, Y.-C. Chang, J. R. Smith, C.-S. Li, and J. S. Vitter. Supporting incremental join queries on ranked inputs. In *Proceedings of ACM VLDB*, pages 281–290, 2001.

[100] S. Pandey, S. Roy, C. Olston, J. Cho, and S. Chakrabarti. Shuffling a stacked deck: the case for partially randomized ranking of search engine results. In *Proceedings of ACM VLDB*, pages 781–792, 2005.

[101] R. K. Pon and A. F. Cárdenas. Data quality inference. In *Proceedings of the 2nd International Workshop on Information Quality in Information Systems (IQIS)*, pages 105–111, 2005.

[102] A. L. Powell and J. C. French. Comparing the performance of collection selection algorithms. *ACM Transactions on Information Systems*, 21(4):412–456, 2003.

[103] Y. Rasolofo, D. Hawking, and J. Savoy. Results merging strategies for a current news metasercher. *Information Processing and Management*, 39(4):581–609, 2003.

[104] M. E. Renda and U. Straccia. Web metasearch: Rank vs. score based rank aggregation methods. In *Proceedings of ACM SAC*, pages 841–846, 2003.

[105] T. Ross. *Fuzzy Logic with Engineering Applications*. McGraw-Hill,Inc., 1995.

[106] Y. Saab. A fast and effective algorithm for the feedback arc set problem. *Journal of Heuristics*, 7(3):235–250, 2001.

[107] D. Saari. *Basic Geometry of Voting*. Springer-Verlag, 1995.

[108] G. Salton. *Automatic Information Organization and Retrieval*. McGraw-Hill,Inc., 1968.

[109] P. C. Saraiva, E. S. de Moura, N. Ziviani, W. Meira, R. Fonseca, and B. Ribeiro-Neto. Rank-preserving two-level caching for scalable search engines. In *Proceedings of ACM SIGIR*, pages 51–58, 2001.

[110] SearchEngineWatch. http://searchenginwatch.com.

[111] L. Si and J. Callan. Using sampled data and regression to merge search engine results. In *Proceedings of ACM SIGIR*, pages 19–26, 2002.

[112] L. Si and J. Callan. A semisupervised learning methods to merge search engine results. *ACM Transactions on Information Systems*, 21(4):457–491, 2003.

[113] J. Teevan, S. T. Dumais, and E. Horvitz. Personalizing search via automated analysis of interests and activities. In *Proceedings of ACM SIGIR*, pages 449–456, 2005.

[114] Teoma. http://www.teoma.com, 2000.

[115] J. Tomlin. A new paradigm for ranking pages on the world wide web. In *Proceedings of ACM WWW*, pages 350–355, 2003.

[116] P. Tsaparas, T. Palpanas, Y. Kotidis, N. Koudas, and D. Srivastava. Ranked join indices. In *Proceedings of IEEE ICDE*, 2003.

[117] T. Tsikrika and M. Lalmas. Merging techniques for performing data fusion on the web. In *Proceedings of ACM CIKM*, pages 127–134, 2001.

[118] L. Vaughan. New measurements for search engine evaluation proposed and tested. *Information Processing and Management*, 40(4):677–691, 2004.

[119] Vivissimo. http://www.vivissimo.com, 2000.

[120] M. Wechsler and P. Schauble. A new ranking principle for multimedia information retrieval. In *Proceedings of the ACM conference on Digital Libraries*, pages 146–151, 1999.

[121] Z. Wu, W. Meng, C. Yu, and Z. Li. Towards a highly-scalable and effective metasearch engine. In *Proceedings of ACM WWW*, pages 386–395, 2001.

[122] C. Yu and W. Meng. Web search technology. In *The Internet Encyclopedia*, pages 738–753, 2003.