3-22-2002

Metasearch: Data Fusion for Document Retrieval

Mark H. Montague
Dartmouth College

Metasearch: Data Fusion for Document Retrieval

(Dartmouth Computer Science Technical Report TR2002-424)

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

in

Computer Science

by

Mark Montague

DARTMOUTH COLLEGE

Hanover, New Hampshire

March 22nd, 2002

Examining Committee:

(chair) Javed A. Aslam

James Allan

Robert Gray

David Kotz

Carol Folt
Dean of Graduate Students

Abstract

The *metasearch* problem is to optimally merge the ranked lists output by an arbitrary number of search systems into one ranked list. In this work:

(1) We show that metasearch improves upon not just the raw performance of the input search engines, but also upon the consistency of the input search engines from query to query.

(2) We experimentally prove that simply weighting input systems by their average performance can dramatically improve fusion results.

(3) We show that score normalization is an important component of a metasearch engine, and that dependence upon statistical outliers appears to be the problem with the standard technique.

(4) We propose a Bayesian model for metasearch that outperforms the best input system on average and has performance competetive with standard techniques.

(5) We introduce the use of Social Choice Theory to the metasearch problem, modeling metasearch as a democratic election. We adapt a positional voting algorithm, the Borda Count, to create a metasearch algorithm, acheiving reasonable performance.

(6) We propose a metasearch model adapted from a majoritarian voting procedure, the Condorcet algorithm. The resulting algorithm is the best performing algorithm in a number of situations.

(7) We propose three upper bounds for the problem, each bounding a different class of algorithms.

We present experimental results for each algorithm using two types of experiments on each of four data sets.

Preface

In this work, as is standard in scientific writing, I use the first person plural pronoun "we" in place of the singular "I," thereby implicitly invoking the host of people who stand behind me, who make it possible for me to say what I do and even to speak at all. That "we" includes too many people to list, but it includes first and foremost my wife, Laura, who has patiently aided me in every stage of this work. It also includes my advisor, Javed Aslam, whose close collaboration has been indispensible. Without these two there would be no thesis.

Mark Montague
Claremont, NH
March, 2002

Contents

I Introduction **1**

1 Introduction **2**
1.1 Background . . . 2
1.1.1 Information Retrieval . . . 2
1.1.2 Search Engines . . . 3
1.1.3 Data Fusion . . . 3
1.2 Metasearch . . . 5
1.2.1 Characterizing Metasearch . . . 5
1.2.2 Benefits of Metasearch . . . 8
1.3 Our Work . . . 10
1.3.1 Improvements . . . 11
1.3.2 New Algorithms . . . 11
1.3.3 Evaluation . . . 12

2 Related Work **13**
2.1 Preliminaries . . . 13
2.1.1 Definitions . . . 13
2.1.2 Evaluation . . . 14
2.1.3 Score Normalization . . . 15
2.1.4 TREC . . . 15
2.2 Related Problems . . . 16
2.2.1 Fusion Effectiveness . . . 16
2.2.2 Collection Fusion . . . 17
2.3 Fusion Techniques . . . 18
2.3.1 Early Studies . . . 18
2.3.2 Min, Max, and Avg Models . . . 19
2.3.3 Logistic Regression Model . . . 21
2.3.4 Linear Combination Model . . . 21
2.3.5 Probabilistic Argumentation . . . 23
2.3.6 Using Document Summaries . . . 23

II Weights and Scores **24**

3 Evaluation **25**
3.1 Experiments . . . 25
3.1.1 Data Sets . . . 25
3.1.2 Procedure . . . 27
3.1.3 Comb Algorithms . . . 28
3.2 Four Variants . . . 31
3.2.1 Ranks only: Score Simulation . . . 31
3.2.2 Training Data: Performance Weights . . . 31
3.2.3 Experimental Results . . . 33
3.3 Consistency . . . 33
3.3.1 Experimental Setup . . . 35
3.3.2 Experimental Results . . . 35
3.4 Conclusion . . . 37

4 Relevance Score Normalization **38**
4.1 Problem . . . 38
4.1.1 Decomposition . . . 38
4.1.2 Related Work . . . 39
4.2 Score Manipulations . . . 39
4.2.1 Normalization . . . 39
4.2.2 Estimation and Combination . . . 41
4.3 Experimental Results . . . 41
4.4 Conclusion . . . 45

5 Bayes-fuse **46**
5.1 The Probabilistic Model . . . 46
5.1.1 Derivation . . . 46
5.1.2 Implications . . . 47
5.1.3 Another Perspective . . . 48
5.1.4 Implementation . . . 49
5.1.5 Experimental Results . . . 52
5.2 Conclusion . . . 52

III Voting **54**

6 Borda-fuse **55**
6.1 Social Choice . . . 55
6.2 The Borda-fuse Model . . . 57
6.2.1 Definition . . . 57
6.2.2 Related Algorithms . . . 57
6.2.3 Relevance Scores . . . 57
6.2.4 Weighted Borda-fuse . . . 58
6.3 Experimental Results . . . 58

6.4 Conclusion . . . 58

7 Condorcet-fuse **60**
7.1 The Voting Model . . . 60
7.1.1 The Condorcet Graph . . . 60
7.1.2 The Voting Paradox . . . 61
7.1.3 Strongly Connected Components . . . 62
7.1.4 Condorcet Paths . . . 62
7.1.5 Efficiency . . . 64
7.1.6 Variants . . . 64
7.2 Experimental Results . . . 65
7.3 Conclusion . . . 68

8 Upper Bounds **70**
8.1 Constrained Oracle Model . . . 70
8.1.1 Naive Bound . . . 70
8.1.2 Pareto Bound . . . 71
8.1.3 Majoritarian Bound . . . 71
8.1.4 Constraint Graph . . . 71
8.1.5 Comparing the Bounds . . . 72
8.2 Implementation . . . 72
8.3 Experimental Results . . . 73
8.4 Conclusion . . . 73

9 Conclusion **75**
9.1 Metasearch Comparison . . . 75
9.1.1 Average Improvement . . . 75
9.1.2 No Scores, No Training Data . . . 76
9.1.3 No Scores, Training Data . . . 76
9.1.4 Scores, No Training Data . . . 76
9.1.5 Scores, Training Data . . . 76
9.2 Contributions . . . 79

List of Figures

1.1 The metasearch scenarios we study. Metasearch techniques may or may not require relevance scores, and may or may not require training. 6

1.2 Internal versus external metasearch. External metasearch is modeled on the left: a metasearch engine combines three complete and independent search engines. Internal metasearch is on the right: a metasearch algorithm combines three cooperating sub-engines over the same set of documents. 7

3.1 The improvement of three Comb systems. The random-sets experiment is shown on the left, best-to-worst on the right. Each row is for a different data set. The x-axis is the number of systems being combined, and the y-axis is improvement. 29

3.2 The improvement of the worst three Comb systems. The ranges of the y-axes here are unusual due to the poor performance of these systems. 30

3.3 The four CombMNZ variants. Note the naming convention: "s" stands for *score,* "r" for *rank,* "w" for *weighted,* and "u" for *unweighted.* 33

3.4 The four CombMNZ variants. swCombMNZ performs especially well; in the random-sets experiment it outperforms the best input system, even over TREC 9. 34

3.5 Each row of graphs is over one data set: TREC 3, TREC 5, Vogt, and TREC 9. Each column shows one statistic: raw performance, standard deviation, and percent standard deviation (i.e., coefficient of variation). The horizontal axis of each graph is the number of systems being fused; the vertical axis is the statistic of interest. The better metasearch performs (column 1), the more it improves upon the consistency of the best input system (column 3). 36

4.1 The improvement of CombSum using the three different score normalization schemes. The ZMUV normalization almost always works best. 42

4.2 The improvement of CombMNZ with different normalizations. The ZMUV norm interacts poorly with CombMNZ, but the sum norm usually improves CombMNZ. 43

4.3 The 2MUV variant of the ZMUV norm with CombMNZ. 2MUV solves the problem of ZMUV with CombMNZ, but the sum norm tends to work at least as well. 44

5.1 `trec_eval` statistics for TREC 3's `inq102`. 50

5.2 The improvement over the best input system given by rwBayes-fuse and swBayes-fuse. Rank-based Bayes-fuse outperforms score-based. 51

6.1 The improvement of all four Borda-fuse variants. Note that score-based Borda-fuse variants are labeled as CombSum algorithms using the sum normalization. 59

7.1 A five candidate Condorcet graph. 60
7.2 The "voting paradox:" a simple case of cyclic preferences. 61
7.3 The three cases for the proof of Theorem 7.1.1. 63
7.4 The improvement of Condorcet-fuse. Weighted and unweighted versions are shown. 66
7.5 Dependence filtering. druCondorcet-fuse is not susceptible to the dependence problems that ruCondorcet-fuse is. 69

8.1 Upper bounds. The only actual metasearch algorithm shown is ruCondorcet-fuse. 74

List of Tables

3.1 The data sets used in our experiments. 25
3.2 Summary of Fox and Shaw's Comb fusion algorithms. 28

4.1 Normalization algorithms. We propose the sum and ZMUV norms as simple and effective replacements for the standard norm. 40

7.1 The top five input systems from the TREC 9 data set. Systems 2, 3, and 4 are runs from the same research team. 65
7.2 The average pairwise set-similarity of runs in each data set. 67

9.1 Summary of metasearch variants that apply when neither scores nor training data are available. 76
9.2 Summary of metasearch variants that apply when scores are not available but training data is. 77
9.3 Summary of metasearch variants that apply when scores are available but training data is not. 77
9.4 Summary of the performance of metasearch techniques that apply when scores and training data are available. The upper bounds are also shown. . 78

Part I

Introduction

Chapter 1

Introduction

This work concerns improving search engines by combining them: metasearch. Once we establish the proper background, we describe the problem and summarize our efforts to solve it.

1.1 Background

Metasearch is an application of data fusion to search engines, and is a topic within the field of Information Retrieval.

1.1.1 Information Retrieval

Information Retrieval (IR) is the subfield of Computer Science concerned with presenting information gathered from online information sources to online information users. Online information sources range from relatively stable document sets like the set of all World Wide Web pages to document streams like email buffers and news-wires. Typical IR tasks can be roughly grouped based on their input:

- Stable document set, incoming queries:
  - *document retrieval*: return documents relevant to the given query.
  - *cross language retrieval*: document retrieval where the queries are given in a different language than the document set.
  - *question answering*: in response to a specific question, extract a set of specific answers from a document set.
- Incoming documents, stable query set:
  - *routing* or *filtering*: label each incoming document as relevant or irrelevant to a predefined category.
  - *categorizing*: sort a given set of documents into predefined categories.

- Stable document set, no queries:
  - *clustering*: organize the set of documents into natural groups.
- Incoming documents, no queries:
  - *topic detection*: identify events reported in a document stream.

In this work, we focus on what is perhaps the most widely used Information Retrieval tool: document retrieval systems or *search engines*.

1.1.2 Search Engines

The World Wide Web contains an enormous amount of useful information and it combines some of the best features of other communication media: both up-to-date and archival information available in text, images, audio, and video. The Web, however, is a *pull* medium: users must request the data they desire, as opposed to *push* media, where the content is broadcast. And it is vast—search engines have indexed over a billion pages. Hence the critical role played by search engines: search engines help the user find pages of interest to pull.

Search engines provide the user with a simple interface: the user enters a short query (a statement of information need), to which the retrieval system responds, usually with a ranked list of documents. The AltaVista[1] and Google[2] Web search engines are good examples. In some cases a *relevance score* is given along with each document, indicating on a more or less arbitrary scale the degree of relevance of each document.

As a concrete example, consider the composition of a Web search engine: First there is a *database*, which contains an internally meaningful representation of each document of which the search engine is aware. Next, a *Web crawler*, whose job it is to update the database. It continuously downloads Web pages from the Internet, *indexes*, or processes them into the internal representation, and stores them in the database. Then there is a *ranking algorithm*, which receives user queries, compares them to documents from the database, and ranks the documents in order of decreasing relevance to the query. Finally, a *user interface*, e.g., a Web site, allows users to enter queries and view the ranked results.

In this work, we focus exclusively on the ranking algorithm. In fact, we are not concerned with the details of any one ranking algorithm, but rather how to build an improved ranking algorithm by combining the results of multiple ranking algorithms. We improve retrieval by applying the tools of *data fusion*.

1.1.3 Data Fusion

Data fusion is a problem-solving technique based on the idea of integrating many (perhaps partial or idiosyncratic) answers to a question into a single, best answer. The data fusion approach:

[1] `http://www.altavista.com`
[2] `http://www.google.com`

- is well suited to problems involving massive amounts of data, where each subsystem may not have the entire data set,
- is well suited to problems with many possible approaches, none of which dominates the others,
- allows for natural and flexible distribution of resources (e.g., database space or processing time),
- aims to provide better performance than the best input system, and
- is practical when many input systems already exist.

These qualities make this approach ideal for Web document searching:

- The data set is vast: large search engines have indexed over 1 billion Web pages.
- Many document databases and search techniques exist today.
- Although search engine performance has improved dramatically over the past few years, there is still room for improvement. Combining systems may give improvements over the best existing system.

Voting procedures are examples of data-fusion techniques. In this case, the problem being solved is that of choosing a particular item out of a set, e.g., citizens electing governmental officials, or stockholders deciding between business strategies. The data being fused is the voters' personal opinions about the items in the set. The voter appraises each candidate and submits this information as required by the voting procedure (i.e., the fusion algorithm). The simple and common plurality voting procedure asks each voter for only their top candidate, and the candidate mentioned the most often is output as the fused result. Many other voting strategies exist, however. Some require a ranked list of candidates from each voter and implement more complicated algorithms to determine the fused result. We return to this example in much more detail in subsequent chapters where we adapt two different voting procedures to the metasearch problem.

An important distinction must be made between so-called *collection* fusion, where results from disjoint data sets are merged, and *data* fusion, where results from identical data sets are merged [54, 75, 78]. The two problems are related but distinct: in collection fusion the goal is to "recover" the performance of a single monolithic search system when there has been reason to divide up the database. Optimal performance is simply the performance of the system when the database is united. In data fusion on the other hand, the goal is to *exceed* the performance of any one of the systems being combined. Here optimal performance is not known—in Chapter 8 we explore upper bounds on optimal data-fusion performance for our problem.

In this work, we restrict ourselves to the study of data fusion, where the data-set overlap is 100%. Because of the close relationship between the problems, we expect that our techniques will be helpful in future studies of collection fusion. They cannot, however, be applied blindly. Moreover, we propose that data fusion algorithms do have a useful direct application—not in combining different, unrelated search engines, but (and this may

prove to be the more important problem) in combining different subsystems within the same search engine. In the next section we propose that metasearch is an excellent way to modularly restructure a single search engine, as opposed to combining multiple pre-existing engines as a post-processing technique.

Note that when applying fusion techniques to combine different Web search engines, one generally cannot assume that their data sets are either disjoint or identical; they can overlap arbitrarily.

1.2 Metasearch

Metasearch is the application of fusion to document retrieval. In its simplest form, a metasearch engine takes as input the n ranked lists output by each of n search engines in response to a given query. As output, it computes a single ranked list, one that is hopefully an improvement over any of the input lists as measured by standard IR performance metrics.

1.2.1 Characterizing Metasearch

We may further clarify the metasearch problem by characterizing the input data it requires and the ways in which it can be used.

Characterizing the Input

Ranks versus scores. Recall that search engines generally return a list of references to documents—commonly Internet URLs (Uniform Resource Locator) or some other unique identifier. However, the specifics of what exactly they return can vary.

1. Boolean search engines return an unordered *set* of results. Though it is easy to imagine ways to combine sets of results, we are not aware of any metasearch engines designed to handle sets alone.

2. Many engines return a *ranked list* of results. We study rank-based metasearch engines in detail. Ng and Kantor [55] call this *decision-level* fusion.

3. Some return a *relevance score* along with each document. We also study score-based metasearch in detail. Ng and Kantor call this *signal-level* fusion.

4. Some return a *title and short text summary* of each document. The Amvish system of Shu and Kak [69] bases its metasearch combination on these textual summaries.

5. Some metasearch systems take the time to look up each document referenced by the input system, and use the *entire document* as input to the metasearch algorithm. Ng and Kantor call this *data-level* fusion. Vogt's ACE model [74] explores this approach in an experimental framework, and the NECI metasearch engine [39] implements it on the Web. The INQUERY retrieval system [72] can be considered another example.

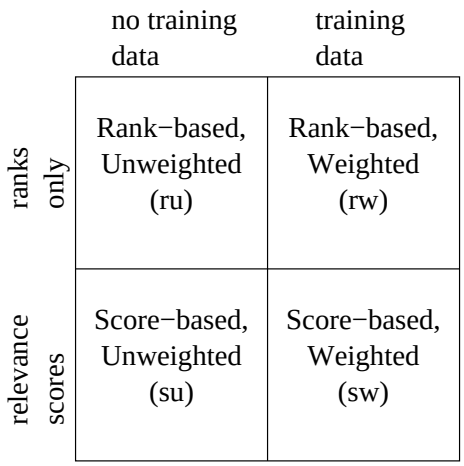

Figure 1.1: The metasearch scenarios we study. Metasearch techniques may or may not require relevance scores, and may or may not require training.

In this work, we restrict ourselves to the two most common cases, when either only ranks or ranks plus relevance scores are available. See Figure 1.1.

Training data. Another dimension in Figure 1.1 is the presence or absence of training data. Training data usually takes the form of *relevance judgments:* a human-made assessment of the relevance of a particular document to a particular query. From a sufficient quantity of this information can be deduced the overall performance of a search engine, or other statistics of interest to the metasearch algorithm. Since these relevance judgments are generally fairly expensive to make (each one involves human analysis), training data may or may not be readily available to a metasearch engine. In this work we study both cases.

Data versus collection fusion. A third dimension could be added to Figure 1.1, representing the amount of overlap in the input systems' databases: no overlap (collection fusion), arbitrary overlap, or 100% overlap (data fusion). In this work we study only the data-fusion case.

Characterizing the Application

Applications of metasearch are of two types: *external* or *internal* (see Figure 1.2). External metasearch takes existing, "complete" search systems and tries to improve upon them by combining them. Numerous search systems have been developed both in academia and in industry ([16, 79], Google, Alta Vista, Lycos,[3] etc.). In practice, no one system performs "better" than each of the others under all circumstances, and the "best" system for a particular task may not be known *a priori.* This being the case, external metasearch

[3] `http://www.lycos.com/`

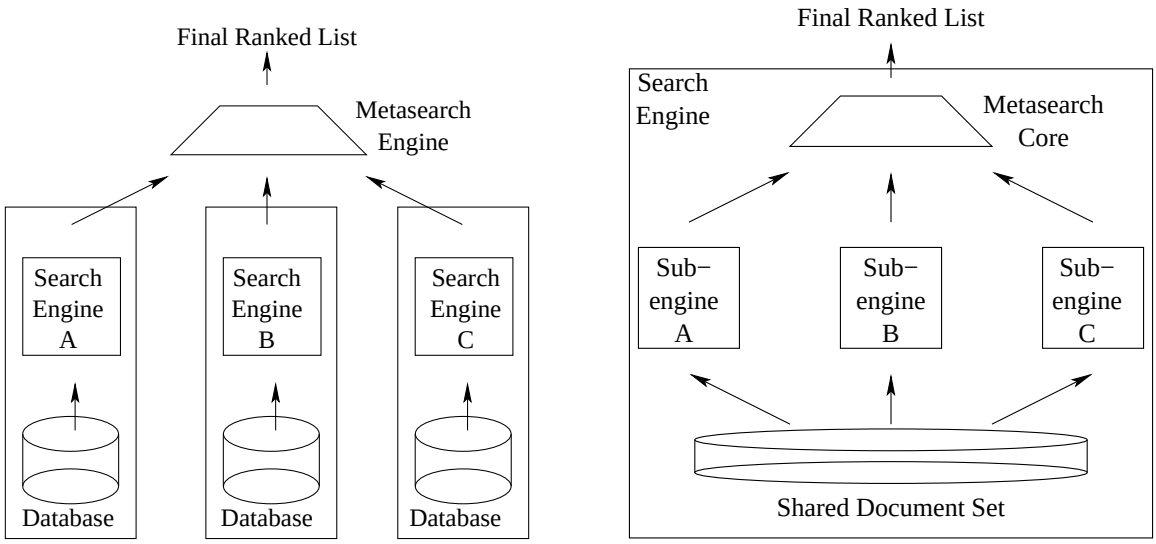

Figure 1.2: Internal versus external metasearch. External metasearch is modeled on the left: a metasearch engine combines three complete and independent search engines. Internal metasearch is on the right: a metasearch algorithm combines three cooperating sub-engines over the same set of documents.

engines (such as MetaCrawler,[4] ProFusion,[5] Debriefing,[6] Search.com,[7] Highway 61,[8] etc.) have been introduced that query several search engines, merge the lists of pages returned, and present a resulting ranked list to the user. Web metasearch engines are external: they combine the output of "complete" search engines, as a kind of post-processing, value-adding stage.

Internal metasearch, on the other hand, is metasearch designed into the heart of a retrieval system. Metasearch offers a systematic way of incorporating all of the various types of evidence available to a search engine. For example, for the problem of Web-page retrieval, there are many sources of information: each page has text, in-links, out-links, images, tags, and keywords. For each of these elements, numerous indexing and searching algorithms may exist. If we can design powerful fusion techniques, all of the information contained in these disparate components can be combined effectively.

We now outline the likely properties of external versus internal metasearch. Note, however, that each application of metasearch will be unique; the properties we list here are mainly inspired by considering the case of today's Web search engines.

External metasearch on the Web is characterized by:

[4] `http://www.metacrawler.com/`
[5] `http://www.profusion.com/`
[6] `http://www.debriefing.com/`
[7] `http://www.search.com/`
[8] `http://www.highway61.com/`

- *Overlapping databases.* Perhaps modeled better by collection-fusion assumptions than by data-fusion assumptions if data-set overlap is low.
- *Ranks only.* In some cases relevance scores are reported, but often they are not.
- Training data may or may not be readily available.
- *Dependent input systems.* It is likely that the input systems construct their rankings based on largely the same information (though they may well use different algorithms), since each one is intended to be a complete system.
- *Short ranked lists* from each input system. The end user often is only interested in the top 10-50 results, so most systems only report that many.

Internal metasearch is characterized by:

- *Identical databases.* Each sub-engine focuses on a different information source within the documents, but each is over the same document set.
- *Relevance scores* are generally available from each sub-engine, since they are cooperating.
- Training data may or may not be readily available.
- *Independent input systems.* Since the sub-engines are cooperating, they naturally do not duplicate each other's work. Rather, they use different sources of information and/or different techniques.
- *Long ranked lists* from each input system. It is likely possible to request the top 1000 documents from each sub-engine.

Thus it appears that metasearch is a more appropriate tool for internal use than for external. And we can expect far better results from internal metasearch. Therefore we have geared our experiments to simulate an internal metasearch environment.

1.2.2 Benefits of Metasearch

We already know that metasearch attempts to combine the strength of multiple search engines, but it is worth pausing to consider in more detail how exactly we expect metasearch to improve the performance of search engines, and in each case how we test how well it has worked.

A larger database: As more retrieval systems with overlapping databases are added to the fusion, new documents are made available to the user. Depending on the fusion scheme, a document appearing in only one database may not be as likely to be retrieved by the metasearch engine as a document appearing in all of the databases. Although it may be hard to measure, it is useful to conceive of the "effective database" size of the metasearch engine—it can be as small as the intersection of the constituent databases for a poor fusion algorithm, or as large as the union for a good one.

Interestingly, the difference between the database union and intersection can be large. In 1997, Bharat and Broder [8] estimated the size of the indexable portion of the Web at 200 million pages. Of these, an estimated 160 million pages were in the union of the databases of HotBot, AltaVista, Excite, and Infoseek. However, the estimated intersection of their databases was less than 1.4% of their union—only about 2.2 million pages.

Using metasearch to obtain a larger database is especially important on the Web, where it has been shown that the major search engines (at least in 1998) each cover only a relatively small fraction of the entire indexable Web [40]. And, in [41], Lawrence and Giles find that the fraction of the Web being covered by search engine databases is actually shrinking.

Note that in this work we restrict our experiments to the data-fusion scenario where each input system is over exactly the same document set. Thus the performance improvements of metasearch that we report later are definitely not due to any increase in database size.

In any case, a larger database will only help the user if it causes the system's performance to improve: if it ranks more relevant documents more highly. Search engine performance is usually measured with *precision* and *recall*.

Improved recall: *Recall* is the ratio of retrieved relevant documents to total relevant documents. In the case of data fusion, when the constituent systems have identical databases, the traditional argument is that metasearch will provide improved recall since different systems will return different documents. Indeed different systems do seem to retrieve different documents: in one data set,[9] each of 61 search engines retrieved 1000 documents for each of 50 queries. The average intersection between pairs of systems on each query is only 238 documents ($\sigma = 206$) [74], so on average the different systems are returning many different documents. But as Ng and Kantor point out [54], to achieve higher recall via fusion it is necessary that the input systems retrieve not just different documents, but different *relevant* documents. In fact, Lee [43] convincingly argues that different retrieval methods (over the same database) often return many different irrelevant documents, but many of the same relevant documents. Thus metasearch may improve recall only marginally.

We do not test for improved recall independent of its cousin, precision.

Improved Precision: *Precision* is the ratio of retrieved relevant documents to retrieved documents. In 1988, Saracevic and Kantor [64] showed that the odds of a document being relevant increase monotonically with the number of search engines that retrieve it. Lee [43] argues that an "unequal overlap property" holds in ranked-list fusion: different retrieval algorithms retrieve many of the same relevant documents, but different irrelevant documents. Ng and Kantor [54] conclude with Lee that if this is true, any fusion technique that more heavily weights common documents should improve precision. Vogt calls this the "chorus effect" [74], and we expect that the greatest strength of metasearch lies in improving precision in this way.

As usual, precision and recall are pitted against one another: weighting common documents heavily may improve precision, but will likely harm recall, as rare relevant documents are de-emphasized. Thus in this work we do not test the ability of metasearch to improve only recall or only precision; rather we use a standard combined measure, *average precision at relevant documents*, which we define later.

[9] The ranked lists submitted to the TREC 5 adhoc track, which we describe in detail later.

More consistent performance: Another desirable quality for a search engine is reliable behavior. Selberg and Etzioni [66] show that the same Web search engine often responds to the same query very differently over time, perhaps due to the evolution of its database. Even with a fixed database, however, each search engine will have its strengths and weaknesses, performing well on some queries and poorly on others. Inasmuch as metasearch is an averaging procedure, we can expect the idiosyncrasies of any one system to be smoothed out in the fusion, yielding a more reliable search system.

Later in this work we explicitly examine the gains in consistency afforded by metasearch by measuring the variation of performance over multiple queries.

Modular Architecture: When designing a search engine, one is faced with many different sources of information about each document: word frequencies, phrase frequencies, textual structure within a document, hyperlink structure between documents, anchor text for each hyperlink, images, etc. How can all of this information be incorporated sensibly, in a way that will take advantage of each? We propose that metasearch is the answer. The architecture is modular: a highly specialized "sub-engine" module can be developed and fine-tuned for each information source. Each sub-engine could be used alone as a search engine, but perhaps with relatively poor performance. But queried in parallel and combined by a metasearch core, each lends its unique perspective on the query and documents, resulting in a high-performance search engine.

This is the scenario that we call *internal* metasearch (Figure 1.2), and it has been the primary motivation for our work. We show that metasearch allows a search engine to be designed as a combination of modular sub-engines by designing our experiments to model this situation. In this way, the performance improvements that we achieve suggest that we can use metasearch to build high-performance search engines composed of simpler sub-engine modules.

Focused ranking algorithms: Effective metasearch techniques may yield an unexpected benefit: they may lead to the invention of focused algorithms for ranking documents that can take advantage of novel, highly-specific information sources within documents. These focused ranking algorithms are not expected to work well in isolation, but they can improve search engine performance when combined with other ranking algorithms [2]. To prove the worth of a new ranking technique R, one can use a standard metasearch algorithm to fuse the results of R with a standard ranking technique. If the fused result significantly improves upon the standard technique alone, then R has indeed tapped a source of novel information.

1.3 Our Work

In the remainder of this work, we present: (1) improvements to standard metasearch algorithms, (2) new metasearch algorithms, and (3) new evaluation techniques for metasearch.

1.3.1 Improvements

Performance Weighting

Training data is not always readily available. But when it *is* available, how can our metasearch algorithms take advantage of it? Dissertations have been devoted to this topic [4, 74], but we take a simple, pragmatic approach that has been used, but has not been thoroughly tested in the metasearch literature: weight by individual performance. That is, test each input system on the training data, evaluate the result with a standard IR metric, and in future, weight each system by the score it received. We show that even this simplistic approach is effective: it substantially improves the performance of standard metasearch algorithms.

Relevance Score Normalization

Most metasearch researchers in the past have concentrated on developing algorithms to combine relevance scores. Before scores can be combined, however, they must be normalized so that they are comparable between systems. The standard normalization technique that has been used is to map the given scores tightly into the range $[0, 1]$. In this work, starting from the simple observation that the standard technique may be sensitive to outliers in the distribution of scores, we design and test two new algorithms that are less outlier-sensitive. From the positive results we obtain we conclude that score normalization is an important phase of the metasearch problem, and that outlier-insensitive normalizations are effective.

1.3.2 New Algorithms

Bayes-fuse

The outlier-insensitive normalization schemes above do improve results over the standard normalization scheme. But they do not address another basic problem: what do we do when an input system does not return one of the documents in question? What score should it be assigned? Faced with this limitation, we have designed a more complete model based on a Bayesian analysis of the metasearch problem. This model does require training data, but it achieves good performance even without relevance scores.

Borda-fuse

Taking a different approach to metasearch than the above, we adapt some results from the field of Social Choice theory to the metasearch problem. Social Choice researchers have extensively studied the problem of voting, with many important algorithms and results. Our basic approach is to identify *elections* with instances of the metasearch problem, *voters* with the input systems, and *candidates* with the documents.

One of the resulting algorithms, Borda-fuse, can be adapted to work with or without relevance scores, and with or without training data. It is not our best-performing algorithm in each circumstance, but it is a simple technique that does outperform the best input system.

Condorcet-fuse

Another voting-based algorithm, Condorcet-fuse, is based on a simple and efficient sorting technique. It works without relevance scores and is one of the best-performing algorithms that we know of, both when training data is available and when it is not.

1.3.3 Evaluation

Consistency

The basic problem of evaluation is to answer the question: does a given metasearch algorithm work? And, how well does it work? We introduce another variation on this question: how consistently does it work? We demonstrate that, on our data sets, not only do standard metasearch algorithms indeed usually have better performance than the best input system, but they have more consistent performance from query to query than the best input system.

Upper Bounds

Another interesting question about metasearch performance: is it optimal? To answer this question, we need a tight upper bound on the performance of metasearch. Toward that end, we present an upper bound model for the metasearch problem, based on the idea of a *constrained oracle*. Given some ranked lists to merge, we present the lists to a metasearch oracle that knows which documents are relevant but is constrained in how it may rank the documents. The constraints are chosen to capture natural limitations of fusion algorithms. Using this model, we develop three different upper bounds, each of which bound a different class of metasearch algorithms. We experimentally approximate the bounds, with the tentative conclusion that for rank-based metasearch without any training data, the Condorcet-fuse algorithm may be relatively close to optimal performance, at least when a sufficient number and quality of input systems are being combined.

Before we can describe the technical details of our algorithms, however, we must review the relevant literature and describe our experimental apparatus.

Chapter 2

Related Work

The combination of document retrieval results has received considerable attention in the past few years: it has been the subject of several doctoral dissertations [4, 53, 74, 67], journal articles [70, 26, 76], and conference papers [23, 7, 42, 35, 43, 54, 75], being especially used by systems competing in the TREC competitions [25, 68, 56]. This chapter reviews the results of previous studies as they relate to our work.

2.1 Preliminaries

Before we begin looking at specific metasearch techniques we have to make some notational conventions and review some fundamentals. In particular, we review document retrieval evaluation metrics and some common manipulations of relevance scores.

2.1.1 Definitions

Let $r_i(d)$ be the *rank* assigned to document d by retrieval system i. Let $s_i(d)$ be the *score* assigned to document d by retrieval system i. When context makes either d or i clear, we may omit them for clarity, e.g., $r(d)$, $s(d)$, r_i, or s_i. Note also that the rank or score of a document is always with respect to a given query. We do not include the query in our notation, as we never have need to make it explicit.

By convention, rank one is the best possible, rank two is second best, etc. Thus the lower the numerical rank given a document, the higher the confidence that it is relevant. Since lower rank means higher relevance, we avoid the words "lower" and "higher" with respect to ranks, and instead use "better" and "worse"—better means closer to rank one.

Relevance scores work opposite of the way that ranks do: the higher the score, the better. In particular, we assume that the relevance score given to a document by an input search system is a real number in the interval $(-\infty, \infty)$ (see Thompson's CEO model [70] for an example where this is not the assumption). By convention, if $s_i(a) > s_i(b)$, then we know that input system i is asserting that a is more relevant, or more likely relevant than b.

To borrow some convenient terminology from the field of Social Choice Theory which studies voting algorithms, we can call a particular instance of the metasearch problem an *election*. Each election contains a *profile*, the set of ranked lists of documents (along with

their associated relevance scores, if available) returned by each input system. For example, we can display a profile like this:

S_1: $d_1, d_{43}, d_{162}, \ldots$
S_2: $d_{43}, d_{14}, d_{501}, \ldots$
S_3: $d_{67}, d_{923}, d_{501}, \ldots$

In this profile, three input systems, S_1, S_2, and S_3 are participating in the election. The first system, S_1, ranks d_1 first, d_{43} second, and so on. Using our notation, $r_1(d_1) = 1$, $r_1(d_{43}) = 2$, $r_3(d_{923}) = 2$, and so on. Not shown are the associated relevance scores; perhaps $s_1(d_1) = -53.77$.

We call the set of documents returned by input system i its document set, D_i. The document set D for an election is the the union of the document sets returned by the input systems being fused. That is, $D = \cup_i D_i$, where the union is over all input systems. Although each input system is aware of more documents than it retrieves for a particular query, the metasearch engine only has to be concerned with those documents actually returned. Since we assume that each input system is over the same database, any document in D that is not in D_i is implicitly ranked by system i worse than every document in D_i.

With this terminology, we can discuss metasearch more precisely. We begin with the important question of evaluation.

2.1.2 Evaluation

Although there are many metrics used in IR, raw performance of a retrieval system is perhaps most often measured with *average precision*[1]. The *precision* of a set of documents is the fraction of documents that are relevant. Since we are considering not sets of documents but ranked lists, we need a measure that takes the rank of each document into account. We first define *precision at rank i* to be the precision of the set of documents with rank i or better. Now, average precision is defined as the average of the precision obtained at the rank of each relevant document. That is, the average precision of one system's ranked list corresponding to one query is:

$$P_{avg} = \frac{1}{|R|} \sum_{d \in R} \frac{|R_{\leq r(d)}|}{r(d)}$$

where R is the set of all relevant documents and $R_{\leq r(d)}$ is the set of relevant documents with rank $r(d)$ or better. In practice, a search engine truncates its ranked list to the top k documents before evaluation is done; for those relevant documents not returned, a precision of zero is used. Usually researchers report the results of an experiment in terms of *mean average precision*, the mean of the average precision obtained over each query in the experiment. We often call mean average precision itself simply average precision.

[1] We have tried evaluating many of the experimental results that we present later with *average precision, 11-point average interpolated precision,* and *R-precision*, but found qualitatively similar results. Note that Buckley and Voorhees [9] show that average precision and R-precision perform similarly as evaluation measures; they did not experiment with 11-point average interpolated precision.

The goal of fusion can be regarded either as (1) maximizing raw performance, or (2) maximizing improvement over the best input system. We measure improvement as $I = \frac{P_f - P_b}{P_b}$, where P_f is the raw performance of the fusion algorithm and P_b is the raw performance of the best input system being fused. In this work we present some graphs displaying raw performance, and some showing improvement. Both are useful.

2.1.3 Score Normalization

In an attempt to make relevance scores comparable across systems, many fusion algorithms based on relevance scores require that the scores are first mapped into the range [0,1]. For the set of relevance scores assigned to documents by a system, the standard normalized score of document d is:

$$\widehat{s}(d) = \frac{s(d) - \min(s)}{\max(s) - \min(s)},$$

where $\min(s)$ and $\max(s)$ are the minimum and maximum relevance scores obtained by any document. Any document that was not retrieved by a system is assigned a normalized relevance score of zero.

We mention the standard normalization scheme now because it is used by many of the fusion algorithms that we review in this chapter. In Chapter 4 we revisit the thorny issue of score normalization in much greater detail.

2.1.4 TREC

In subsequent discussions of fusion algorithms we frequently refer to to the TREC conferences. TREC[2] (Text REtrieval Conference) is an annual IR competition sponsored by the US National Institute of Standards and Technology (NIST). Each year, teams from academia and industry prepare IR systems capable of handling various IR problems. The TREC "adhoc task" is the relevant one for us: it is query-based document retrieval. The TREC organizers distribute data sets in advance (a few gigabytes of documents, taken mostly from news-wires and Web crawls) and 50 new queries each year. The competing teams then submit ranked lists of documents that their systems gave in response to each query (up to 1000 documents per query). Human assessors at NIST judge documents relevant or irrelevant to each query and score the submitted results based on this assessment.[3] Fortunately for us, the ranked lists are then available for metasearch researchers to download and fuse. TREC 1 refers to the first TREC competition, which took place in 1993, TREC 2 the second, and so on.

Metasearch has also been used for the TREC "routing" task. In this case, the queries are fixed in advance and distributed to the competitors along with some training data (a sample corpus of documents where documents have been annotated as relevant or irrelevant to each query). Then new documents are presented to the systems and they must judge them relevant or irrelevant.

[2] `http://trec.nist.gov/`

[3] They do not judge every document with respect to every query. Instead they use "pooling": they judge the top 100 documents returned by each system. This allows them to reasonably assume that *almost all* unjudged documents are irrelevant.

2.2 Related Problems

2.2.1 Fusion Effectiveness

Metasearch researchers have found a wide range of results using different techniques and different data sets. In some studies metasearch algorithms are not able to consistently beat the best input system, while others achieve up to 50% improvement. Thus, several researchers have tried to characterize when metasearch is effective.

Ng and Kantor [54, 53, 55] review the metasearch literature and conclude that at least these two factors are generally important for successful fusion:

1. *Efficacy*: the systems being fused should be of comparable quality, and
2. *Dissimilarity*: the systems being fused should produce significantly different results.

They measure ranked list similarity by considering the $n(n-1)/2$ ranked pairs defined by a ranked list of n elements, and defining the similarity of two ranked lists as the fraction of ranked pairs they agree on. Pairs unranked by one of the systems are given half credit. They find that a model trained on efficacy and dissimilarity alone is able to predict when fusion will improve performance with about 70% accuracy. Note that Ng and Kantor aim at predicting when metasearch will yield *improvement*, not high *performance*.

Vogt et al. [77] simulate the problem by generating random ranked lists, and seeing which pairs fuse well (using their own fusion techniques that we describe below). They find that improvements are often obtained when

1. both constituent systems perform well, and
2. both rank relevant documents similarly.

This conclusion is consistent with the *improved precision* argument: due to the unequal overlap of relevant and irrelevant documents, relevant documents can be detected.

In later work, Vogt [74] uses linear regression to predict the performance of fusing two systems using linear combination. The regression is based on several measured properties of the systems individually and together. He also models the problem algebraically. Both analyses lead to the conclusion that the raw performance of the fused system will generally be good if:

1. the performance of at least one of the two constituent systems is good,
2. the two systems return similar sets of relevant documents, and
3. the two systems return dissimilar sets of irrelevant documents.

The above work is probably best summarized in Croft's [15] review of the literature surrounding fusion in IR. He concludes that the systems being combined should:

1. have compatible outputs (e.g., outputs on the same scale),
2. each produce accurate probability estimates, and
3. be independent of each other.

The first criterion is technical: the outputs cannot be meaningfully combined unless they are compatible. The second two criterion can be understood simply: if the systems are *independent*, they are contributing different information to the combination. If they are *accurate*, they are contributing a lot of information.

The research above tries to predict when fusion will be effective; we attempt to make it effective.

2.2.2 Collection Fusion

In this work, we do not tackle the problem of combining search results from non-identical databases. However, as it is a closely related problem to our own, we here provide a few pointers into the literature that can serve as a starting point for further study.

Voorhees et al. [80] train a fusion system to retrieve a different number of documents from each input system in response to the query.

Callan et al. [11, 10] apply the INQUERY retrieval system to the collection-fusion problem, using INQUERY for the retrieval within each collection, and merging the resulting ranked lists with straightforward techniques like interleaving documents based on relevance score.

Gravano et al. [30, 31] develop a protocol for metasearch engines, *STARTS*, that allows metasearch engines and search engines to communicate in a structured way. They break the problem into three stages: *database selection*, where the resources appropriate to a given query are determined, *query translation*, where the appropriately formatted queries are submitted, and *result merging*, where the answers are fused.

Database Selection

Xu and Callan [82] use phrase information and query expansion to improve the database-selection stage.

Craswell et al. [13] propose that the fusion system download a selection of the documents from each input system to learn some statistics about each database. They also propose new weighting functions for document features that require the fusion algorithm to download at least the beginning of each document in the fusion.

Meng et al. [51] study the metadata needed from each search engine to accurately select which databases to query.

Garbe [27] presents a client-side metasearch engine, BINGOOO that selects a subset of the search engines to query.

Apple Computer's Sherlock[4] is another client-side metasearch engine. Sherlock allows the user to select a particular category (like news, reference, products, etc) for each query, and the appropriate subset of search engines is used.

Merging Results

To fuse the ranked lists returned by the input systems, the MetaCrawler system[5], as reported in [67], uses a three-phase algorithm called *Normalize-Distribute-Sum*. Starting with scores

[4] `http://www.apple.com/sherlock/`

[5] `http://www.metacrawler.com/`

normalized in the standard fashion (phase 1), the algorithm "distributes" them (a kind of second normalization stage—phase 2) according to the following formula:

$$s'_i(d) = s_i \cdot \frac{\max(r_i) - r_i(d) + 1}{\max(r_i)}.$$

The first term is just the old score for d, and the second term is basically a rank-based, simulated score. In effect, instead of using relevance scores alone, or ranks alone, they use a combination of the two. Finally (phase 3), they assign a final score to a document d as the sum of the scores given d by the input systems.

The SavvySearch metasearch engine [34, 20] (now Search.com[6]), combines results similarly, summing the normalized scores for each document.

The ProFusion metasearch engine[7] [28, 29], in advance of receiving any queries, evaluates the performance of each search engine based on human-made relevance judgments. This performance (a function of the number and ranks of relevant documents in the top ten returned), is used as a weight w_i for each search engine i. Then, for each query, it normalizes the scores for each search engine into $[0, 1]$, and assigns the final score of document d as the maximum weighted score received:

$$s_{ProFusion}(d) = \max_i(w_i \cdot s_i(d)).$$

This approach is similar to a performance-weighted version of CombMax that we present later. ProFusion also uses more sophisticated search engine weighting based on learning performances for each search engine for different kinds of queries.

MetaSEEK [5] uses metasearch to find images on the Web.

Some of these techniques are similar to those used in our problem, data-fusion metasearch, but how well they perform in each scenario may be quite different. We now look at work directly related to our problem.

2.3 Fusion Techniques

2.3.1 Early Studies

In 1972, Fisher and Elchesen [21] showed that document retrieval results were improved by combining the results of two Boolean searches: one over the title words of the documents, and one over manually-assigned index terms. They noted that while the search over the index terms gave better results, the best performance was achieved by using both information sources.

Other studies in the 1970s and '80s [50, 36, 64] showed that using different *query representations* or different *document representations* led to very different retrieval results.

[6] `http://www.search.com/`

[7] `http://www.profusion.com/`

Data-Level Fusion

In 1991, Turtle and Croft [72] proposed the the INQUERY retrieval system, based on inference networks. The inference-network model is a general model for combining evidence; INQUERY falls in the category of data-level fusion, where the documents and queries themselves are available to the fusion algorithm, and thus is outside the scope of our work. The model is based on probabilistic updating of the values of nodes in the network, and many retrieval techniques can be implemented by configuring the network properly. They configure the network to implement both a probabilistic, $tf \cdot idf$ retrieval algorithm and Boolean retrieval, and then use two appropriate representations of each query in an experiment. They find that indeed, using both improves the results. They conclude that "evidence from multiple sources can be combined to improve our estimate of the probability that a document satisfies a query."

Belkin et al. [6] also use the INQUERY retrieval system to experiment with combining runs using different query representations. They manually generate five queries for each topic and find that the more queries that are used, the better the results generally are. See also [59].

Thompson [70, 71] proposes a Bayesian model that he calls the Combination of Expert Opinion (CEO) model, which is related to work on reconciling probability distributions (see [46]). It bears some resemblance to a Bayesian model that we present in Chapter 5, but also differs in several ways. First, unlike the model we present, CEO is not a general-purpose metasearch algorithm but is really a complete search system—thus it uses data-level fusion. It has two basic search components (each driven by Bayesian updating of probabilities) that are fused with a Bayesian formula. Secondly, the fusion algorithm is based on straight probability, not odds—an important feature of our formulation (recall that if the probability of an event is p, then the odds of the event are $\frac{p}{1-p}$). Finally, CEO requires a full probability distribution to be specified for each document (not simply a probability of relevance). As far as we know, the CEO system has never been fully implemented.

2.3.2 Min, Max, and Avg Models

For their submissions to TREC 1, Fox et al. [23] tried an ambitious architecture that used both data fusion and collection fusion: they used each of 5 retrieval systems on each of 5 disjoint collections. They first merge results for each sub-collection (data fusion step), and then merge the results across collections (collection-fusion step). The final collection-fusion step is solved by simply ranking documents according to relevance scores, despite the inherent differences in the scores due to the differences between the collections. For the data-fusion step, they experiment with generating a final ranking based on the minimum rank a document receives from the constituent systems. They also experiment with merging lists based on iteratively selecting the remaining document most likely to be relevant, where probability estimates are based on average precision numbers for each system. They combine the systems' probability estimates simply: a document's probability is the maximum of the probability estimates that each system ascribes to it. Neither technique demonstrated improvement over the best constituent system.

In TREC 2, Fox and Shaw [25] experimented with using relevance scores instead of mere ranks to fuse ranked lists. They tried fusing based on the unweighted[8] min, max, median, or sum of each document's normalized relevance scores over the constituent systems (documents not returned by a system are assigned a relevance score of zero for that system). They also experimented with ways to more or less heavily weight $n(d)$ (the number of systems that returned a given document d) by using the formula

$$s(d) = n^{\gamma}(d) \sum_i s_i(d)$$

for $\gamma \in \{-1, 0, 1\}$. The sum is over the constituent systems. When $\gamma = -1$, the result is equivalent to the average similarity over systems that returned d, "CombANZ" (Average-of-Non-Zeros). When $\gamma = 0$, the result is the sum of the similarities, "CombSum" (assuming similarity of 0 for docs not returned as they do, CombSum is equivalent to the average similarity over all systems—even those that do not return d). And when $\gamma = 1$, the result is the formula they call "CombMNZ" (Multiply-by-Non-Zeros). They found CombSum to be most effective, usually yielding some improvement over the best single system. We present these "Comb" algorithms in more detail in Chapter 3, since they are the most standard algorithms in the field.

Lee showed that CombSum can be used to obtain improved results when combining two runs from the *same retrieval system*, if it uses different parameters for the two runs [42], or using different query expansion techniques [44]. He also [43] performed experiments with the other Comb algorithms, arguing that "different runs retrieve similar sets of relevant documents but retrieve different sets of non-relevant documents." He further argues that the CombMNZ combination rule appropriately takes advantage of this feature of variant systems' ranked lists by heavily weighting common documents. His experiments on a subset of the systems submitted to TREC 3 yield significant improvement for both CombSum and CombMNZ. He also tried other values for γ in the above model, using $\gamma \in \{0, 0.5, 1, 2, 5, 10\}$. He found CombMNZ ($\gamma = 1$) to be the most effective on his data set.

McCabe et al. [49] corroborate Lee's findings that CombMNZ improves performance when the systems being combined return different irrelevant documents.

Ng et al. [56] find that no improvement is gained when fusing two very different routing task algorithms by averaging their normalized relevance scores.

Fox et al. [26] use a fusion formula that incorporates the maximum score a document receives, the minimum score, the average score, and the number of systems that retrieves it.

Hull et al. [35] try metasearch via averaging on the document filtering problem. The four systems they fuse each outputs an estimate for the probability of relevance of each document, so they need not normalize: being probabilities, the scores are directly comparable. They try both directly averaging these probabilities as well as averaging the log-odds ratios, $\log \frac{p}{1-p}$, a technique related to our own Bayesian log-odds formulation. They find that averaging the log-odds does yield improvement and in fact works better than more complicated regression and weighting techniques.

[8] They do suggest that in general weighting the input systems may be helpful.

Note that most of the min, max, and averaging techniques in this section are designed for the data-fusion setting only, not for overlapping or disjoint databases. Assigning a relevance score of zero for a non-retrieved document assumes that each input system has actually evaluated this document, and decided not to retrieve it.

2.3.3 Logistic Regression Model

Savoy et al. [65] train a logistic regression (LR) model over TREC data. The model is as follows:

$$s_{\mathrm{LR}}(d) = \frac{1}{1 + e^{-\alpha - \vec{\beta}_i \cdot \vec{x}_i(d)}},$$

where

$$\vec{\beta}_i \cdot \vec{x}_i(d) = \sum_i \beta_{i1} r_i(d) + \beta_{i2} s_i(d) + \beta_{i3} var_i(d),$$

α, β_{i1}, β_{i2} and β_{i3} are each parameters to be learned for each input system i, and var_i is the variance of the normalized relevance scores for system i. The sum is over the input systems. They train this model using a statistical software package, SAS, and find that over one TREC collection (WSJ2), it achieves performance slightly superior to a linear combination fusion model, improving on the best input system's performance by almost 11%.

2.3.4 Linear Combination Model

Several researchers experiment with simply linearly combining the normalized relevance scores given to each document. That is,

$$s_{\mathrm{LC}}(d) = \sum_i \alpha_i s_i(d)$$

where the sum is over the constituent systems. We refer to this algorithm as *LC*. Note that this fusion model assumes relevance scores are available, and it requires training data to train the weight α_i given to each input system.

In 1988, Fox et al. [24] use a hand-weighted linear combination of vector-space retrieval runs based on different document representations: some using terms, authors, and links between documents like direct reference, co-citation and bibliographic coupling. They find significant improvement over using only one representation.

Bartell [2, 3, 4] trains a linear combination of two systems: a vector space model over single terms only (implemented using SMART), and a phrase-based system. He optimizes the weights with the Conjugate Gradient method [58], using Guttman's Point Alienation statistic as the target function to maximize (it is correlated with average precision, but is differentiable with respect to the fusion parameters α_i—a requirement for Conjugate Gradient and most other multi-dimensional optimization techniques). His data set is unusual: the Encyclopedia Brittanica, simulating queries using article categories. Nonetheless, he achieves the surprising result that although the performance of the phrase system alone is far inferior to the vector space system, the optimal linear combination weights the two

systems roughly equally, with the phrase-system weight slightly higher.[9] He achieves an average of 12% improvement, though he notes that very different weights are optimal for different queries.

In another experiment, Bartell linearly combines three systems over another unusual data set: Compton's MultiMedia Encyclopedia, simulating queries with a set of specific questions that are answered in usually only one article. After fine-tuning the training regimen, he achieves 47% improvement over the best single system.

Bartell also experiments with more complicated neural networks for fusion (note that the linear combination formula can be viewed as a simple neural net), but finds no improvement over linear combination.

Vogt et al. [77, 76, 74, 75] pick up where Bartell left off, performing more experiments with linear combination and neural net fusion methods. Overall, the performance of their linear combinations as tested on TREC adhoc and routing runs is mixed. Sometimes the best input system can be improved upon, especially if only queries with many relevant documents are chosen and input systems very different from each other are fused, but sometimes the best input system cannot be improved upon.

Some representative results come from Vogt's dissertation [74]: On the routing task, where a set of fusion weights can be trained for each individual query, he achieved 15% average improvement fusing two systems when trained by Golden Section Search [58] to optimize the difference between average score on relevant documents and the average score on irrelevant documents. Strangely, he found that if the same setup was trained to optimize average precision (the more standard measure of performance), the resulting system performed 14% worse than the better of the two systems being fused. On the ad hoc task (where queries are not available for training), the best performance achieved with a linear combination is a 2% decrease in performance. One counter-intuitive result of his work is that negative weights are often used in the combination, meaning that in some cases, for some systems, it is better to down-weight documents ranked better. Another important conclusion of his work is that effective fusion is highly query-dependent.

Vogt also trains a neural network model to combine pairs of systems over the ad hoc task and, despite apparent problems with insufficient training data, is able to achieve an average 10% improvement when trained to optimize *exact* precision (one of 9 metrics used throughout the work), using a carefully selected subset of 10 input systems. This neural network approach actually takes a representation of the documents as input (note that this is data-level fusion), as well as the relevance scores reported for each document by each input system.

Finally, Vogt experiments with linearly combining three, four, or ten systems at once for the TREC 5 adhoc problem. For a carefully selected subset of ten of the 61 input systems, and a carefully selected subset of ten of the 50 queries with enough relevant documents, average improvements up to 30% are achieved. We use this data set for some of our experiments.

[9] In spite of this result, later in this work we show that simply weighting systems according to their individual performance is an effective technique.

2.3.5 Probabilistic Argumentation

Picard [57] suggests using a Probabilistic Argumentation System (PAS) for metasearch. PAS is a model of logical reasoning in the presence of uncertainty that uses both probability and logic. For a relatively small data set, he finds that retrieval performance is improved by combining different link structure-based retrieval methods.

2.3.6 Using Document Summaries

Shu and Kak [69] propose the *Amvish* system. It almost completely ignores the relevance scores *and* the ranks given to the documents by the input search systems. Instead, it uses the title and short summary that are commonly supplied for each document by Internet search engines. Somewhat like the common technique of pseudo-relevance feedback, it assumes the top two documents returned by each search are relevant. Then, based on keywords in the titles and summaries of these "relevant" documents, it looks for similar documents to mark as relevant.[10] They only present results for two sample queries using WWW search engines, but the approach itself is interesting. We do not take advantage of title or summary information in this work.

The above chapters introduce the problem and the state of the art of metasearch. Subsequent chapters present our research.

[10] Curiously, Amvish also assumes the document ranked last by each system is irrelevant, and looks for similar documents to mark as irrelevant. It trains a neural network to distinguish the relevant documents from the irrelevant based only on the keywords of the assumed two relevant and two irrelevant documents. Perhaps this is why Amvish assumes the last documents irrelevant—it needs negative training examples.

Part II

Weights and Scores

Chapter 3

Evaluation

In previous chapters we discuss the problem of metasearch and review approaches that have been taken to solving it in the past. In this chapter we explain our basic experimental environment and we present standard metasearch techniques. Along the way we obtain our first novel results: we propose a simple way of taking advantage of training data, and we show that metasearch improves search engine consistency.

3.1 Experiments

Before we can look at our first set of experimental results, we must describe the data sets and experimental setup that we use.

3.1.1 Data Sets

We use systems submitted to the annual Text REtrieval Conference (TREC) as input to our fusion algorithms. TREC offers large, well-respected, standard data sets with many ranked lists for each query, ready to be fused. Also, each system submits runs for 50 queries, enough to allow training and testing. Table 3.1 shows information about the data sets.

Data Set Name	TREC Topic Number	Number of Systems	Avg Precision			
			Min	Max	Avg	St Dev
TREC 3	151–200	40	0.029	0.423	0.257	0.0859
TREC 5	251–300	61	0.004	0.317	0.190	0.0683
TREC 9	451–500	105	0.000	0.352	0.144	0.0779
Vogt	(10 topics)	10	0.225	0.395	0.288	0.0543

Table 3.1: The data sets used in our experiments.

The first column in the table shows the name of the data set. Out of the nine TREC data sets available for our work, we chose the TREC 3 data set because part of it was used by Lee [43], the TREC 5 and Vogt data sets because they were used by Vogt [75], and TREC 9 because it is a more recent data set and contains Web data (previous TRECs

used mostly newspaper and magazine archives and news-wires). We have never tried any metasearch experiments with any data sets other than these four.

For the TREC 3 and TREC 5 data, we used the data submitted to the TREC "adhoc" task. For TREC 9, the adhoc task had been replaced by the "Web" track. Therefore, over that data set we are fusing the results of search engines retrieving Web pages.

The Vogt data set consists of a subset of the TREC 5 data set as defined by Vogt [75]. In particular, it contains only 10 of the 61 TREC 5 systems, and only 10 of the 50 TREC 5 queries. He chose this subset of retrieval systems to maximize diversity as measured by nine similarity criteria; note that some of the measures he used involve knowing which documents are relevant. The systems are: CLTHES, DCU961, anu5aut1, anu5man6, brkly17, colm1, fsclt4, gm96ma1, mds002, and uwgcx0. The queries were chosen for their large number of relevant documents: queries 257, 259, 261, 272, 274, 280, 282, 285, 288, and 289. We include this data set because of its diverse inputs: it more closely models the environment of some future instances of internal metasearch than the other data sets.

The second column of the Table 3.1 indicates topic numbers. In TREC terminology, a "topic" is a query; they are numbered consecutively. Note that each system retrieved up to 1000 documents for each query.

The third column contains the number of search systems that submitted results to TREC that year—this is the number of systems available for us to fuse.

The last four columns in the table show performance information about the retrieval systems that compose each data set. Each system in each data set has one performance number, its mean average precision. For each data set, the minimum, the maximum, the mean, and the standard deviation of these performance numbers are shown. Note that the Vogt data set and the TREC 9 data set are very different: not only is Vogt small while TREC 9 is large, but Vogt has the highest average performance, while TREC 9 has the lowest (only half of Vogt's 0.288). And, Vogt has low deviation, especially given its high performance, while TREC 9 has high deviation, especially given its low performance. Thus the performances of systems in Vogt are similar to each other (a situation that may be advantageous for metasearch) while the performances of systems in TREC 9 vary widely.

Appropriateness

We saw before (see Chapter 1) that the most suitable use of metasearch, internal metasearch, is characterized by identical databases, the availability of relevance scores, independent systems, and long ranked lists. Thus have we chosen our experimental environment: each input system is over the same document set, relevance scores are available, and each system ranked up to 1000 documents for each query. The one tricky condition is the independence of the input systems; how different would the systems be in practice? Presumably more different than the systems submitted to TREC, since in TREC, each one is a complete (albeit experimental) search engine, taking advantage of all of the information that it can. Thus the TREC 3, 5, and 9 data sets probably contain systems that are more dependent, and therefore harder to effectively combine, than in most internal metasearch situations. The Vogt data set, on the other hand, was chosen specifically for the diversity of its inputs; perhaps this data set is more like real-world internal metasearch systems of the future.

Training and Testing

The TREC data that we use for our experiments also includes the correct answers for each query: documents labeled as relevant or irrelevant. We use these answers primarily to evaluate the result of each metasearch experiment that we perform, but we also use it for training purposes. In all of the experiments that we present in this work that require training data, we use two-way cross validation: we use odd-numbered queries as training data, and even-numbered queries as testing data. Then we reverse the data sets and use evens for training and odds for testing. The performance numbers we report are the average of these two. Thus for three of our data sets (TREC 3, TREC 5, and TREC 9) we train on 25 of the 50 queries. Since the Vogt data set only contains ten queries, we train on only five: as it happens, five of the queries Vogt selected are even-numbered and five are odd, so we use the same even/odd, training/testing split.

3.1.2 Procedure

We predominantly use two experimental procedures in our work: one selects random sets of input systems to combine, and the other combines the best input systems.

Random-Sets Experiment

In our first type of experiment, which we call the *random-sets* experiment, we examine the performance of metasearch strategies when combining random groups of retrieval systems. Each data point represents the average value obtained over 200 unique trials (or as many as are combinatorially possible) performed as follows. Randomly select a set of n (for $n \in \{2, 4, ...12\}$) input systems, apply the given metasearch algorithm to these systems, and record the average precision of the metasearch algorithm's output. (Additionally, we record the average precision of the best underlying system in order to meaningfully assess the improvement gained, if any.) This experiment is designed:

1. to test how well a fusion algorithm *usually* performs on a *randomly selected* group of input systems, including groups that contain very poor systems and very similar systems. And,

2. to see how the number of input systems affects the performance of the algorithm.

Best-to-Worst Experiment

In our second type of experiment, which we designate the *best-to-worst* fusion experiment, we use the given metasearch algorithm to combine the best i retrieval systems. In other words, we combine the top two systems and record the performance of the result. We then combine the top three systems, the top four, and so on, up to the the top 20 systems. This experiment is designed to test how the metasearch algorithm performs when combining high-quality systems.

3.1.3 Comb Algorithms

We are now in a position to present our first experimental results. As mentioned in Chapter 2, Fox and Shaw [25] designed some of the most simple, popular, and effective metasearch algorithms to date. Table 3.2 summarizes how each algorithm computes the final score of each document d based on the scores given d by each input system. The input scores are

Name	New relevance score is:
CombMin	minimum of individual relevance scores
CombMed	median of individual relevance scores
CombMax	maximum of individual relevance scores
CombSum	sum of individual relevance scores
CombANZ	CombSum $\div$ num non-zero relevance scores
CombMNZ	CombSum $\times$ num non-zero relevance scores

Table 3.2: Summary of Fox and Shaw's Comb fusion algorithms.

normalized into the range $[0, 1]$ before this computation takes place. After the computation, the final ranking of documents is formed by sorting the documents by their final scores, and truncating to the top 1000 documents.

Experimental results for the Comb algorithms is shown in Figures 3.1 and 3.2. Here, as elsewhere, we present performance plots in blocks of eight graphs: four rows—one for each of our four data sets—and two columns—one for each experimental setup. The random-sets experiment is on the left and the best-to-worst experiment is on the right. Each of the four graphs in a column contains the same set of curves, and they are labeled consistently. A legend appears in only one of the four graphs, where there is room for it—usually in the TREC 9 graph. The x-axes show the number of input systems being combined, and the y-axes show not raw performance in this case, but improvement over the best input system being combined. Also note that the y-axes are consistent throughout the work unless otherwise noted. In this figure, the algorithms' names are prefaced by "su"—this stands for "score-based, unweighted" as opposed to other variants we present below.

Note also that the title of each random-sets graph contains confidence-interval information. If the title says "(max err: x)," this means we are 90% confident that the actual value of each data point in the graph lies within plus or minus x of the value shown.

We can conclude from these figures that CombMNZ, CombSum, and CombMed all perform reasonably well—often outperforming the best input system—while CombMax, CombMin and CombANZ perform poorly.

Since the range of performance from best input system to worst input system is quite large in the TREC 3, TREC 5, and TREC 9 data sets, some of the sets of systems being fused in the random-sets experiment are wildly mismatched with regard to performance. With regard to diversity, except on the Vogt data set, we make no attempt to ensure that the systems being fused are basing their relevance judgments on different information, or are otherwise different. Despite these difficult conditions, CombMNZ is often able to outperform the best input system, sometimes substantially. On the Vogt data set, where the input systems are more diverse, CombMNZ (and, to a lesser extent, CombSum and CombMed)

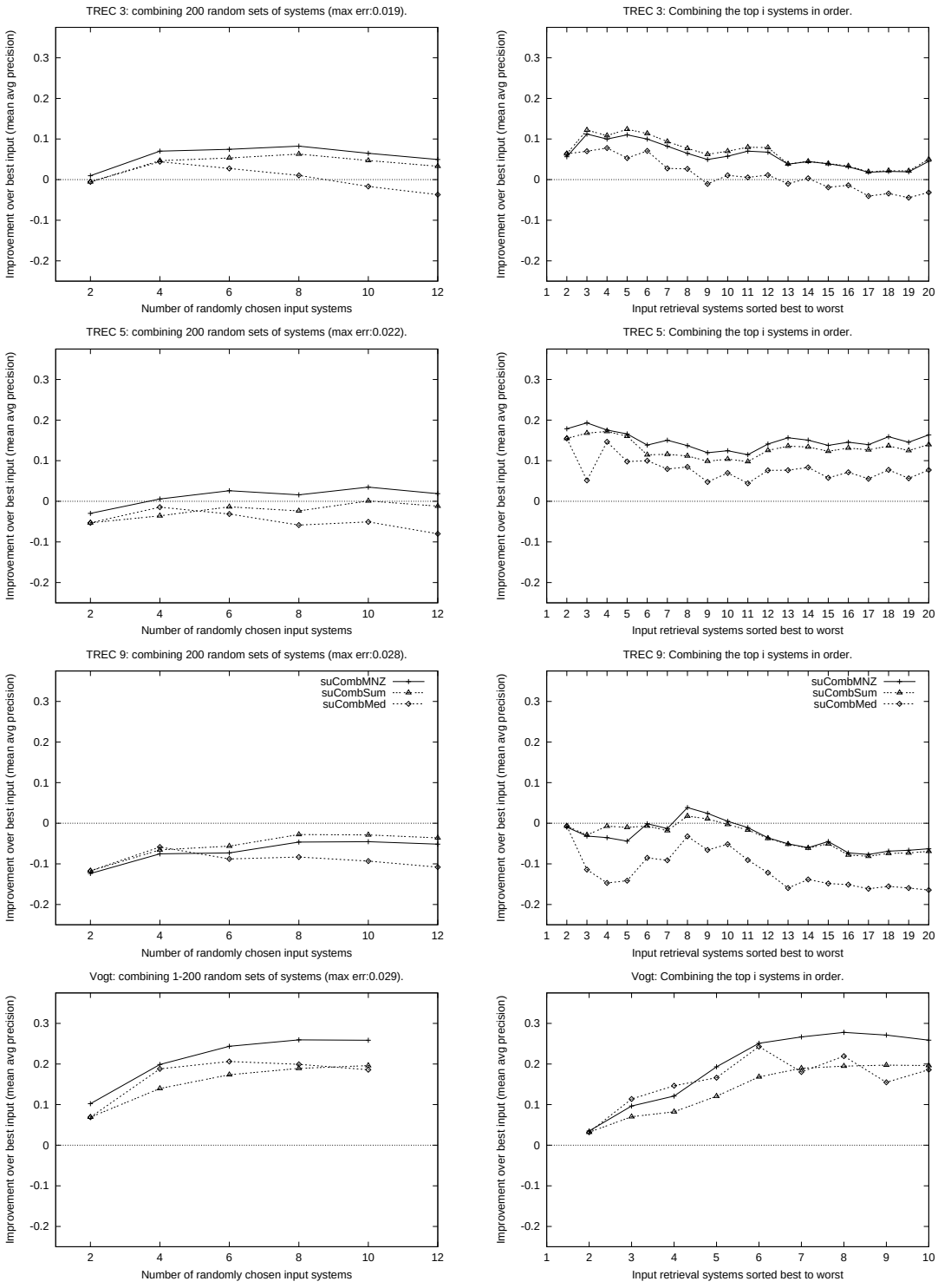

Figure 3.1: The improvement of three Comb systems. The random-sets experiment is shown on the left, best-to-worst on the right. Each row is for a different data set. The x-axis is the number of systems being combined, and the y-axis is improvement.

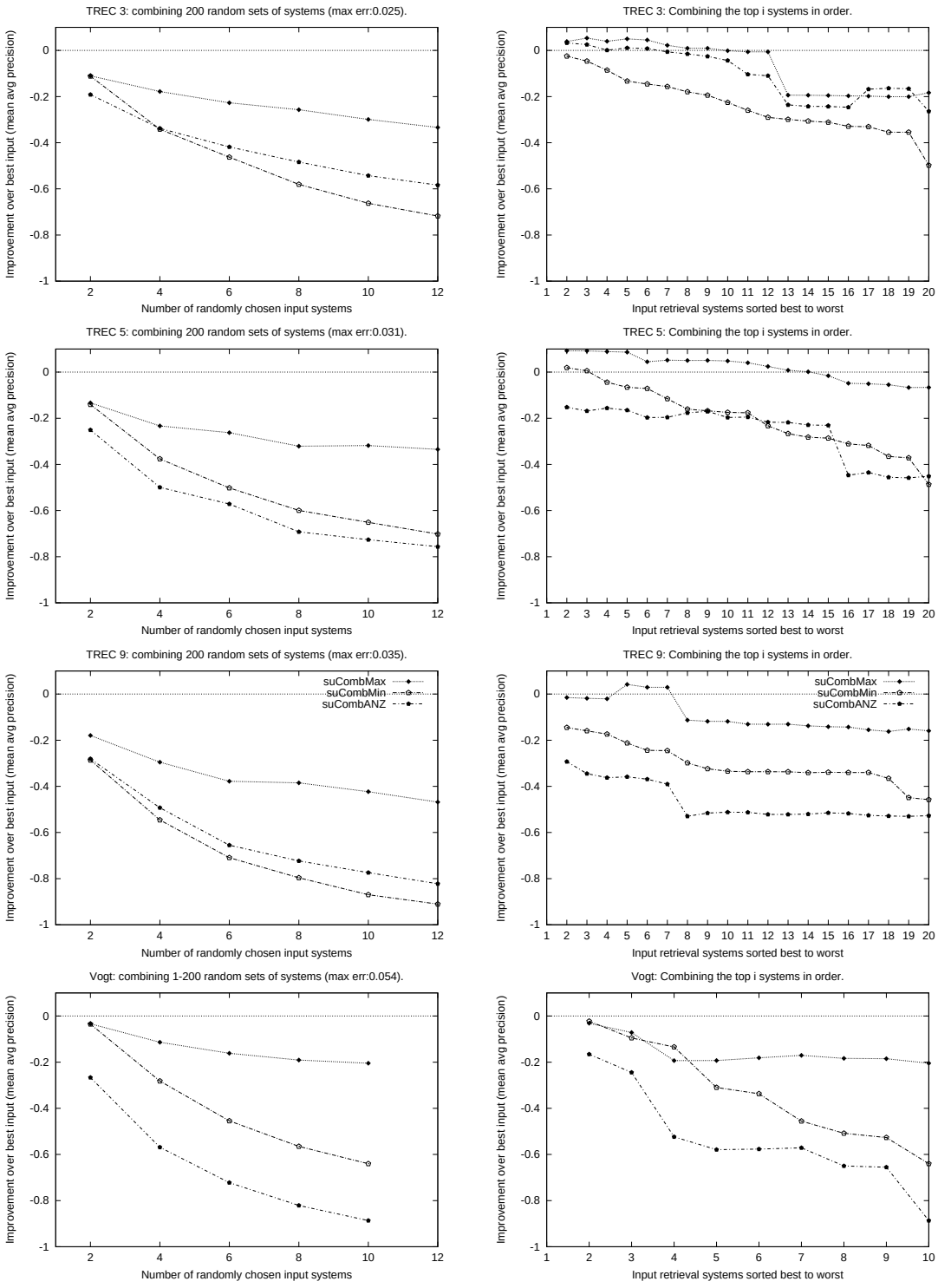

Figure 3.2: The improvement of the worst three Comb systems. The ranges of the y-axes here are unusual due to the poor performance of these systems.

perform especially well. On the TREC 9 data set, however, the fusion algorithms rarely achieve the performance of the best input system.

Two fundamental results are already apparent:

1. metasearch can significantly improve retrieval results, and
2. the performance of reasonable-sounding metasearch algorithms can vary widely (e.g., compare CombSum and CombMin).

3.2 Four Variants

Figure 1.1 shows that the topic of metasearch in the data fusion setting encompasses four different situations: relevance scores may or may not be available, and training data may or may not be available. In this section we present extensions to the Comb algorithms for each input class.

3.2.1 Ranks only: Score Simulation

When we are in a "ranks only" situation, i.e., relevance scores are not available, but we are using an algorithm that requires relevance scores, they may be simulated with ranks. The usual way of doing this is, for a document d retrieved by input system i:

$$s_i(d) = \frac{\max_i(r) - r_i(d)}{\max_i(r) - \min_i(r)},$$

where max's and min's are taken over all occurring ranks for each query. As we define ranks, the min rank is always one. That is, $\min_i(r) = 1$. And on the data sets we use, the max rank is usually (but not always) 1000. Thus, in the normal case,

$$s_i(d) = \frac{1000 - r_i(d)}{999},$$

and s_i ranges linearly from one (for the document ranked best) down to zero (for the document ranked worst). Documents not returned by a system are usually assigned a relevance score of zero for that system. Of course, this simulation does not contain as much information as actual relevance scores would. But when scores are not available for some or all of the input systems, it allows one to use algorithms that require scores. When clarity requires, we differentiate between algorithm variants by prepending "r" to the name of an algorithm when it is using only ranks (e.g., "rCombMNZ"), or "s" when using scores (e.g., "sCombMNZ").

3.2.2 Training Data: Performance Weights

When training data is available, we would like to be able to take advantage of it. As defined, however, CombMNZ does not use training data. In this section we describe a simple but effective way of incorporating training data into a fusion algorithm: weight each input system according to its individual performance. We refer to this as *performance weighting.*

Background

Recall that CombSum can be written:

$$s_{\text{CombSum}}(d) = \sum_i s_i(d)$$

where the sum is over the input systems. Bartell [4] and Vogt [74] both extensively studied a variant of this algorithm in which each input system is assigned a weight α_i:

$$s_{\text{LC}}(d) = \sum_i \alpha_i s_i(d).$$

In their respective dissertations, each researcher tried to optimize metasearch performance through optimization of the α_i's; that is, finding the perfect weights. They used optimization algorithms such as Conjugate Gradient to simultaneously find the set of α_i's that maximizes mean average precision[1]. This approach is partially justified by Bartell's surprising observation when combining two systems: in at least one case, the best performance was achieved with the worse system weighted more heavily than the better. Thus it would seem that the "naive" approach that we take (using individual performance directly as a system's weight) would not be effective. The experiments that we present, however, show that performance weighting is effective: it yields substantial improvement over the unweighted version of each metasearch algorithm.

Performance weighting is not a novel idea; for instance Lee proposes it in [42], and Bartell thought the idea worth dismissing, as explained above. And, it is used in the ProFusion metasearch system [29]. We include it here as an *experimental* result: we present experimental evidence of its effectiveness, which otherwise does not exist in the metasearch literature.

wCombMNZ

We now apply performance weights to sCombMNZ and rCombMNZ. When clarity requires, algorithms that use training data have "w" (for "weighted") prepended to their name (e.g., rwCombMNZ, swCombMNZ). Those that do not use training data have "u" (for "unweighted") instead (e.g., ruCombMNZ, suCombMNZ). Thus, in our notation, the original CombMNZ as defined by Fox and Shaw is written "suCombMNZ."

Let us summarize our procedure. We let α_i be the mean average precision of system i over the set of training queries. For each query in the set of testing queries, swCombMNZ assigns a score to each document d as follows:

$$s_{\text{swCombMNZ}}(d) = n(d) \sum_i \alpha_i s_i(d),$$

where the sum is over the input systems, and $n(d)$ is the number of systems that returned document d. We sort the documents by this score, and report the best 1000. This generates

[1] Actually, since mean average precision is a rank-based evaluation metric, it is not differentiable with respect to the α_i's. Therefore they had to use other performance metrics as target functions—ones that work similarly to average precision but are differentiable.

a ranked list of documents for each query in the test set, allowing us to calculate mean average precision p_1 over the entire set of testing queries. We swap the testing and training sets, and repeat, finding performance p_2. We report the average of p_1 and p_2.

3.2.3 Experimental Results

We now compare the four CombMNZ variants that we use as baselines for the four categories of metasearch algorithms: with or without relevance scores, with or without performance weighting. They are summarized in Figure 3.3.

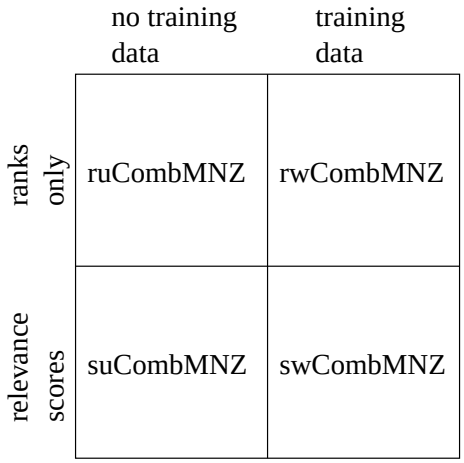

Figure 3.3: The four CombMNZ variants. Note the naming convention: "s" stands for *score,* "r" for *rank,* "w" for *weighted,* and "u" for *unweighted.*

Figure 3.4 shows the performance of the four CombMNZ variants. In the random-sets experiment, using scores and weights allows swCombMNZ to outperform the best input system on each data set. In the best-to-worst experiment, weighting makes almost no difference. And using scores sometimes helps, sometimes hurts.

We conclude that:

1. weighting is important when combining systems of very different performance (as in the random-sets experiment), but not as important when the input systems have similar performance (as in the best-to-worst experiment),
2. performance weighting can make a big improvement, and
3. relevance scores only sometimes improve performance.

3.3 Consistency

Up to here, the graphs we present help us to evaluate the baseline performance of metasearch in terms of the usual performance metric: *mean average precision.* We now turn to a

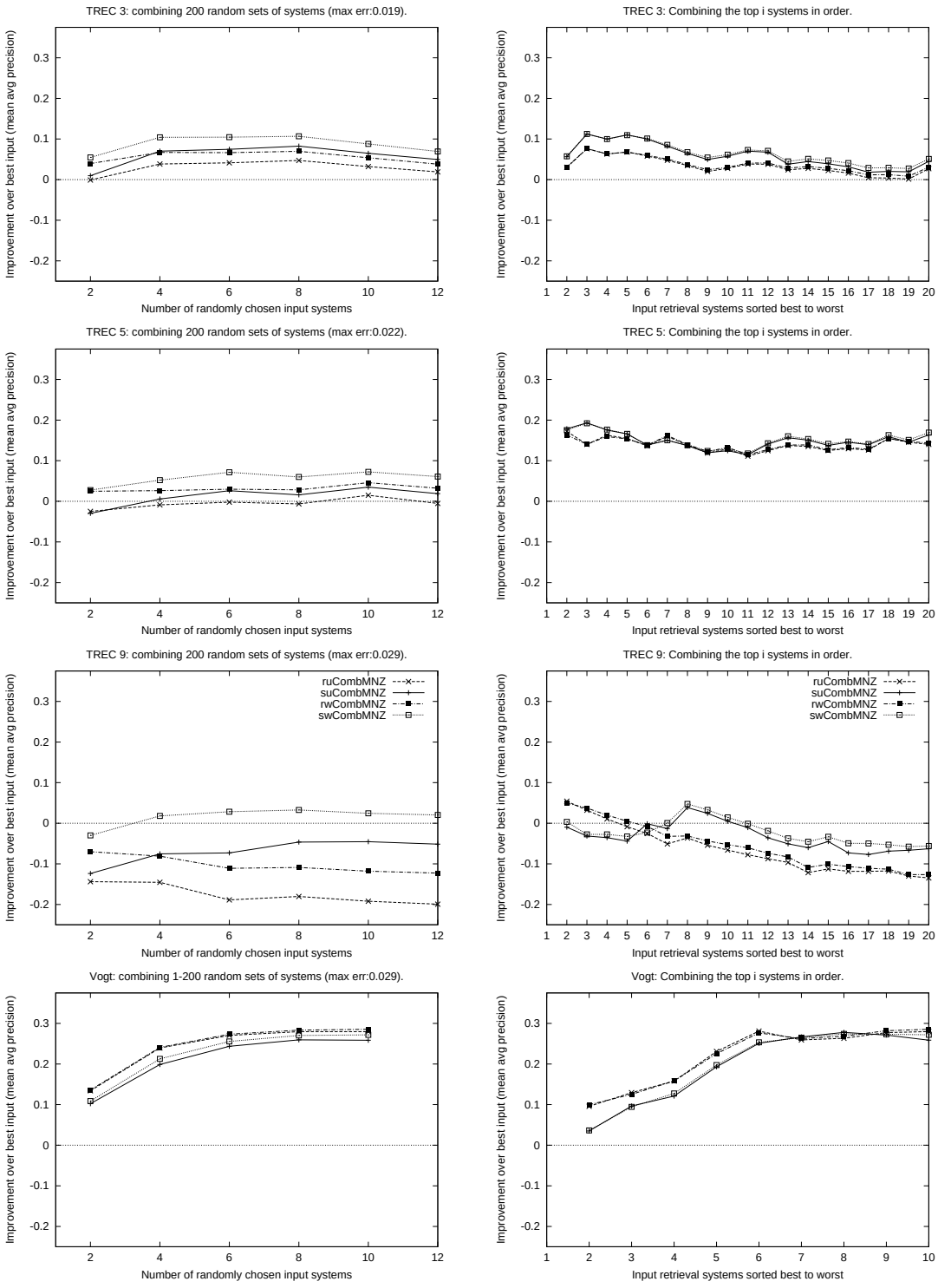

Figure 3.4: The four CombMNZ variants. swCombMNZ performs especially well; in the random-sets experiment it outperforms the best input system, even over TREC 9.

secondary evaluation: how does metasearch affect the *consistency* of performance across queries?

In this section, we investigate the performance of metasearch algorithms in terms of how much they improve *consistency*. We find that metasearch (here represented by CombMNZ), usually improves the consistency of search results; sometimes the improvement is dramatic. Furthermore, consistency tends to improve when performance improves.

We contend that metasearch can improve upon not only the raw performance of its input retrieval systems (e.g., as measured by *average precision*), but it can also ensure that good results are obtained for more queries. *Metasearch can improve the consistency of performance over the queries.* One might expect this to be the case since combining the information gleaned from a number of different sources, each with different strengths, might smooth out performance wrinkles. Our goal is to verify and quantify this effect. As far as we know, no one has previously studied specifically the improvement in performance consistency afforded by metasearch.

3.3.1 Experimental Setup

We use suCombMNZ as our metasearch algorithm here—in our experience, the gains in consistency for other, similarly-performing metasearch algorithms are comparable.

We perform the following variation on our random-sets experiment: Randomly select a set of n (for $n \in \{2, 4, \ldots, 12\}$) input systems, fuse them, and report the resulting list's: (1) average precision, (2) standard deviation of average precision over the queries, and (3) *percent standard deviation:* standard deviation as a percentage of average precision. Repeat this experiment 200 times for each n (or the maximum number of times combinatorially possible), and average the results. See Figure 3.5.

The percent standard deviation measure (known in statistics as the "coefficient of variation") is motivated by the observation that the standard deviation of performance tends to be higher for systems of higher performance. We can more directly compare the variability of two systems by normalizing their performances; e.g., scaling their means to one. This scales their standard deviations appropriately. That is, if $S(X)$ is the standard deviation of a random variable X, then $S(cX) = cS(X)$.

In addition to the performance of each metasearch algorithm, the left column of graphs in Figure 3.5 shows the performance of the "best input system." This is the mean average precision of the best-performing system of those being combined in each of the 200 trials (averaged over each of the 200 trials that goes into each data point). Similarly, the "average input system" is shown—the average performance of all the inputs being combined.

3.3.2 Experimental Results

The second column of graphs in the figure shows that metasearch consistency (as measured by standard deviation) is usually better than the best input system's consistency but worse than the average system's consistency (remember that here, as in column three, lower is better). The third column shows that after accounting for raw performance by normalizing, fused results are generally more consistent (in terms of coefficient of variation) than any of their inputs. Furthermore, as more systems are included in the fusion, consistency generally improves. Finally, when metasearch is able to improve performance over the best

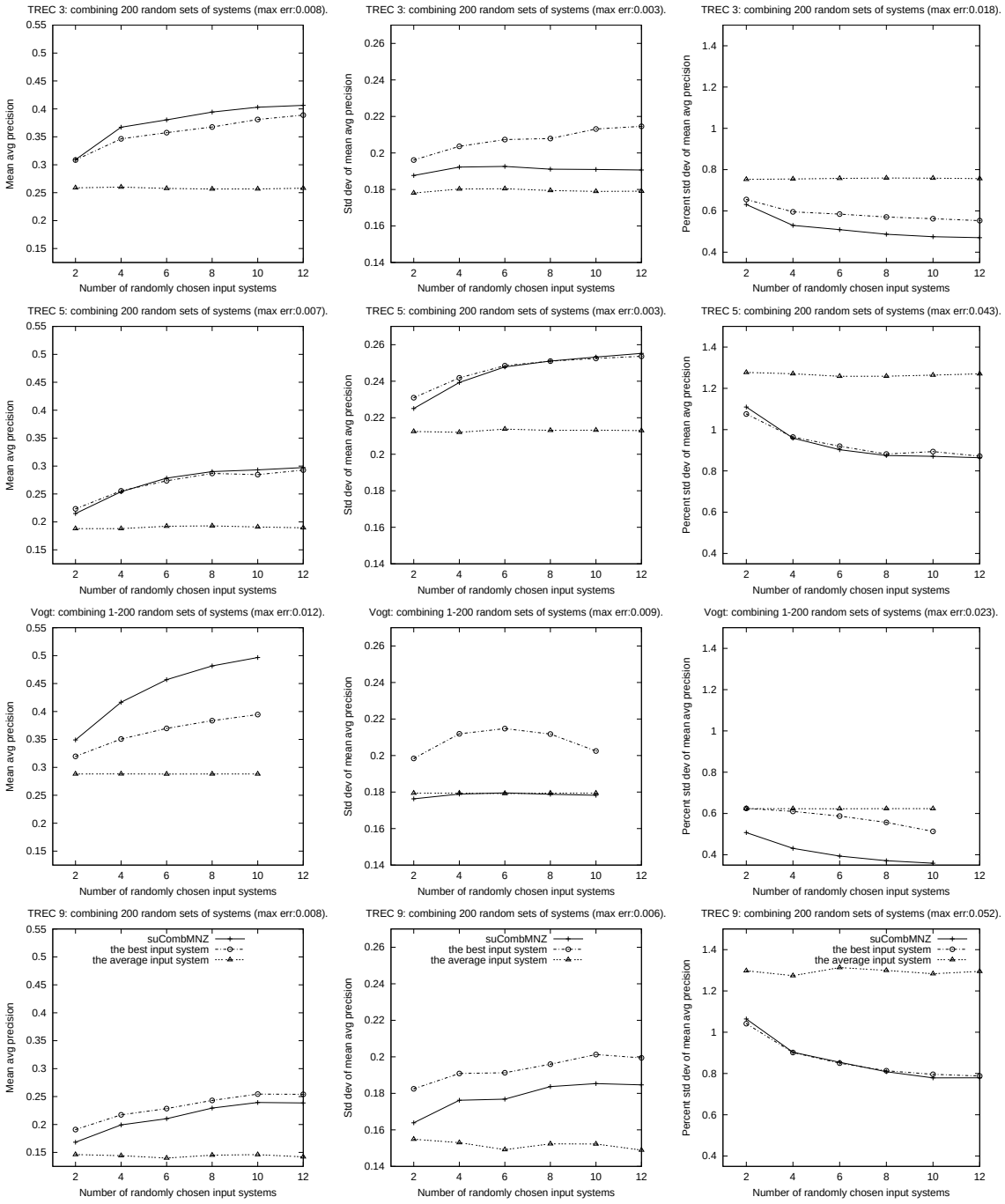

Figure 3.5: Each row of graphs is over one data set: TREC 3, TREC 5, Vogt, and TREC 9. Each column shows one statistic: raw performance, standard deviation, and percent standard deviation (i.e., coefficient of variation). The horizontal axis of each graph is the number of systems being fused; the vertical axis is the statistic of interest. The better metasearch performs (column 1), the more it improves upon the consistency of the best input system (column 3).

input system (see column 1), it is also able to improve consistency (see column 3): TREC 9 and TREC 5 show little or no improvement in either, TREC 3 shows good improvement in both, and the Vogt data set shows large improvement in both.

3.4 Conclusion

In this chapter we describe our experimental setup and procedure, and we present the first two interesting results of our work:

1. We experimented with a simple but effective method of using training data: performance weighting. Performance weighting is easily adaptable to a wide variety of metasearch algorithms, including all of the Comb algorithms and two of the three algorithms that we present in subsequent chapters. It circumvents the complications of simultaneously optimizing the weights for each input system as has been attempted in the past (see [76, 74, 2, 3, 4]). And, it yields sizable improvements when combining systems of varying quality.

2. We show that metasearch improves not only raw performance but also performance consistency.

Now we can begin the discussion of our main work: the improvement of old metasearch algorithms and the development of new ones. First we tackle the problem of relevance score normalization.

Chapter 4

Relevance Score Normalization

The success of performance weighting shows that a simple modification of the standard Comb algorithms can improve their performance. In this chapter we consider another simple modification; we modify the score normalization algorithm. We propose two new normalization techniques and demonstrate empirically that the performance of the CombMNZ and CombSum algorithms can be significantly improved through their use.

4.1 Problem

4.1.1 Decomposition

Normalizing relevance scores is naturally only interesting when relevance scores are available to be normalized. Hence in this chapter we restrict our study to the class of algorithms that use relevance scores. We propose that algorithms of this type can be decomposed at least conceptually into three phases:

Normalization addresses the problem that relevance scores given to the same document by different input systems may be totally incomparable. Although they are usually real numbers, they may be on different scales, in different ranges, and distributed differently. The goal of relevance score normalization is to make relevance scores comparable across input systems. The normalization stage is the topic of this chapter.

Estimation, or *unretrieved document relevance score estimation*, addresses the problem that the input systems return different documents. Since we assume that the input systems' databases contain the same set of documents, if system S does not return a document d, we know S would have given d a lower score than any of the observed scores from S. But how much lower? If we can estimate the score S would have given d, we may simplify the score combination stage: each document will have a score from each system. We raise the issue of estimation in this chapter; we propose a solution in Chapter 5.

Combination is the final relevance score calculation for each document as a function of the newly-comparable (through normalization), and newly-complete (through unretrieved document score estimation) set of input scores.

In most previous work on metasearch, the combination stage has been the focus of the study, with the normalization technique implicitly assumed to be less important. In this chapter, we take the opposite approach: we fix the combination technique to two simple, standard combination algorithms (CombMNZ and CombSum), and experiment with different normalization algorithms.

We find that indeed the normalization stage is important: simple modifications to the standard shift-and-scale normalization scheme yield significant improvements for standard combination algorithms. The key factor seems to be removing the normalization's sensitivity to outliers.

4.1.2 Related Work

Lee performed experiments with CombSum [42] and CombMNZ [43], suggesting that scores be normalized before combination by shifting and scaling them into the range $[0, 1]$ so that they are comparable across systems. This is the standard technique.

Croft [15] underlines the importance of normalization by saying that to achieve good performance when fusing, besides being accurate and independent, the input systems should have compatible outputs. The goal of normalization is precisely to make the input systems' scores compatible.

Bartell [4], Vogt [77, 76, 74, 75], and others experiment with linearly combining the normalized relevance scores given to each document. Their work focuses on the training required to learn the weights to give each system, and uses the $[0, 1]$ normalization without question.

Manmatha et al. [47] have done the most extensive work on normalization for the metasearch problem. They model the distribution of scores for each query, using a mixture model of a negative exponential distribution (for irrelevant documents) and a Gaussian distribution (for relevant). Using an Estimation/Maximization (EM) procedure, they tune the model for each query. This allows them to compute the final normalized relevance score for a document as the probability that it is relevant based on its original score and the model. They find that this normalization scheme yields some improvement.

Independently, Arampatzis and van Hameren [1] developed a similar Gaussian-exponential score-distribution model, which they use to find improved relevance thresholds for the filtering problem.

4.2 Score Manipulations

In this section, we propose two new normalization techniques and describe the algorithms used for the estimation and combination phases.

4.2.1 Normalization

Recall that we assume relevance scores are real numbers in the interval $(-\infty, \infty)$. For each query, each input system returns its top n documents, along with their associated scores. By convention, if $s_i(a) > s_i(b)$, then we know that S_i is asserting that a is more relevant, or more likely relevant, than b. But beyond this, we do not assume that we know how to

Name	Method
Standard	Shift min to 0, scale max to 1
Sum	Shift min to 0, scale sum to 1
ZMUV	Shift mean to 0, scale variance to 1

Table 4.1: Normalization algorithms. We propose the sum and ZMUV norms as simple and effective replacements for the standard norm.

interpret the relevance scores; for example, we do not know if they represent probabilities of relevance, odds of relevance, log odds of relevance, or some other measure. We treat each input system as a black box expert that need not and perhaps cannot reveal how it generated these scores or how to interpret them. Hence the problem: how can we combine the scores if we do not know what they mean? Even if we cannot interpret them individually, the goal of normalization is to make scores directly comparable between input systems.

We propose three desirable qualities of a normalization scheme:

Shift invariant: Let R be a set of relevance scores and R_c be R shifted by an additive constant c. That is, for $s(a) \in R$, $s(a_c) = s(a) + c \in R_c$. Let $s'(a)$ denote the normalized score of document a. Then we say that a normalization scheme is shift invariant if $s'(a) = s'(a_c)$; both the shifted and unshifted set of scores normalize to the same set. In other words, we would like our normalization scheme to be insensitive to mere shifts of the input.

Scale invariant: Similarly, we would like our normalization scheme to be insensitive to its input being scaled by a multiplicative constant.

Outlier insensitive: It is also desirable that normalization not be overly sensitive to the score of a single document. That is, adding a "reasonable" outlier does not significantly change the normalized score for the rest of the documents.

This chapter tests three simple normalization schemes, summarized in Table 4.1: the standard $[0, 1]$ scheme, and two others that we designed to avoid sensitivity to outliers.

Standard Norm: The standard norm is shift and scale invariant, but is sensitive to the max and min scores given for each query, and hence highly sensitive to outliers.

Sum Norm: The sum norm is shift and scale invariant, and it is sensitive only to the min score given for each query. The sum of the scores, as an aggregate statistic, is more robust. In practice the min score is not an outlier, due to the fact that ranked lists are truncated to only return a certain number of documents. So in practice, the sum norm is fairly outlier insensitive. Note also that equivalent results would be obtained using the average score instead of the sum of scores.

ZMUV Norm (Zero-Mean, Unit-Variance): ZMUV is shift and scale invariant. It is also outlier insensitive: the ZMUV transformation does not depend directly on either the min or max scores given (except inasmuch as they individually affect the mean and variance of

the collection). The mean and variance of the relevance scores are both aggregate, and thus more robust, statistics.

Note that none of these normalization schemes requires training data.

4.2.2 Estimation and Combination

In this chapter, since we are studying the normalization phase, we use only the simplest possible score estimators for unretrieved documents. Typically, a normalized relevance score of zero is assigned to unretrieved documents [25], so we also use a normalized relevance score of zero for the standard and sum normalization schemes.

For the ZMUV normalization scheme, where a normalized relevance score of zero would imply average relevance, we instead assign a relevance score of -2 to unretrieved documents. In other words, unretrieved documents are assigned a relevance score two standard deviations below the mean. Note that ZMUV treats unretrieved documents differently than the two other normalization schemes: both others use zero, a score equivalent to that of the worst-ranked document, while ZMUV uses what is probably a lower score. Although this is a factor in the performance that each technique achieves, we still can draw two important conclusions from this work: that the normalization phase is an important area of study, and that outlier dependence is a problem.

Note also that we have not attempted to optimize these constants; in subsequent chapters we address unretrieved document score estimation and its effect on metasearch.

For the combination phase, we use the standard CombMNZ and CombSum schemes, as originally defined (suCombSum). Refer back to Table 3.2 for details. Usually they are used with the standard norm, and unretrieved documents are assigned a relevance score of zero.

4.3 Experimental Results

Figure 4.1 shows the results of using different normalization schemes with CombSum. The ZMUV norm almost always performs significantly better than the standard norm. The ZMUV norm also usually outperforms the sum norm. Indeed, on TREC 5, both ZMUV and the sum norm cause CombSum to outperform the best input system, whereas it had not previously.

Figure 4.2 shows the results of using different norms with CombMNZ. ZMUV performs very poorly with CombMNZ, but this is easily understood: CombMNZ assumes all relevance scores are positive. But, since the ZMUV norm shifts the mean to zero, roughly half of the scores it produces are negative. Thus the final step in CombMNZ—multiplying by the number of systems that returned a document—actually penalizes those documents with negative scores whenever they are returned by many systems.

The sum norm, however, almost always improves CombMNZ, sometimes significantly. In TREC 5, for instance, the standard norm has trouble reaching the performance of the best input system, but the sum norm easily exceeds it.

To explore the problem of CombMNZ interacting poorly with ZMUV, we tried a "2MUV" normalization scheme: shift the mean to two instead of zero (still using unit variance). This will force most scores to be positive, so that CombMNZ can handle them properly. The

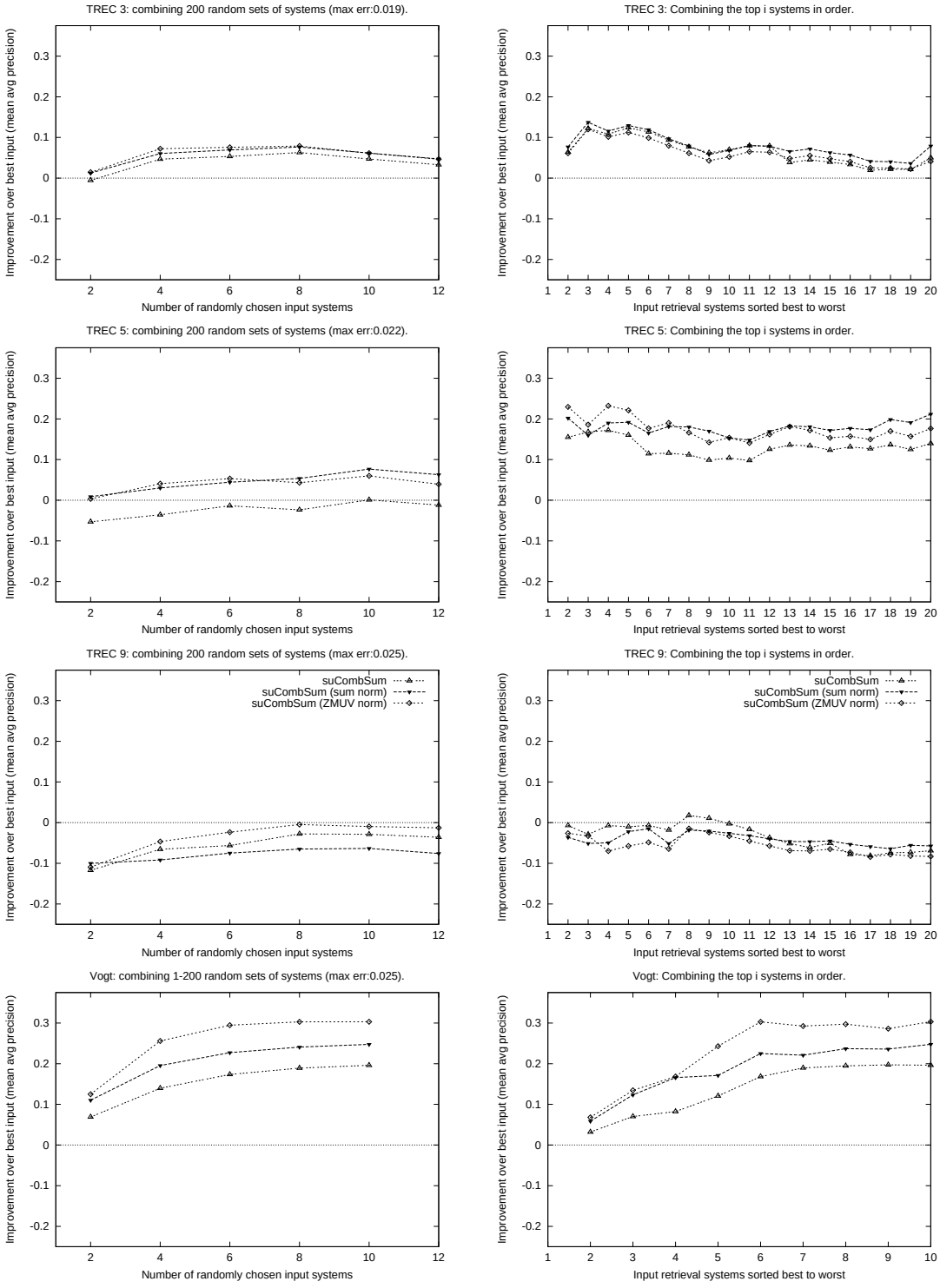

Figure 4.1: The improvement of CombSum using the three different score normalization schemes. The ZMUV normalization almost always works best.

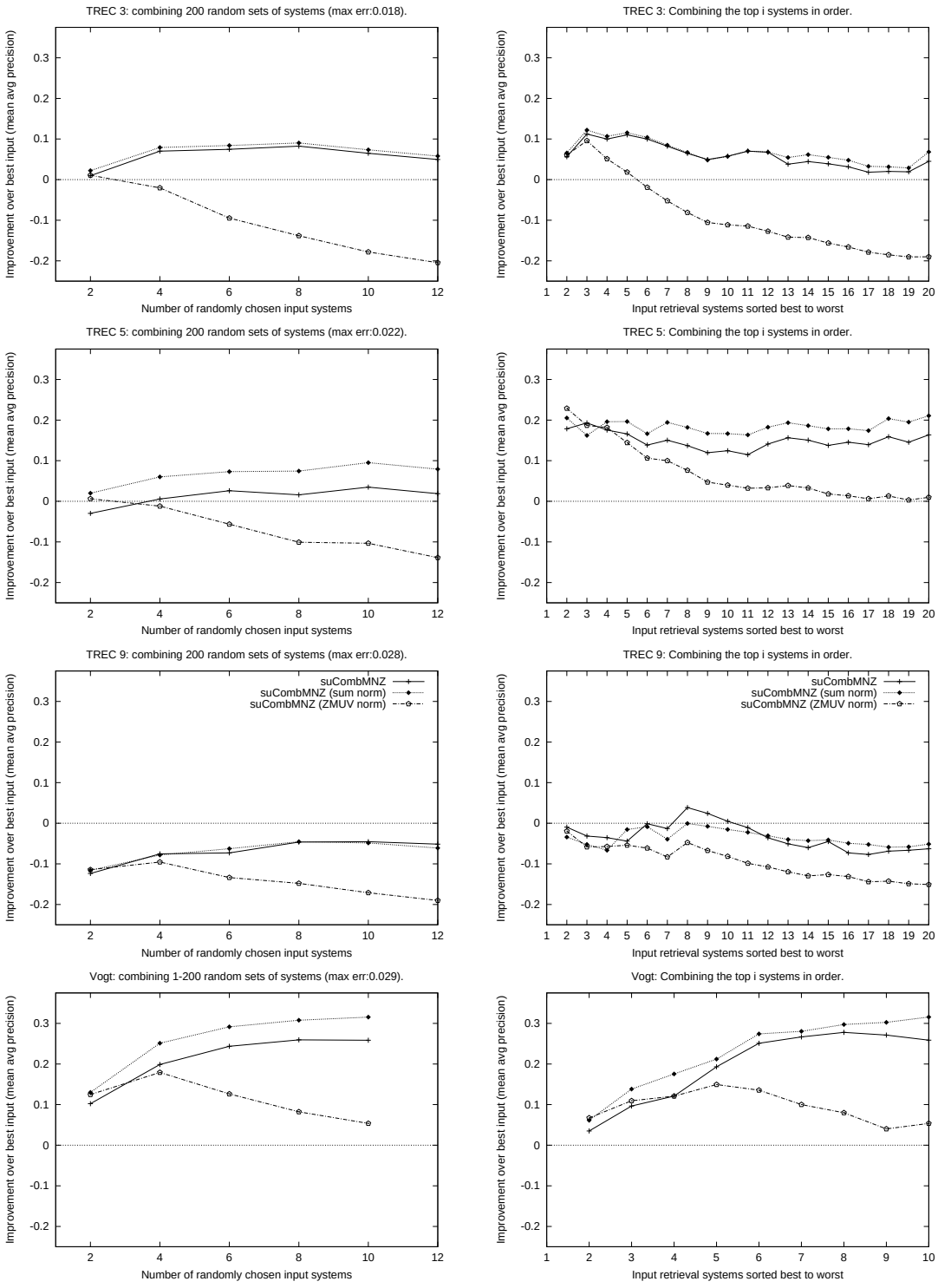

Figure 4.2: The improvement of CombMNZ with different normalizations. The ZMUV norm interacts poorly with CombMNZ, but the sum norm usually improves CombMNZ.

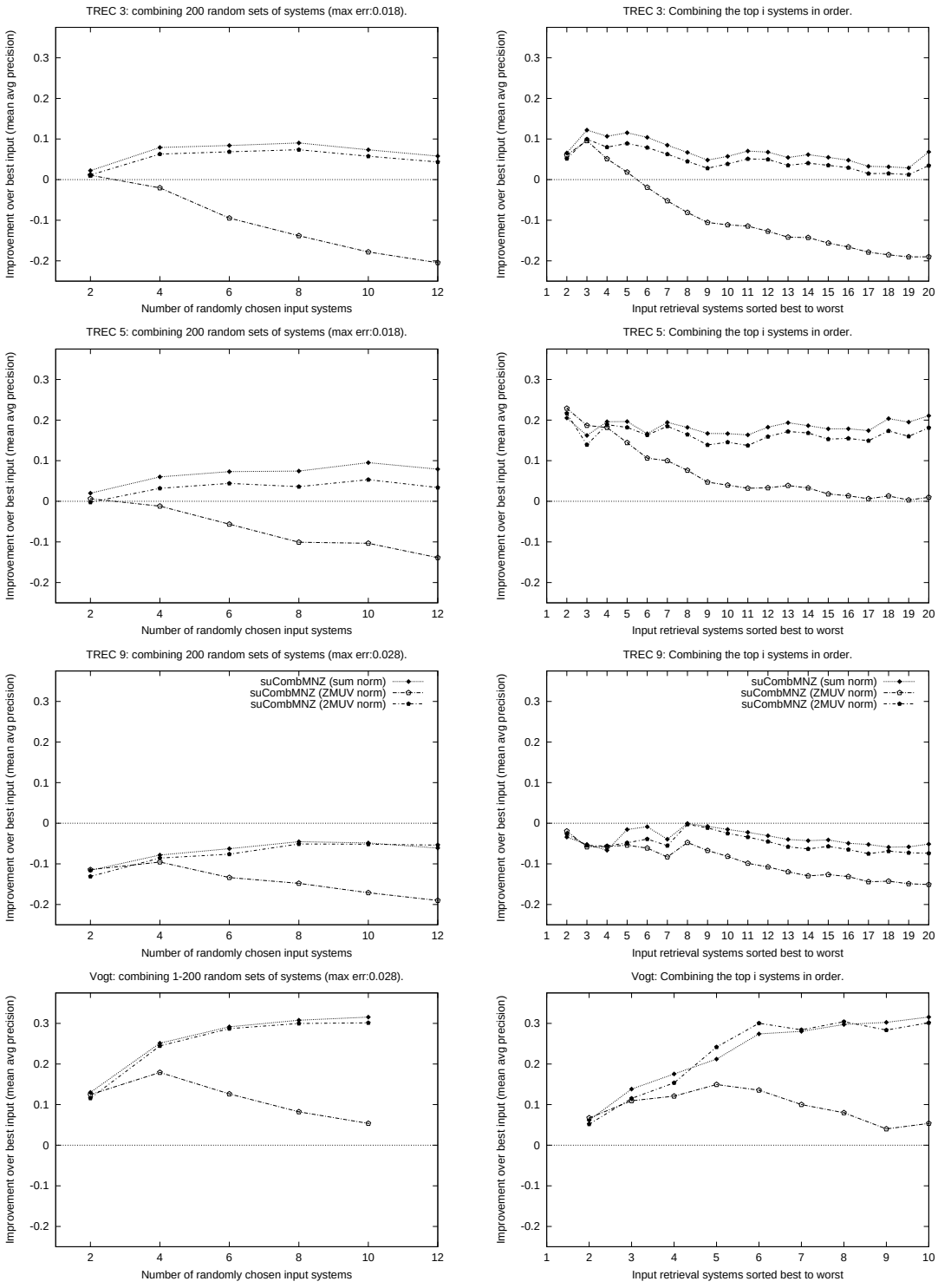

Figure 4.3: The 2MUV variant of the ZMUV norm with CombMNZ. 2MUV solves the problem of ZMUV with CombMNZ, but the sum norm tends to work at least as well.

results are shown in Figure 4.3. Here we can see that the ZMUV norm can be made to work with CombMNZ, though the sum norm usually works about as well.

4.4 Conclusion

We may conclude that relevance score normalization is an important step in the metasearch problem. By simply using more robust statistics than max and min in the normalization scheme we can achieve significant improvements in the performance of both CombMNZ and CombSum. The inferior performance of the standard shift-and-scale normalization scheme is likely due to its dependence on the value of outliers (the min and max relevance scores); normalization schemes that are not as sensitive to outliers (such as those proposed) yield better performance.

Interestingly, Manmatha and Sever have recently re-interpreted the sum normalization scheme as approximately equalizing the distribution of *irrelevant* document scores [48]. Since in TREC data sets the vast majority of documents retrieved are irrelevant, the mean of the entire distribution of scores is very close to the mean of the distribution of scores of only irrelevant documents. They interpret the distribution of irrelevant documents as a negative exponential function [47], and suggest that mapping the minimum score to zero and average score to one (equivalent in terms of final rankings to setting the score sum to one) is an effective way of normalizing negative exponential distributions.

In this chapter we do not address the proper estimation of normalized relevance scores for unretrieved documents. Simply assigning a normalized relevance score of zero—equivalent to the relevance score assigned to the worst-ranked document—seems overly optimistic, though this is classically done for the standard normalization scheme and we have adopted it for the sum normalization scheme as well. Assigning a normalized relevance score two standard deviations below the mean for unretrieved documents in the ZMUV scheme, while reasonable, is probably non-optimal.

In this chapter, we successfully improve metasearch performance in a somewhat ad hoc fashion: we made some simple observations about the behavior of current normalization schemes, in particular that they are sensitive to outliers in the input, and attempted to alleviate this problem by making relatively minor modifications. We did not derive a model that informed us as to how to normalize properly, and without such a model it is not clear how to estimate scores for unretrieved documents. Indeed, it appears that in such a model, the two problems of normalization and score estimation would be inexorably linked; to try to separate the two, while conceptually useful, may not be practically possible. Thus in the next chapter we start from scratch with a probabilistic model that addresses both normalization and estimation simultaneously.

Chapter 5

Bayes-fuse

This chapter examines the metasearch problem from first principles: instead of improving standard techniques as in previous chapters, we derive a probabilistic model for metasearch. When training data is available, the model provides simple, rigorous solutions to all three stages of the metasearch problem: score normalization, score estimation for unretrieved documents, and score combination.

Although we describe the model assuming that the inputs contain ranks instead of relevance scores, the model itself can handle either one. The use of ranks or scores is a choice that can be made in a particular implementation; it only depends on how one estimates some basic probabilities from the training data.

5.1 The Probabilistic Model

5.1.1 Derivation

Given the ranked lists of documents returned by a set of n retrieval systems, recall that $r_i(d)$ is the rank assigned to document d by retrieval system i (a rank of ∞ may be used if document d is not retrieved by system i). We take this rank as the *evidence of relevance* provided to the metasearch strategy concerning document d. Let

$$\begin{aligned} P_{\text{rel}} &= \Pr[\text{rel}|r_1, r_2, \ldots, r_n] \quad \text{and} \\ P_{\text{irr}} &= \Pr[\text{irr}|r_1, r_2, \ldots, r_n] \end{aligned}$$

be the respective probabilities that any document ranked $r_1, r_2, \ldots, r_n$ by the n input systems is *relevant* and *irrelevant*. The Bayes optimal decision rule for determining the relevance of a document dictates that a document should be assumed relevant if $P_{\text{rel}} > P_{\text{irr}}$ and irrelevant otherwise. Since we are interested in *ranking* the documents, we compute the *odds* of relevance

$$O_{\text{rel}} = P_{\text{rel}}/P_{\text{irr}}$$

and rank documents according to this measure. Applying Bayes' rule, we obtain

$$\begin{aligned} P_{\text{rel}} &= \frac{\Pr[r_1, r_2, \ldots, r_n|\text{rel}] \cdot \Pr[\text{rel}]}{\Pr[r_1, r_2, \ldots, r_n]} \quad \text{and} \\ P_{\text{irr}} &= \frac{\Pr[r_1, r_2, \ldots, r_n|\text{irr}] \cdot \Pr[\text{irr}]}{\Pr[r_1, r_2, \ldots, r_n]}. \end{aligned}$$

While the $\Pr[r_1, r_2, \ldots, r_n]$ term would be difficult to assess in practice, it is eliminated in the odds formulation

$$O_{\text{rel}} = \frac{\Pr[r_1, r_2, \ldots, r_n|\text{rel}] \cdot \Pr[\text{rel}]}{\Pr[r_1, r_2, \ldots, r_n|\text{irr}] \cdot \Pr[\text{irr}]}. \tag{5.1}$$

By making the simplifying assumption that the search engines are independent[1] and taking logs, we get

$$\begin{aligned} O_{\text{rel}} &= \frac{\prod_i \Pr[r_i|\text{rel}] \cdot \Pr[\text{rel}]}{\prod_i \Pr[r_i|\text{irr}] \cdot \Pr[\text{irr}]}, \\ \log O_{\text{rel}} &= \sum_i \log \frac{\Pr[r_i|\text{rel}]}{\Pr[r_i|\text{irr}]} + \log \frac{\Pr[\text{rel}]}{\Pr[\text{irr}]}. \end{aligned}$$

Finally, since we are solely concerned with *ranking* the documents, we may drop the $\Pr[\text{rel}]/\Pr[\text{irr}]$ term which is common to all documents. This term changes each document's final score by the same amount, leaving ranks unaffected. This yields our log-odds relevance formula

$$s_{\text{Bayes}}(d) = \sum_i \log \frac{\Pr[r_i(d)|\text{rel}]}{\Pr[r_i(d)|\text{irr}]}. \tag{5.2}$$

In summary, recall that $\Pr[r_i(d)|\text{rel}]$ is the probability that document d would be given rank r_i by system i if it were relevant. Similarly, $\Pr[r_i(d)|\text{irr}]$ is the probability that d would be given rank r_i by system i if it were irrelevant. To obtain the relevance of a document for ranking purposes, we simply sum the log of the ratio of these probabilities over all systems.

5.1.2 Implications

A few important notes are in order. First, in the model we used $r_i(d)$, the *rank* given by system i to document d. But we could just as well have used the *score* given by system i; in either case it is simply a piece of evidence of relevance provided to us by system i. The model itself does not change, only our method of estimating the probabilities changes.

Second, note that the model includes a method of estimating the score of unretrieved documents. That is, we do not estimate the *score* of unretrieved documents, but instead we estimate $\Pr[r_i(d) = \infty|\text{rel}]$ directly; the probability that a relevant document was not retrieved by system i (likewise for an irrelevant document). In this way the model solves the problem posed at the end of the last chapter: it simultaneously solves the problems of normalization (since *probabilities* are estimated for each system) and of unretrieved docu-

[1] This is the common *naive Bayes* independence assumption, which is an approximation.

ment score estimation. At least in our implementations, however, this solution depends on training data.[2]

Third, this derivation is similar to many other probabilistic models that have been proposed for different problems in IR, e.g., the "binary independence model" (see the excellent review of Bayes-based IR techniques by Lewis [45]).

Finally, Cooper has pointed out [12] that our derivation need not rely on the two independence assumptions we said it did, namely:

$$\begin{aligned}\Pr[r_1, r_2, \ldots, r_n|\text{rel}] &= \prod_i \Pr[r_i|\text{rel}] \quad \text{and} \\ \Pr[r_1, r_2, \ldots, r_n|\text{irr}] &= \prod_i \Pr[r_i|\text{irr}].\end{aligned}$$

Instead, the derivation can rely on the weaker (and hopefully therefore less false) so-called "linked dependence assumption":

$$\frac{\Pr[r_1, r_2, \ldots, r_n|\text{rel}]}{\Pr[r_1, r_2, \ldots, r_n|\text{irr}]} = \prod_i \frac{\Pr[r_i|\text{rel}]}{\Pr[r_i|\text{irr}]}.$$

Of course, this changes only the derivation, not the final model.

5.1.3 Another Perspective

Hull et al. [35] provide an alternate method of computing the log odds of relevance (following our notation):

$$\log O[\text{rel}|r_1, r_2, \ldots, r_n] = \sum_{i=1}^{n} \log O[\text{rel}|r_i] - (n-1) \log O[\text{rel}]$$

This formula can be proven by induction as follows. The base case is trivial. Now if we assume that the formula holds for $n-1$, we can show that it holds for n.

$$O[\text{rel}|r_1 \ldots r_n] = \frac{\Pr[\text{rel}|r_1 \ldots r_n]}{\Pr[\text{irr}|r_1 \ldots r_n]}$$

$$= \frac{\Pr[r_1 \ldots r_n|\text{rel}] \Pr[\text{rel}]}{\Pr[r_1 \ldots r_n|\text{irr}] \Pr[\text{irr}]}$$

(assuming independence)

$$= \frac{\Pr[r_1 \ldots r_{n-1}|\text{rel}] \Pr[r_n|\text{rel}] \Pr[\text{rel}]}{\Pr[r_1 \ldots r_{n-1}|\text{irr}] \Pr[r_n|\text{irr}] \Pr[\text{irr}]}$$

$$= \frac{\Pr[\text{rel}|r_1 \ldots r_{n-1}] \Pr[\text{rel}|r_n] \Pr[\text{irr}]}{\Pr[\text{irr}|r_1 \ldots r_{n-1}] \Pr[\text{irr}|r_n] \Pr[\text{rel}]}$$

[2] It is conceivable that the Bayes-fuse model can be used without training data. The basic probabilities could be estimated based on a theoretical model (like Manmatha et al.'s mixture model [47]) instead of actual previous queries with relevance judgements. Our implementations use training data.

$$= \frac{O[\text{rel}|r_1 \ldots r_{n-1}]O[\text{rel}|r_n]}{O[\text{rel}]}.$$

So,

$$\log O[\text{rel}|r_1 \ldots r_n] = \log O[\text{rel}|r_1 \ldots r_{n-1}] + \log O[\text{rel}|r_n] - \log O[\text{rel}].$$

Applying our inductive hypothesis yields the desired result.

Dropping terms that are the same for all documents gives another formula for calculating the log odds of relevance for ranking, equivalent to our own (we call it HPS for Hull, Pedersen, and Schütze):

$$s_{\text{HPS}}(d) = \sum_{i=1}^{n} \log O[\text{rel}|r_i]$$

Either formulation can be used, depending on which way of computing the basic probabilities is easier. The HPS formulation is convenient if one or more of the input systems actually returns reliable probability-of-relevance estimates as relevance scores. In this case, training data may not be necessary (if the system also supplies an estimate for the probability of relevance of unretrieved documents), since the log odds can be computed directly from the scores.

5.1.4 Implementation

Ranks Only

Several details are left unspecified in the model; in particular, how should we estimate $\Pr[r_i(d)|\text{rel}]$? As we noted earlier, this probability could be estimated by using a model for the distribution of retrieval scores like those in [47, 1]. Or, they can be experimentally determined from the training data by interpreting $r_i(d)$ as either the rank or score given to d by system i. The method that we present here, based on training and using ranks only, is attractive in its simplicity, ease of description and implementation, and only requires a small amount of training data. We do not suppose it is optimal.

We use the results of the `trec_eval` program (available from the TREC Web site) to obtain the data necessary to infer the probability of $\Pr[r_i(d)|\text{rel}]$. An example of the output of this program for the best system in the TREC 3 competition, `inq102`, is given in Figure 5.1. In particular, we use the average precisions at the various document levels together with the average number of relevant documents per query to estimate the probability that a relevant document would be ranked, for example, in the range 31–100. From this we obtain $\Pr[31 \leq r_i(d) \leq 100|\text{rel}]$. We repeat this procedure for each rank bucket {1–5, 6–10,..., 501–1000, 1001–∞}. All unretrieved documents go in the last bucket—this yields the probability estimates for unretrieved documents. Repeating for irrelevant documents gives us all of the basic probabilities we need for Equation 5.2.

We experimented with slightly more sophisticated variants of this method of estimating the basic probabilities, including smoothing and interpolating probabilities between buckets, but we obtained no performance gains over this simple method.

Since this estimation technique is rank-based and uses training data, we refer to it as rwBayes-fuse.

```
Queryid (Num):          50
Total number of documents over all queries
    Retrieved:       50000
    Relevant:         9805
    Rel_ret:          7305
Interpolated Recall - Precision Averages:
    at 0.00         0.8992
    at 0.10         0.7514
    at 0.20         0.6584
    at 0.30         0.5724
    at 0.40         0.4982
    at 0.50         0.4272
    at 0.60         0.3521
    at 0.70         0.2915
    at 0.80         0.2173
    at 0.90         0.1336
    at 1.00         0.0115
Average precision (non-interpolated)
for all rel docs(averaged over queries)
                    0.4226
Precision:
  At     5 docs:    0.7440
  At    10 docs:    0.7220
  At    15 docs:    0.6867
  At    20 docs:    0.6740
  At    30 docs:    0.6267
  At   100 docs:    0.4902
  At   200 docs:    0.3848
  At   500 docs:    0.2401
  At  1000 docs:    0.1461
R-Precision (precision after R
(= num_rel for a query) docs retrieved):
    Exact:          0.4524
```

Figure 5.1: `trec_eval` statistics for TREC 3's `inq102`.

Relevance Scores

To use relevance scores in Bayes-fuse, we need a way of estimating $\Pr[s_i(d)|\text{rel}]$—if we use our formulation—or $\Pr[\text{rel}|s_i(d)]$ if we use the HPS formulation. We have experimented with some of both, but so far our best-performing estimation technique uses the HPS formulation.

Given a score $s_i(d)$, we want to estimate the probability that d is relevant. To do this, we simply look at scores near $s_i(d)$ from our training data. That is, out of all the scores that system i returned for any query in our training set, we examine the q scores above $s_i(d)$ and the q below. If p of them corresponded to relevant documents, we estimate that $\Pr[\text{rel}|s_i(d)] = \frac{p}{2q}$.

We refer to the resulting algorithm as swBayes-fuse.

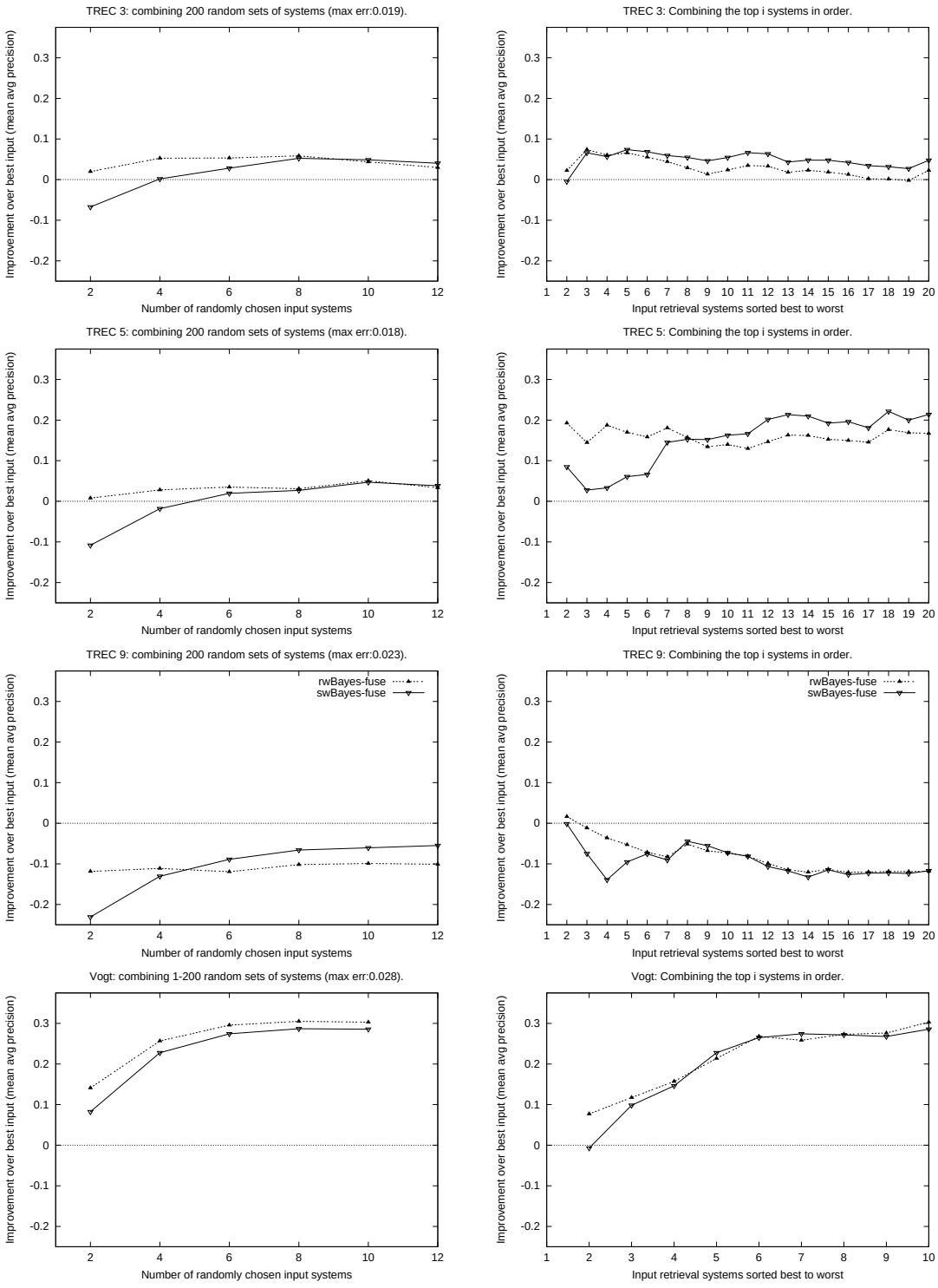

Figure 5.2: The improvement over the best input system given by rwBayes-fuse and swBayes-fuse. Rank-based Bayes-fuse outperforms score-based.

5.1.5 Experimental Results

Experimental results are shown in Figure 5.2 in terms of improvement over the best input system. Rank-based Bayes-fuse outperforms score-based, at least until a sufficient number of systems are being combined. The inferiority of the score-based version is perhaps due to the different estimation techniques each version uses.

5.2 Conclusion

This chapter presents a new model for metasearch based on Bayesian analysis (or, a new formulation of an old model—under different circumstances, Hull et al. [35] independently derived a different formula to estimate the same log odds of relevance for the filtering problem in 1996). Bayes-fuse has a few virtues:

1. It is based on a clear, simple, and rigorous mathematical model.
2. It naturally addresses the relevance score normalization problem and the unretrieved document score estimation problem simultaneously.

At this stage in its research, however, it also has some shortcomings:

1. It requires training data. When such data is not available to estimate the basic probabilities required, Bayes-fuse cannot be used.
2. In its current implementation, the availability of relevance scores does not improve its performance.

Despite these disadvantages, the Bayes-fuse algorithm is worth consideration: the importance of having a clear, mathematical model of a problem should not be underestimated. This particular model—the naive Bayes model—relates our problem, metasearch, more closely to a host of other problems in the field of Information Retrieval that have been studied from a Bayesian perspective (again, we recommend Lewis [45]). This connection may lead to further analogies and insights into the deeper structure of both problems.

To improve the performance of Bayes-fuse, we first need better probability estimation techniques. The basic conditional probabilities can be estimated in many ways, and are fundamental to the model. It would be especially interesting to apply a score-distribution model like those found in [47, 1] and use Bayes-fuse without training.

Another potential route to improvement lies in addressing the naive Bayes assumption that the input systems are independent. We know that this assumption is only an approximation, but we also know that in other IR problems the independence assumption has proved difficult to successfully relax. See Lewis [45] for many examples in IR of models that account for dependence but do not yield performance improvements large enough to be used in practice. One problem is that models that account for the possibility of dependence in their inputs must be much more complex, more expressive models, and this in turn means that in order to work, they require much more training data—a precious commodity.

But, relaxing the independence assumption is not the only approach. As Lewis points out, to solve this dependence problem, one can either (1) try to make a more sophisticated

model that accounts for the dependence directly, or (2) try to eliminate the dependence in the inputs themselves.

If option (1) has largely failed, option (2) seems more promising. If we can detect the dependence in our inputs, perhaps we can eliminate it, and thereby improve performance.

Finally, note that the problem of dependent input systems is not unique to Bayes-fuse. Metasearch is the art of finding agreement within diversity—diversity is a precondition. We encounter this problem again in Chapter 7, and experiment with a provisional solution. One of the strengths of a clear mathematical model is that it makes one's assumptions explicit. Bayes-fuse has brought the problem of input dependence in metasearch to the fore, and perhaps this will be its biggest contribution.

Part III

Voting

Chapter 6

Borda-fuse

As we noted in Chapter 1, voting algorithms are examples of data fusion: they make decisions by combining the opinions of several "experts". In this chapter and the next, we apply two of the most prominent voting algorithms from Social Choice Theory, namely the Borda Count and the Condorcet algorithm, to the metasearch problem. Despite the strong connection between metasearch and Social Choice, none of the previous work in metasearch has made use of the many important results from the much older Theory of Social Choice. Thus we begin with a short introduction to Social Choice.

6.1 Social Choice

The field of *Social Choice Theory* is concerned with developing techniques that allow groups of people to make corporate decisions; essentially, voting algorithms. In the late eighteenth century, voting was becoming more popular in Western Europe as a result of the political philosophy of the Enlightenment. The shortcomings of simple majority voting when there are more than two candidates led Borda and Condorcet to propose more sophisticated algorithms. In 1770, Jean-Charles de Borda suggested the Borda Count algorithm for the French Academy of Sciences in Paris [18]. Around the same time, the Marquis de Condorcet proposed his algorithm [19], and the two algorithms have been at the center of a large corpus of research in Social Choice in the past century. See [61, 52, 37] for good introductions to the field.

An *election* is an instance of a voting problem. The input is called a *voting profile*. For example, consider the following profile of a 5 candidate, 10 voter election:

3: a, b, c, d, e
3: b, e, c, a, d
2: c, a, d, e, b
2: d, b, e, a, c

A slight variation on the way we display voting profiles in Chapter 2, in this case each row is prefaced by the number of voters that submitted the given ranking. This means that of the 120 possible ways of ranking the five candidates a, b, c, d, and e, three voters chose to

rank a first, followed by b, then c, then d, and e last. Likewise, three voters chose to rank b above e above c above a above d, and so on.

A *social choice function* is a function that maps voting profiles to a set of candidates—the winners. For the above profile, simple majority rule (or *plurality* voting) would dictate that candidates a and b tie, since each received three first place rankings, more than any other candidate.

A useful distinction is made by Riker [61]. He distinguishes *majoritarian* voting methods, those based on a series of pairwise comparisons of candidates, from *positional* methods, based on n-ary comparisons.[1] Majoritarian methods use *pairwise comparisons*: for example, in the above profile, candidate b is ranked ahead of c in eight of the ten profiles. Thus in a simple majority run-off election between these two candidates, b would receive eight of the ten votes cast. Majoritarian algorithms work by making a series of such binary comparisons; the Condorcet algorithm, for instance, specifies that the winner of the election is the candidate(s) that beats or ties with every other candidate in a binary comparison.

Positional algorithms, on the other hand, compute a score for each candidate based on the positions, or ranks, given to each candidate by the voters. Plurality voting, for example, gives one point for each time a candidate is ranked first, and zero points for any other position. The Borda count, for an m candidate election, gives $m-1$ points for being ranked first, $m-2$ points for being ranked second, and so on, down to 0 points for being ranked last. In positional algorithms, the candidate(s) that accumulates the most points wins.

To apply voting algorithms to metasearch, we consider documents as candidates and input systems as voters. We have to modify the algorithms somewhat, however, since they are only concerned with locating the top candidate, while we need to produce an entire ranking of the candidates. Because of this difference and because we are in a many-candidates, few-voters situation, some voting algorithms are not applicable. Plurality voting, for instance, is not applicable: if we are combining two input systems, it is likely that plurality voting would give two candidates a score of one, while the other thousand or so would tie with a score of zero.

In the field of Social Choice, algorithms are usually justified by considering the properties they satisfy. For instance, May's theorem shows that, in the case of a two-candidate election, "majority voting is the only method that is anonymous (equal treatment of all voters), neutral (equal treatment of the candidates), and monotonic (more support for a candidate does not jeopardize its election)" [52]. This lends support to the Condorcet algorithm, since the Condorcet winner wins (or ties in) every possible pairwise majority contest. On the other hand, Saari has recently shown that only the Borda Count satisfies all of the symmetry properties that one would expect of any reasonable election strategy [62, 63]. In this work, we take a complementary approach: we use our experimental setup to test the effectiveness of the algorithms directly. We begin with the Borda Count, and move on to the more interesting Condorcet algorithm in the next chapter.

[1] Riker actually has a third category of algorithms that he calls *utilitarian*. This is a less well-used and less well-justified group that we do not study here.

6.2 The Borda-fuse Model

6.2.1 Definition

Recall that the Borda count works as follows: each voter ranks some or all of the m candidates in order of preference. For each voter, the top-ranked candidate is given $m-1$ points, the second-ranked candidate is given $m-2$ points, and so on. If there are some candidates left unranked by the voter, they are assumed to tie for last place and the remaining points are divided evenly between the unranked candidates. The candidate with the most points wins.

This algorithm can be easily adapted to metasearch if documents are considered as candidates and input retrieval systems as voters. The document pool is the set of all documents returned by any input system. Documents are ranked by the number of points they obtain.

The Borda count has been applied to other problems somewhat similar to our own. Van Erp and Schomaker [73] use the Borda count (along with two variants) to solve the problem of combining classifiers for multi-class problems like handwriting recognition where each glyph must be classified as one symbol in the alphabet. The classifiers are identified with the voters, and the classes with the candidates. In this problem, as in voting, the desired output is usually a single class (e.g.,. a letter in the alphabet), not a ranking of the classes. In our problem, we seek the entire ranking of the candidates.

6.2.2 Related Algorithms

Note that the Borda-fuse algorithm is similar to ruCombSum. In the case that exactly the same set of documents is returned by each retrieval system, ruCombSum and Borda-fuse are equivalent. The difference lies in how documents that are not retrieved affect the scoring.

In ruCombSum, document d not returned by system S_i contributes the same amount to $s_{\text{ruCombSum}}(d)$ as the 1000th document returned by S_i: zero. In Borda-fuse on the other hand, documents not returned by S_i contribute significantly less than the 1000th document. How much less depends on how much overlap there is between the systems: if there is a lot of overlap, then the difference is less significant; if there is little overlap, the difference is greater. As we note in Section 1.2.2, the average overlap between retrieval systems (for TREC systems at least), is surprisingly small; thus the two algorithms may behave quite differently.

6.2.3 Relevance Scores

When relevance scores are available, we would like to take advantage of them. Thus we may adapt the Borda-fuse algorithm by allotting each input system an arbitrary, fixed quantity of points p to be distributed according to the relevance scores of the documents. The final score of a document can still be the the sum of the points that it accrues from each input system. But how should the relevance scores cause the points allotted to an input system to be distributed?

There are of course many different ways of doing this. Perhaps the simplest is the solution we adopt: We assign zero points to unretrieved documents, and we use the sum

normalization scheme that we defined in Chapter 4. That is, we shift the scores so that the document with the lowest score has score zero, and then scale the scores so that they sum to one. If we set $p = 1$, we can directly interpret the normalized scores as the distribution of the points allotted to the input system. Indeed, suBorda-fuse is simply suCombSum using the sum normalization. In fact, although we have presented them in reverse order for clarity, it was our search for an appropriate definition of suBorda-fuse that led in part to our discovery of the sum normalization.

6.2.4 Weighted Borda-fuse

In making social choices, it is often desirable that the voters have equal sway. Sometimes it is not, for example in the case of a share-holder decision where the share-holders may be weighted according to the amount of stock they own. In metasearch, when training data is available, the best performance will likely be achieved by unequally weighting the systems. Here we may adapt the performance weighting that we used in Chapter 3: we multiply the points given to each candidate by system i's performance weight, α_i. We call this variant weighted Borda-fuse, or wBorda-fuse.

6.3 Experimental Results

Figure 6.1 compares all of the Borda-fuse variants in terms of improvement over the best input system. The score-based variants of Borda-fuse are labeled as CombSum algorithms using the sum normalization to emphasize the fact that we presented these graphs earlier under a different name. Here, as with the four CombMNZ variants presented in Figure 3.4, using scores generally helps (see especially the best-to-worst experiment graphs). Using weights never hurts performance, and weights help when—as in the random-sets experiment—the input systems may be of different quality.

6.4 Conclusion

This chapter introduces the use of results from Social Choice Theory to the metasearch problem. While the first algorithm based on these results, Borda-fuse, has only reasonable performance, it has nevertheless been fruitful: in our research it has led to higher-performing algorithms.

It first led to the sum score normalization presented in Chapter 4, which, together with swCombMNZ, has yielded one of the highest performing algorithms. Even the ZMUV normalization is indirectly indebted to Borda-fuse for inspiring the study of normalization itself.

Secondly, our study of Borda-fuse led to the invention of Condorcet-fuse, a better-performing algorithm that we present in the next chapter.

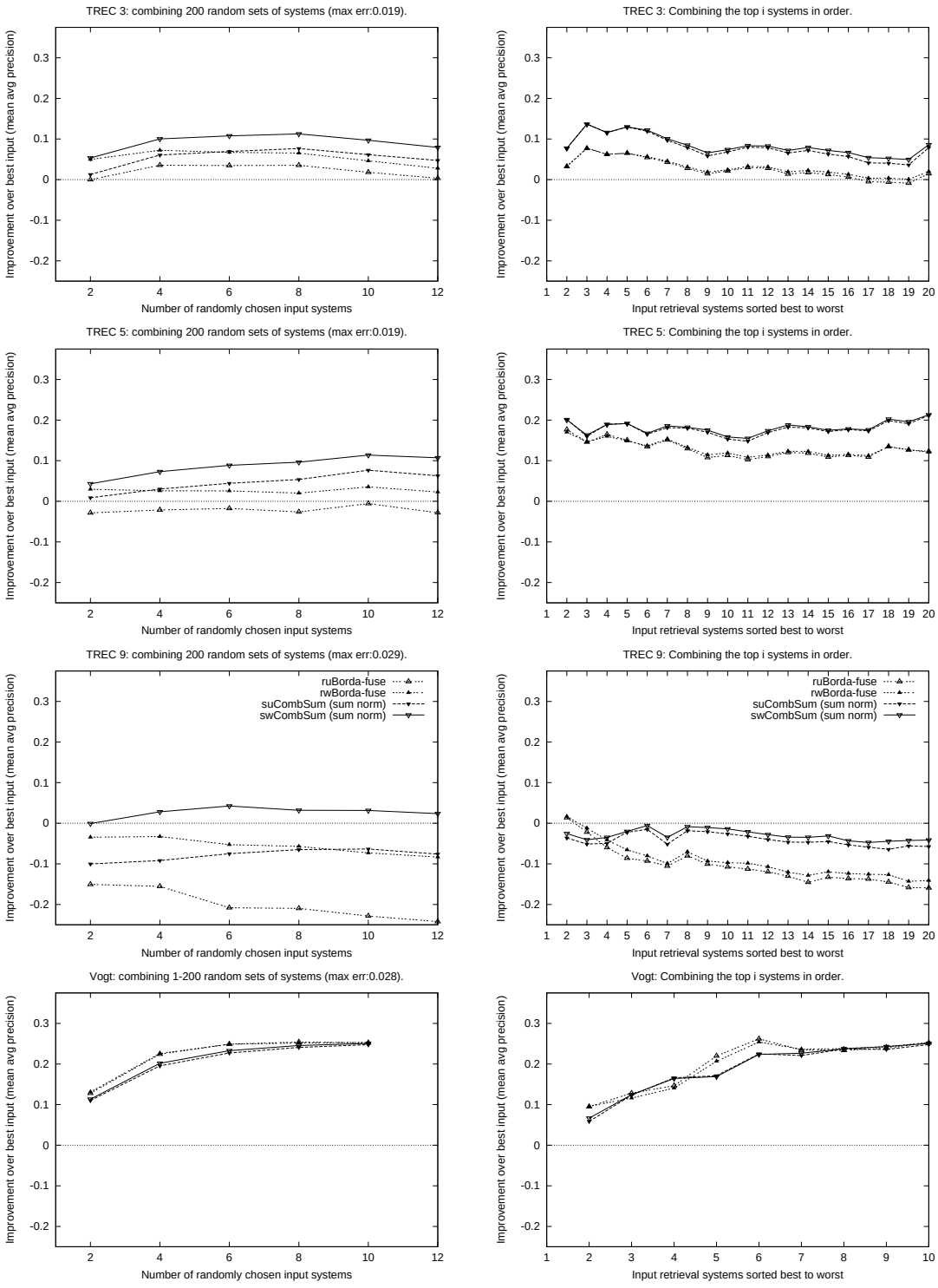

Figure 6.1: The improvement of all four Borda-fuse variants. Note that score-based Borda-fuse variants are labeled as CombSum algorithms using the sum normalization.

Chapter 7

Condorcet-fuse

In the previous chapter, we apply a positional voting method to the metasearch problem: the Borda Count. In this chapter we apply a majoritarian voting method, the Condorcet algorithm. In short, the Condorcet algorithm says that any candidate that can beat all other candidates in a head-to-head contest should win the election. We generalize this notion to generate a ranked list of candidates.

7.1 The Voting Model

We model an election with a directed graph of candidates which we call the *Condorcet graph*.

7.1.1 The Condorcet Graph

Given a voting profile for an election, consider the directed graph containing one node for each of the m candidates. For each candidate pair (x, y), there exists an edge from x to y (which we denote $x \to y$) if x would receive at least as many votes as y in a head-to-head contest. That is, $x \to y$ if x is ranked above y by at least as many voters as ranked y above x. We call this the *Condorcet graph* of the given profile. A five-candidate example is shown in Figure 7.1.

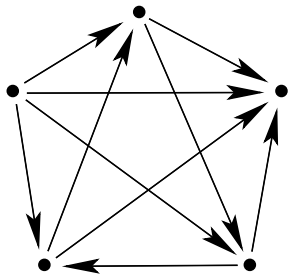

Figure 7.1: A five candidate Condorcet graph.

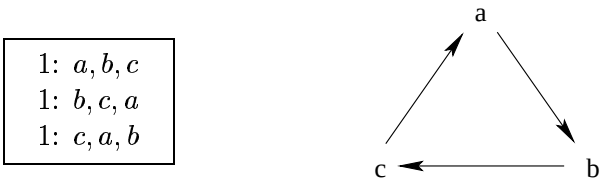

Figure 7.2: The "voting paradox:" a simple case of cyclic preferences.

Note that in the Condorcet graph of any profile there is at least one edge between every pair of candidates. Between those that directly tie (when the same number of voters rank x above y as rank y above x), there is an edge pointing in each direction (denoted $x \leftrightarrow y$). We call such graphs (where there is at least one directed edge between any two nodes) *semi-complete*.

7.1.2 The Voting Paradox

Any candidate that beats or ties with all others is called a *Condorcet winner*. In the Condorcet graph, this corresponds to having an out-degree of $m-1$; that is, an edge pointing to all other candidates. Interestingly, for some voting profiles there are no Condorcet winners. This is called the *Voting Paradox*, and has been the cause of great controversy in the field of Social Choice. The simplest "paradoxical" profile has three voters; see Figure 7.2. In this profile, a beats b twice, b beats c twice, and c beats a twice. Thus we obtain the Condorcet graph shown in the figure. We say that this group of voters exhibits *cyclic preferences*. Note that cycles may occur in the Condorcet graph even if there exists a Condorcet winner; there could have been a candidate d in the above example that each voter ranked first.

The possibility of cyclic preferences has been considered a paradox because it has seemed to many that combining a set of acyclic preferences should not yield a cyclic result. One of the most important results in the field of social choice, however, is Arrow's Theorem, which states that under a certain set of reasonable conditions, any majoritarian voting algorithm will sometimes produce a cyclic set of preferences. In light of this, we prefer to take the perspective that a cycle in the Condorcet graph is simply a special kind of tie. In fact, given the way we define the Condorcet graph, normal two-candidate ties appear as cycles. Consider the profile:

1: a, b
1: b, a

By our definitions, this too yields a cyclic Condorcet graph.[1]

[1]Note that we could have arrived at more standard definitions had we chosen to put *no edges* between candidates that tie in a pairwise runoff. In this case, Condorcet winners would be those candidates with in-degree zero, and the above example would not be a case of cyclic preferences. Instead we chose to put edges in both directions in the case of direct ties. Besides the simplicity of having all ties correspond to cycles, this allows us to develop our algorithms based on paths through the graph and strongly connected components.

From a philosophical perspective, we actually view the possibility of cycles in the Condorcet graph as a virtue of the technique, since we feel that it captures the true complexity of the voting profile. We view the cycles as real and not simply an artifact of the voting procedure. Other algorithms (like the Borda Count and most positional methods) may generate fewer ties, but only at the cost of ignoring some of the structure of the problem[2].

7.1.3 Strongly Connected Components

Here we review the definition and properties of the strongly connected component graph of a graph. The idea is that we can form a new acyclic graph from an old cyclic one by contracting all of the nodes in a cycle into one. More formally, the strongly connected components (SCC's) of a graph partition the nodes into equivalence classes: two nodes are equivalent if there exists a cycle in the graph containing them. Thus each SCC of the Condorcet graph contains a set of nodes that tie. In the strongly connected component graph (SCCG), each node represents one SCC, and for nodes X, Y in the SCCG, $X \to Y$ if there exists an $x \in X$ and $y \in Y$ such that $x \to y$. Moreover, in semi-complete graphs, $x_i \to y_j$ for all $x_i \in X$ and $y_j \in Y$. SCCG's are acyclic.

The strongly connected components of a semi-complete graph can be unambiguously sorted. Since the original graph was semi-complete, so is the SCCG. Since the SCCG is acyclic, there is one node X_0 with in-degree zero. If it is removed from the SCCG, another semi-complete, acyclic graph remains. Thus it has one node X_1 with in-degree zero. This process can be repeated until all k nodes have been exhausted. The unique ordering is $X_0 \to X_1 \to \ldots \to X_{k-1}$.

Though we do not need to compute this SCC graph in our work, it is a useful way to think about the structure of the Condorcet graph. The graph is totally orderable at the level of the SCC's, and each SCC is a "pocket" of cycles, within which each candidate is tied. The strongly connected components partition the graph into sets of tied candidates. Thus we can generalize the concept of Condorcet winner(s) to include all candidates that tie for first place: those in the top SCC.

Fortunately for us, ties are less of a concern in metasearch than in Social Choice. Since we are ranking thousands of candidates, if a relatively small number of candidates are ordered incorrectly in the final ranking because we chose a poor way to break the tie, it does not change the evaluation score greatly. In fact, if they all are relevant or all are irrelevant, it does not change the score at all. And, unless they are all to be ranked among the top documents, any way of breaking the tie does not affect the score greatly.

7.1.4 Condorcet Paths

The Condorcet voting algorithm reports only the Condorcet winners. For metasearch, however, we want to generate a ranked list of all the candidates, not just those that tie for first place. Thus we define a *Condorcet-consistent Hamiltonian path* (or just *Condorcet*

[2]In response to the voting paradox, Social Choice researchers do not usually consider the Condorcet algorithm complete without a cycle-breaking rule. To them, the Condorcet technique encompasses a family of algorithms, *Condorcet extensions*, each of which breaks cycles differently. Dartmouth's own John Kemeny defined one of the most interesting Condorcet extensions; it minimizes a distance function that he defines between ranked lists (see [38]).

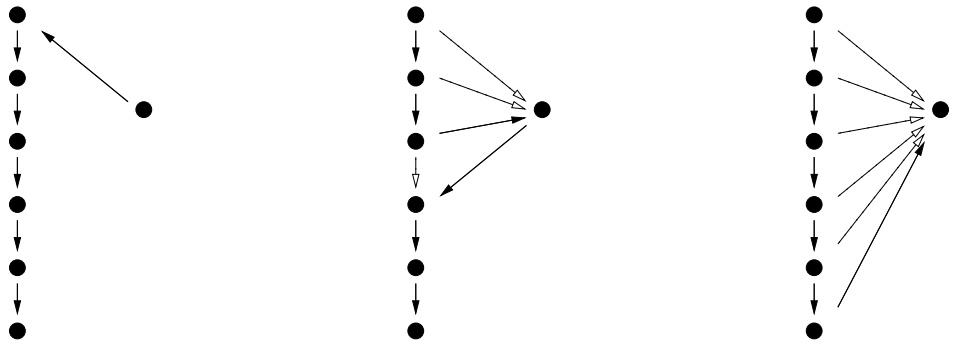

Figure 7.3: The three cases for the proof of Theorem 7.1.1.

path) to be any Hamiltonian path through the Condorcet graph. Our goal is to efficiently find such a path. Two important points must be made about these paths.

Theorem 7.1.1. *Every semi-complete graph contains a Hamiltonian path.*

Proof. We prove Theorem 7.1.1 by induction. The base case, a graph with one node, is trivial.

For the inductive step we assume that every semi-complete graph with $n-1$ nodes contains a Hamiltonian path. In particular, for a problem of size n, consider the Hamiltonian path H for the subproblem containing only an arbitrary $n-1$ of the nodes. Now the nth node x is introduced, along with its $n-1$ edges to the other nodes. There are three cases (see Figure 7.3): (1) If x points to the first node in H, then x followed by H is a Hamiltonian path. (2) If not, then the first node in H points to x. Now consider each node in H in turn. A new Hamiltonian path can be created by inserting x into H just before the first node that x points to, if one exists. (3) If x does not point to any of the nodes in H, then the last node in H points to x, so a new Hamiltonian path can be created by appending x to H. □

The next theorem shows that any Condorcet path "properly" orders the SCC's.

Theorem 7.1.2. *Suppose x and y are nodes in a graph g, and that X and Y are nodes of the associated SCCG G such that $x \in X$ and $y \in Y$. If there exists a path from X to Y in G, then every Condorcet path of g has x before y.*

Proof. This is a simple consequence of the fact that there is only one way to sort the SCCG; thus there does not exist a path from y to x. So any Hamiltonian path puts x before y. □

Because every path properly orders the SCC's, a randomly chosen Condorcet path correctly orders the candidates that do not tie, and breaks ties arbitrarily.

7.1.5 Efficiency

We want to efficiently find a ranking that orders nodes in different SCC's correctly. We are not concerned with the ordering of nodes in the same SCC, since they have tied. Thus, we could generate the entire Condorcet graph, compute SCC's, sort the SCC's, and order candidates within each SCC arbitrarily. Note that this would not yield a Condorcet path, but the important thing is that it would order the SCC's properly. This algorithm is $O(m^2n)$ for an election with m candidates and n voters, since it takes time $O(m^2n)$ just to generate the semi-complete Condorcet graph. Happily, however, there is an $O(mn \lg m)$ algorithm based on finding a Condorcet path that need not ever generate the entire Condorcet graph.

The algorithm is suggested by the proof that a Hamiltonian path exists in the Condorcet graph. The inductive step of the proof strongly resembles insertion sort, where incoming items are compared to all previous items until their proper location is found. Indeed, insertion sort can be used to find a Condorcet path; all that needs to be specified is the comparison function that the sorting algorithm will use. In fact, a similar proof can be used to show that mergesort will also find a Condorcet path; thus we can find a path using only $O(m \lg m)$ comparisons.

How can we efficiently compare two candidates? That is, how can we quickly have a run-off election between them? The answer is to represent the election so that instead of having n ranked lists of up to m candidates, we have m lists of the n ranks given to each candidate in some canonical order. This is a simple preprocessing step of time $O(mn)$. Then a binary comparison can be made between any two candidates in time $O(n)$, yielding total time $O(mn \lg m)$.

One note of interest is that quicksort will also properly order the SCC's. All comparisons made between nodes in different SCC's are answered consistently since they do not involve any cycles in the graph. Comparisons made between nodes in the same SCC are meaningless since the nodes are in a cycle. Thus, within each SCC, quicksort will arbitrarily order the nodes, perhaps in a way not consistent with the Condorcet graph. Since we are not concerned with the ordering within each SCC this does not present a problem, and in our experimental experience, using quicksort yields ranked lists of the same quality as using mergesort. This is to be expected, since they both properly order SCC's.

7.1.6 Variants

One advantage of using a sorting routine together with a black-box comparison function is that we can easily replace the comparison function to implement a new algorithm. We can implement a weighted version of Condorcet-fuse by using a weighted binary comparison.

In a simple majority, two-candidate election, candidate x beats y if more voters choose x than choose y. We can generalize this to the case that different voters have different weights by saying x beats y if the sum of the weights of those that voted for x is larger than the sum of the weights of those that voted for y. Using this comparison function in the sorting algorithm yields weighted Condorcet-fuse.

Interestingly, the most obvious way to incorporate relevance score information into a comparison function for Condorcet-fuse replicates CombSum. Consider the comparison function that says x beats y if the sum of scores given x is larger than the sum of scores given y. This is the rule that CombSum uses. We have not yet found an interesting way

for Condorcet-fuse to use relevance scores. Indeed, the use of relevance scores violates the majoritarian spirit of the algorithm, which only ever asks, "who is preferred, x or y?" It never asks, "by how much?"

CombSum is not the only metasearch algorithm that Condorcet-fuse can mimic. To mimic CombMax, for instance, we say x beats y if the maximum score given x is greater than the maximum score given y. Thus a large class of metasearch algorithms can be conveniently implemented as a simple, black-box comparison routine passed to a quicksort algorithm.

Finally, note that Condorcet-fuse itself generates no relevance scores. It ranks the documents by direct sorting, without first assigning a relevance score to each document.

7.2 Experimental Results

Note that all of the experiments we describe using Condorcet-fuse were implemented using quicksort, not mergesort. We have found that the two perform identically in practice.

Figure 7.4 shows experimental results for Condorcet-fuse in terms of improvement over the best input system. ruCondorcet-fuse is the only algorithm that, without training data, ever matches the performance of the best input system over TREC 9.

In the best-to-worst experiment, an interesting anomaly occurs: input system dependence becomes an obvious performance issue. Consider the TREC 9 data set. Notice the dip in performance that Condorcet-fuse suffers when adding input systems three, four, and five. The names of the first five input systems in TREC 9, (along with their mean average precisions) are shown in Table 7.1. Systems two, three, and four are runs by the

TREC Run	Avg Prec
iit00m	0.3519
jscbt9wll2	0.2801
jscbt9wcl1	0.2687
jscbt9wll1	0.2659
ric9dpn	0.2616

Table 7.1: The top five input systems from the TREC 9 data set. Systems 2, 3, and 4 are runs from the same research team.

Justsystem Corporation, and are actually quite similar to each other. A simple method to measure the similarity of these ranked lists is to consider the ranked lists for each query simply as unordered sets of documents, and define the similarity of two sets A and B as $sim(A, B) = \frac{A \cap B}{A \cup B}$. This measure runs from zero for disjoint sets to one for identical sets. We use $sim()$ as a measure of similarity between two input systems by averaging its value over the 50 queries. This is a rough but reasonable measure of the similarity of two input systems—if they consistently return the same set of top 1000 documents, they *probably* return similar sets of top 100 documents, and similar sets of top 10 documents, etc.

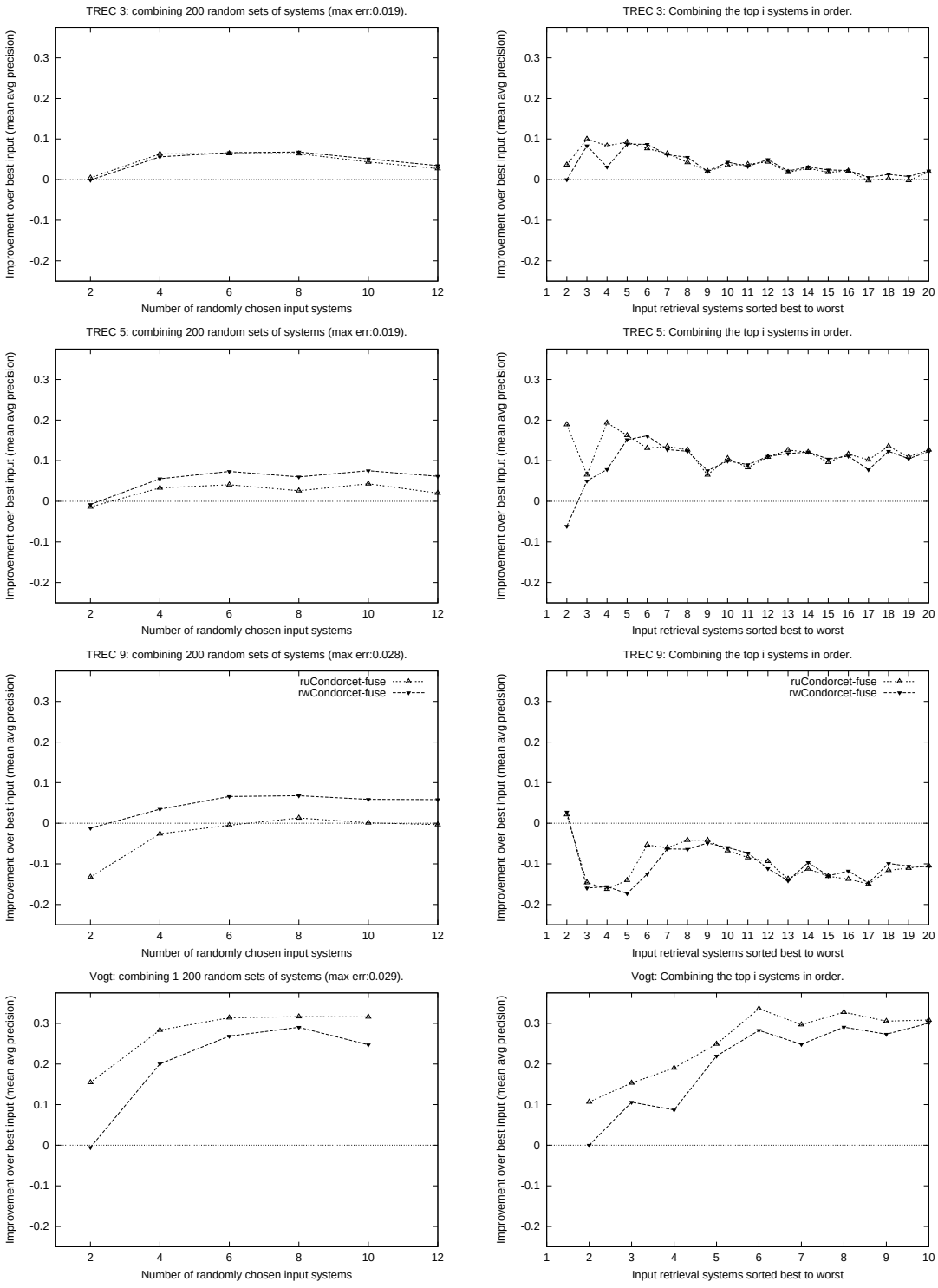

Figure 7.4: The improvement of Condorcet-fuse. Weighted and unweighted versions are shown.

If we compare TREC runs using our set-similarity measure, we find that the average similarity of any two systems in each data set is as shown in Table 7.2. Note that it is not

Data Set	Avg Sim
TREC 3	0.229
TREC 5	0.167
TREC 9	0.162
Vogt	0.149

Table 7.2: The average pairwise set-similarity of runs in each data set.

surprising that the Vogt data set has the lowest average similarity, since this data set was chosen for its diversity. But, the average similarity between the three Justsystem Corp. runs listed above is 0.789—the runs are very similar.

Thus, on the TREC 9 data set, when Condorcet-fuse combines the top two systems, it performs well (refer back to Figure 7.4). But when it combines the top three systems, suddenly the two Justsystem Corp. runs dominate over the (much better performing) IIT run—they have a two versus one majority when voting on each pair of documents. This is repeated when combining the top four systems, and it is not until combining the top six systems that the three Justsystem runs do not have a majority voice—at this point the performance of Condorcet-fuse returns to a reasonable level. Thus the effect of dependent input systems makes its first obvious appearance in our work. Condorcet-fuse seems particularly sensitive to it when combining only a few systems. Note that the same effect can be seen in the TREC 5 data set, where input systems two and three are again two runs from the same retrieval team, and are highly similar (in this case, their average set-similarity is 0.856).

Dependence Filtering

As we pointed out in Chapter 5, one way to address the problem of input dependence is to try to eliminate it directly. We experiment with perhaps the most simple way of doing this, which we call *dependence filtering*. Dependence filtering can be used with any metasearch algorithm A: before A runs on a set of input systems $\mathcal{S}$, filter the dependent systems out of $\mathcal{S}$. That is, examine each pair of systems $S_1, S_2 \in \mathcal{S}$, in order of descending set-similarity. If the average set-similarity of S_1 and S_2 is above some threshold (we use 0.66 for our experiments), randomly drop one of them from $\mathcal{S}$, resulting in a smaller set of input systems $\mathcal{S}'$. Then proceed to fuse $\mathcal{S}'$ using A.

If $|\mathcal{S}| = n$, our brute-force implementation of dependence filtering performs $O(n^2)$ similarity comparisons between input systems, and each similarity comparison takes $O(mq)$ time for q queries each with m documents. However, the actual similarity comparisons need only be done once per data set. In other words, we pre-compute the similarity between every pair of input systems in each data set, and, when fusing, we need only look up the $O(n^2)$ values. This has not been prohibitively expensive for us since we never fuse more than $n \approx 100$ input systems at once.

Also note that dependence filtering requires no training data in the form of relevance judgments; the similarity assessments are made based on the ranked lists alone. To get an average similarity, dependence filtering does require several ranked lists from each system in response to the same set of queries, but this is generally easy to obtain.

We present the results of using dependence filtering with ruCondorcet-fuse in Figure 7.5 in terms of improvement over the best input system. The dependence-filtered Condorcet-fuse is labeled "druCondorcet-fuse." The results are more or less unchanged over the TREC 3 and Vogt data sets, where we had not observed any performance dips due to dependence (the slight variation that is observable is due to the randomness of the quick-sort algorithm—when two documents directly tie or are in a cycle, the tie can be broken arbitrarily). On the TREC 5 and TREC 9 data sets, however, the large performance dips due to input system dependence are replaced by areas of constant performance, where no new systems are added to the fusion.

7.3 Conclusion

In this chapter we propose a new perspective on the Condorcet voting technique based on modeling a voting profile with a *Condorcet graph*, in which cycles represent a generalized notion of ties between candidates. Based on this model, we define a *Condorcet-consistent Hamiltonian path*. Its key property is that it correctly orders nodes that do not share a cycle. Perhaps most importantly, we propose a sorting-based algorithm for finding such a path in near-linear time. This represents an advance in the *computational* side of Social Choice Theory itself.

With regards to the *experimental* side of Social Choice Theory, we show that the family of Condorcet voting algorithms is superior to the Borda Count over the TREC data sets. Our sorting-based approach breaks cycles in the Condorcet graph arbitrarily; one would expect a more principled approach, like Kemeny's extension, to perform at least as well. But even when breaking ties arbitrarily Condorcet-voting outperforms the Borda count.

In this chapter we also detect the first performance decrease directly attributable to the dependence of the input systems, and we propose the *dependence filtering* algorithm which effectively solves the problem. Dependence filtering is applicable to any metasearch algorithm, and is a step toward fully addressing the problem of dependence in metasearch—the first in the metasearch literature.

A final advantage of the Condorcet-fuse algorithm is that it has directly led to new upper bounds for the metasearch problem, which we address in the next chapter.

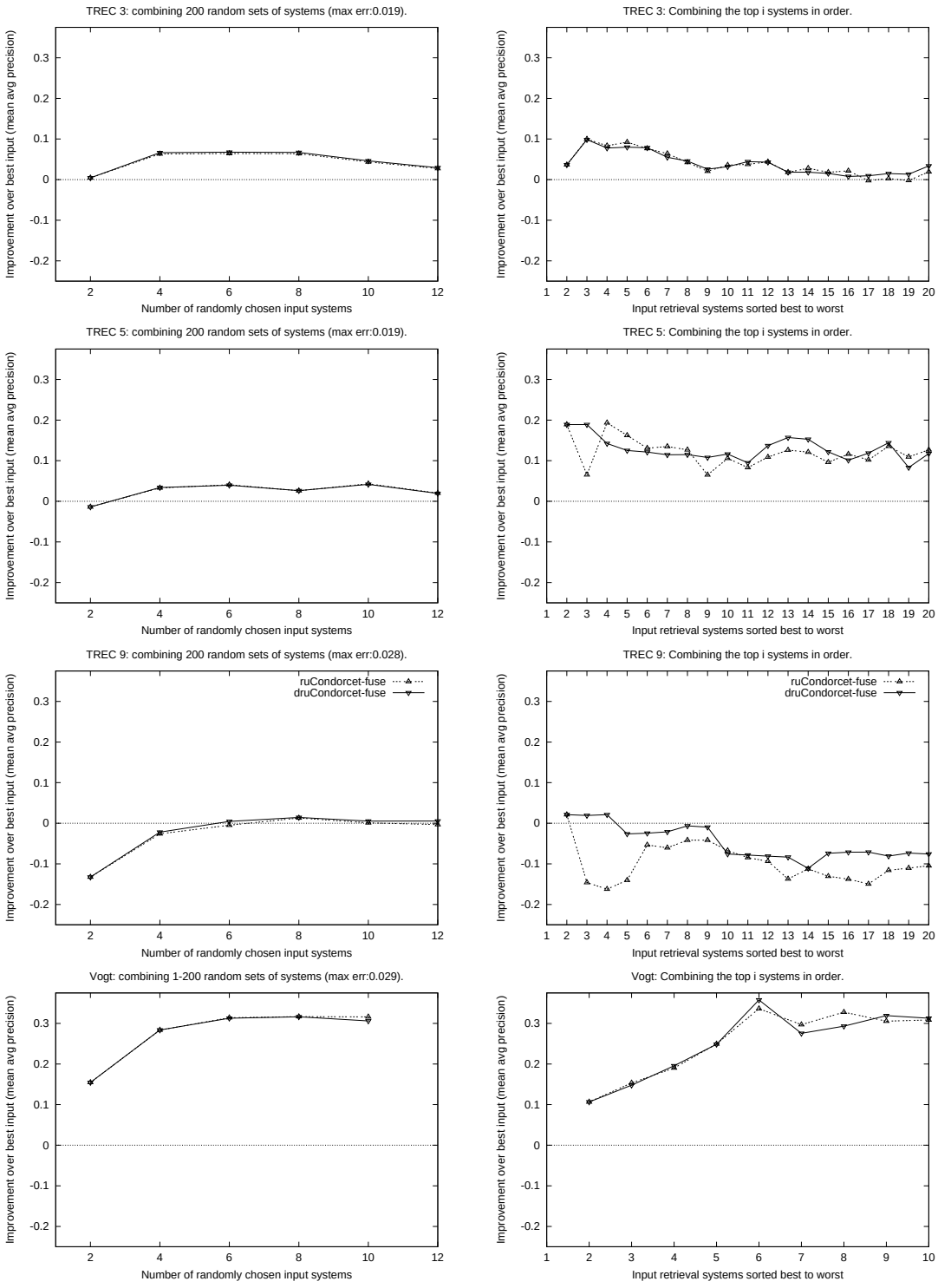

Figure 7.5: Dependence filtering. druCondorcet-fuse is not susceptible to the dependence problems that ruCondorcet-fuse is.

Chapter 8

Upper Bounds

Previous chapters present techniques for improving metasearch, and evaluate the results by using standard performance measures to compare each metasearch algorithm with the best input system. This chapter introduces another evaluation technique: it compares metasearch performance against theoretical upper bounds. That is, we know how well metasearch *is* performing; how well *might* it perform? After all, there is only so much information in the input systems. Can we hope for improved algorithms in the future, or are we nearing the best performance possible?

8.1 Constrained Oracle Model

Assume we have an omniscient oracle that is willing to act as a metasearch engine. Like other metasearch engines, the oracle takes ranked lists as input and must produce a single ranked list as output. Unlike other metasearch engines, the oracle knows which documents are relevant. Without any constraints, the oracle's job is easy: for each fusion problem, the oracle simply ignores its input ranked lists and outputs the relevant documents for the given query (ranked arbitrarily)—it scores perfectly every time. To obtain a meaningful upper bound on the performance of achievable metasearch strategies, we may *constrain* the oracle in ways consistent with natural limitations of conceivable metasearch algorithms.

We present three different constraints that yield upper bounds for three different classes of metasearch algorithms. The first constraint is the least restrictive.

8.1.1 Naive Bound

Our first natural constraint is that the oracle may *only output documents that appear in one of the input ranked lists.* Although the oracle can return all of the relevant documents in the input and none of the irrelevant, it no longer scores perfectly since it is not allowed to include *all* relevant documents for each query; it is dependent on its input. We call the performance of this oracle the *Naive bound.*

The Naive bound applies to all metasearch algorithms.

8.1.2 Pareto Bound

We can further constrain the oracle in a natural way by insisting that it effectively heed the unanimous advice of its inputs. In Social Choice Theory, one of the fundamental desirable properties of a voting procedure is *Pareto optimality:* "if a candidate a is unanimously preferred to candidate b, then b should not be elected" [52]. We may generalize this property for the case where we are interested in *ranking* the candidates instead of electing a single candidate. In particular, we define Pareto optimality thus: *if candidate a is unanimously preferred to candidate b, then a must be preferred to b in the output ranked list.* We call the performance of an oracle constrained by Pareto optimality the *Pareto bound.*

The Pareto bound applies to all Pareto-optimal metasearch algorithms, which includes all of the Borda-fuse and Condorcet-fuse variants but does not include the Bayes-fuse algorithm. Bayes-fuse learns odds for documents with different ranks (or scores) independently; thus it is possible that for input system S, documents ranked poorly tend to be more relevant than documents ranked well. In this case, Bayes-fuse would give higher odds to worse-ranked documents, perhaps violating Pareto optimality, for better or worse.

8.1.3 Majoritarian Bound

Going a step further than Pareto optimality, we define *Majoritarian optimality* thus: *if candidate a is preferred to candidate b by a majority of the voters, then a must be preferred to b in the output ranked list.* Recall that this is the driving intuition behind the Condorcet voting algorithm. We call the performance of an oracle constrained by Majoritarian optimality the *Majoritarian bound.*

The Majoritarian bound applies to all Majoritarian-optimal metasearch algorithms, which includes unweighted Condorcet-fuse but does not include weighted Condorcet-fuse. Weighted Condorcet-fuse has the ability to override a numerical majority when comparing two candidates, since it considers the weights of the input systems voting for and against each candidate. Nor does the Majoritarian bound apply to the dependence-filtered, unweighted Condorcet-fuse algorithm—we insist that the bound listen to the majority opinion of *all* its inputs; dependence filtering is precisely the decision to ignore some of the inputs—the ones that sound redundant.

The Majoritarian bound does not apply to the Bayes-fuse algorithm (for the same reasons the Pareto bound does not apply), nor the Borda-fuse algorithm. The Borda Count is a positional voting method, not a majoritarian one. Thus if candidate a is ranked better than b by a majority of input systems, b may still be placed better than a if, for instance, b's average rank is better than a's.

8.1.4 Constraint Graph

We use a *constraint graph* to model the application of a constraint rule (like Majoritarian optimality) to a given profile. In a constraint graph, the nodes represent candidates, and the edges are constraints. If node a points to b, then a is constrained to be ranked better than b. Thus the addition of an edge to a constraint graph can only reduce the size of the set of feasible ranked lists (those that satisfy all of the constraints). For this reason, adding

constraints can only decrease (or leave unaffected) the performance of the best feasible solution, and removing constraints can only increase (or leave unaffected) performance.

8.1.5 Comparing the Bounds

The three bounds form a clear hierarchy: for any input profile, the Naive bound is greater than or equal to the Pareto bound, which is greater than or equal to the Majoritarian bound. Consider their respective constraint graphs: the Naive bound constraint rule puts no edges in the constraint graph. The Pareto optimality constraint may put some edges in the graph. The Majoritarian optimality constraint keeps all of the Pareto optimality constraint edges, since if a unanimously beats b, it certainly beats b with a majority. And the Majoritarian optimality constraint may add some new edges, further reducing the performance of the oracle.

8.2 Implementation

Implementing the oracle for the Naive bound is straightforward since it simply ranks all relevant retrieved documents first. In general, however, simulating a constrained oracle is challenging. Given a constraint graph for a profile, the optimal feasible ordering is highly sensitive to the exact performance measure that is being used. We do not have an algorithm for optimizing the ranking with respect to average precision. More fundamentally, however, computing the constraint graph itself is computationally infeasible for the size and quantity of experiments we perform. Therefore we approximate the Pareto and Majoritarian bounds.

Both can be approximated with algorithms similar to Condorcet-fuse. We use quicksort as the basic sorting function, and use a specialized comparison function for each bound. For the Pareto bound, we use the following comparison function (say we are comparing candidates a and b): if all of the voters rank a better than b, a wins. If not, if all rank b better than a, b wins. If not, if a is relevant, a wins. Otherwise, b wins.

With this comparison function, relevant documents are ranked as well as possible unless they are constrained by unanimous opposition. Still, it may not be the optimal feasible ranking. Thus it is a lower bound on the Pareto bound. As the experimental results show, however, it is still quite a high-performing algorithm compared to actual, non-oracle algorithms, and thus it serves as a reasonable approximation of the Pareto bound.

For the Majoritarian bound, we use the following comparison function: if a majority of the voters rank a better than b, a wins. If not, if a majority rank b better than a, b wins. If not, if a is relevant, a wins. Otherwise, b wins.

Note that the only difference between the Majoritarian bound and Condorcet-fuse occurs when a and b directly tie—when the same number of voters prefer each.

Like the Pareto bound approximation, the Majoritarian bound approximation does produce a feasible, though not necessarily optimal, solution. Thus it is a lower bound. We do not know how closely it approximates the true Majoritarian bound, but we do know that as the number of ties decreases (this happens as the number of input systems increases), the number of mistakes made by our comparison function decreases. This is simply due to the fact that only in the case of a tie can the comparison function choose wrongly; otherwise it

is simply enforcing a constraint. We conclude that it is a reasonable approximation when combining more than a few input systems.

8.3 Experimental Results

The experimental results are shown in Figure 8.1. Note first that the y-range of these graphs is $[0, 1]$, unlike all the previous graphs in this work. In all of the experiments, the Naive bound quickly approaches perfect retrieval. The Pareto bound is significantly lower, but still quite high relative to any actual metasearch algorithm (represented in the graph by unweighted Condorcet-fuse). The Majoritarian bound, on the other hand, starts out identical to the Pareto bound—the two are always equal when combining two input systems; in this case majority implies unanimity. But, it actually declines as the number of input systems increases. This decrease in performance is due to the decrease in the number of pairs of documents that tie as the number of input systems increases. It is only when an "election" between two documents results in a tie that the Majoritarian bound approximation inspects whether or not the document is relevant; if no two-way ties occur between candidates, the Majoritarian bound and Condorcet-fuse are identical. Indeed, in the best-to-worst experiment, the plot has a jagged appearance because ties are more likely when an even number of systems are being combined (note that in the random-sets experiment, we only perform tests with an even number of input systems). When an odd number of systems are being combined, the number of ties is so few that the Majoritarian bound approximation and the Condorcet-fuse algorithm are almost identical. This is reasonably strong evidence that Condorcet-fuse is close to being an optimal majoritarian algorithm. There may, of course, be non-majoritarian algorithms that are better—though it does not appear in this graph, druCondorcet-fuse actually outperforms the Majoritarian bound approximation in the performance dip on the TREC 9 data set in the best-to-worst experiment (refer to Figure 7.5).

8.4 Conclusion

This chapter presents models and experiments for three different constrained-oracle upper bounds for the metasearch problem. Perhaps most interesting is the Majoritarian bound. Though it is an upper bound for the smallest class of metasearch algorithms out of the three classes considered here, it is an upper bound for one of our best performing algorithms, unweighted Condorcet-fuse. When combining a sufficiently large number of input systems, the Condorcet-fuse and the bound approach each other. There may be better algorithms outside the class of Majoritarian optimal algorithms (dependence-filtered Condorcet-fuse, for example), but this convergence toward an upper bound is substantial evidence that the quality of our metasearch algorithms is approaching a natural limit.

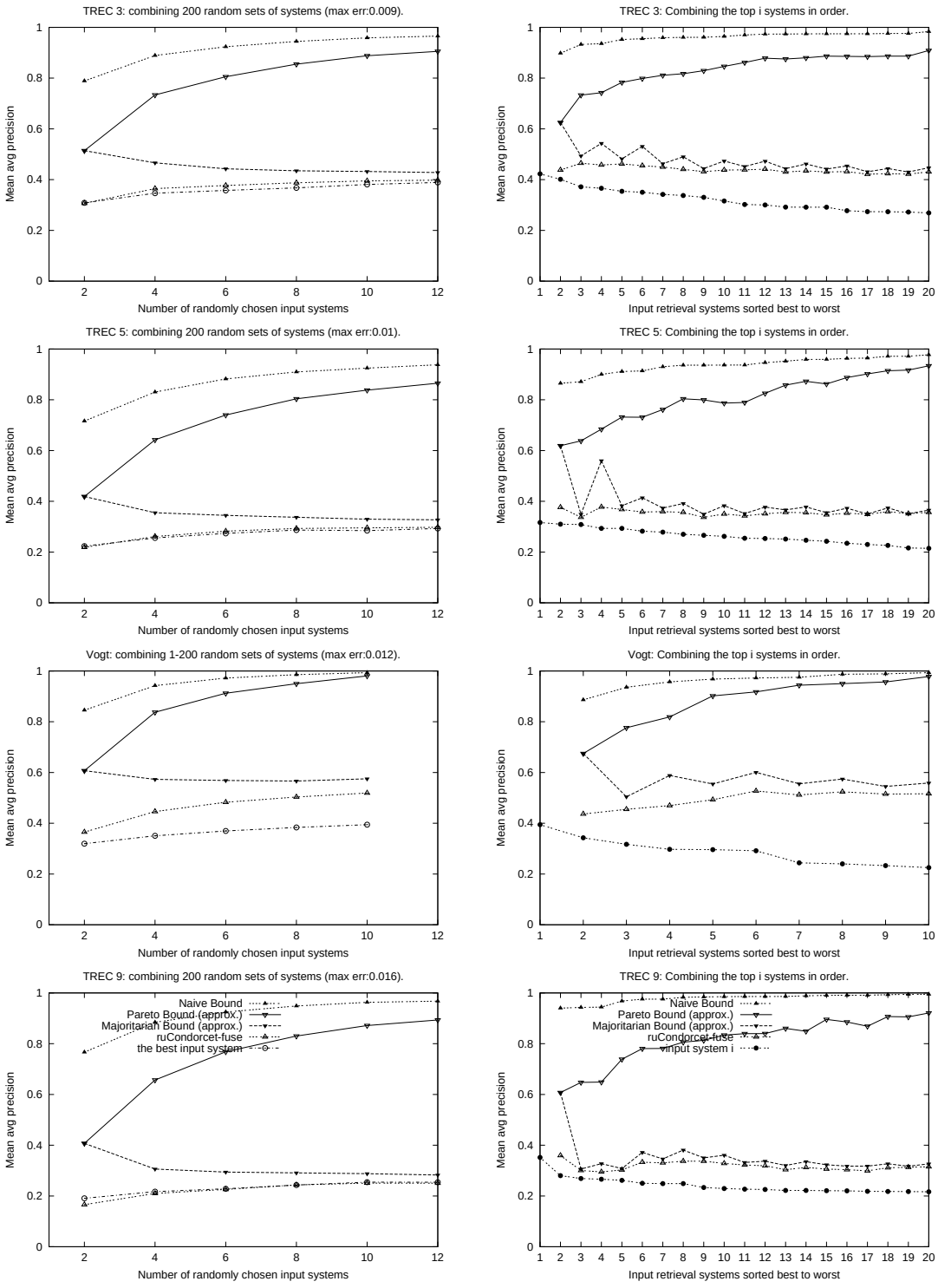

Figure 8.1: Upper bounds. The only actual metasearch algorithm shown is ruCondorcet-fuse.

Chapter 9

Conclusion

This chapter compares all of the metasearch algorithms presented in this dissertation, and summarizes the contributions of this body of work to the field of Information Retrieval.

9.1 Metasearch Comparison

Previous chapters present several metasearch algorithms and study the performance of each in isolation; here we compare them.

9.1.1 Average Improvement

We use a measure called *average improvement*, calculated as follows. Fix an experiment type e—either random-sets or best-to-worst. Let k be a fixed number of input systems being combined in experiment e. Let M be a metasearch algorithm, and ds be a data set. Then we define a *data point*, $dp(M, e, ds, k)$ as the improvement (percent improvement over the best input system) achieved by M when combining k systems in experiment e over data set ds, averaged over all such trials performed. For example, when e is the random-sets experiment, we usually perform 200 trials for fixed M, ds, k. When e is the best-to-worst experiment, we perform only one trial. Note that all of the percent-improvement graphs we present plot these data points.

Then, the *average improvement* of algorithm M in experiment e, denoted $imp(M, e)$, is simply the set of data points $dp(M, e, ds, k)$, averaged over all k used in each data set, then averaged over all data sets. More formally,

$$imp(M, e) = \frac{1}{|DS|} \sum_{ds \in DS} (\frac{1}{|K_{e,ds}|} \sum_{k \in K_{e,ds}} dp(M, e, ds, k)),$$

where DS is the set of data sets used (for us: TREC 3, TREC 5, TREC 9, and Vogt) and $K_{e,ds}$ is the set of k used in experiment e over data set ds. For example, for us, $K_{e,ds} = \{2, 4, 6, 8, 10\}$ when e is random-sets and ds is the Vogt data set.

Average improvement is a simple aggregate statistic that summarizes a lot of information about the performance of a metasearch algorithm in one number. We feel that the differences between the two experiments are great enough to warrant treating them separately, but we

	Average Improvement (%)		
Metasearch Algorithm	Rand-Sets	Best-Worst	Avg of Both
druCondorcet-fuse	8.13	9.29	8.71
ruCondorcet-fuse	8.02	7.85	7.94
ruCombSum	4.53	9.42	6.98
ruCombMNZ	2.25	8.14	5.19
ruBorda-fuse	0.56	6.24	3.40

Table 9.1: Summary of metasearch variants that apply when neither scores nor training data are available.

also average these two to get an overall ranking of the systems. All three numbers are shown in the tables below.

9.1.2 No Scores, No Training Data

Table 9.1 shows the average improvements for each metasearch algorithm that can be used when neither relevance scores nor training data are available. The algorithms are sorted from best to worst according to the "Avg of Both" column—overall performance. Generally, this agrees with the ordering of the "Rand-Sets" column, though there are several exceptions.

All of the algorithms yield some improvement over the best input system, with (dependence-filtered) druCondorcet-fuse performing best. Note that dependence filtering improves its performance in both experiments, but especially in the best-to-worst experiment, where we observed obvious dependence problems.

9.1.3 No Scores, Training Data

Table 9.2 shows the average improvements for those algorithms that can be used when relevance scores are not available, but training data is. Note that this includes algorithms that do not take advantage of the training data.

The benefit of performance weighting can be seen by comparing the CombSum variants here. Unweighted Condorcet-fuse again ranks well. And, the utility of the dependence-filtering technique is confirmed on Bayes-fuse.

9.1.4 Scores, No Training Data

Table 9.3 presents metasearch algorithms for when scores are available but training data is not. In this case, careful relevance score normalization is a key factor in achieving good performance; four of the top six algorithms use either the sum or ZMUV norms. The Condorcet-fuse variants place well for the third time, despite the fact that they ignore the relevance scores. Interestingly, dependence-filtering hinders CombMNZ.

9.1.5 Scores, Training Data

Table 9.4 shows the final input class, when both scores and training data are available. The

	Average Improvement (%)		
Metasearch Algorithm	Rand-Sets	Best-Worst	Avg of Both
rwCombSum	10.23	9.94	10.09
druCondorcet-fuse	8.13	9.29	8.71
ruCondorcet-fuse	8.02	7.85	7.94
rwCondorcet-fuse	8.62	5.86	7.24
rwCombMNZ	5.72	8.53	7.13
drwBayes-fuse	5.54	8.50	7.02
ruCombSum	4.53	9.42	6.98
rwBayes-fuse	5.64	8.06	6.85
rwBorda-fuse	6.20	6.63	6.42
ruCombMNZ	2.25	8.14	5.19
ruBorda-fuse	0.56	6.24	3.40

Table 9.2: Summary of metasearch variants that apply when scores are not available but training data is.

	Average Improvement (%)		
Metasearch Algorithm	Rand-Sets	Best-Worst	Avg of Both
suCombMNZ (sum norm)	8.14	11.12	9.63
suCombSum (ZMUV norm)	8.00	10.30	9.15
druCondorcet-fuse	8.13	9.29	8.71
suCombMNZ (2MUV norm)	6.52	9.73	8.12
ruCondorcet-fuse	8.02	7.85	7.94
suCombSum (sum norm)	5.66	10.08	7.87
suCombMNZ	5.35	9.32	7.33
ruCombSum	4.53	9.42	6.98
dsuCombMNZ	3.94	8.80	6.37
suCombSum	2.87	7.54	5.21
ruCombMNZ	2.25	8.14	5.19
ruBorda-fuse	0.56	6.24	3.40
suCombMed	0.85	3.40	2.13
suCombMNZ (ZMUV norm)	-5.01	-0.59	-2.80
suCombMax	-24.93	-7.45	-16.19
suCombMin	-51.41	-27.07	-39.24
suCombANZ	-57.18	-34.23	-45.70

Table 9.3: Summary of metasearch variants that apply when scores are available but training data is not.

three top ranked algorithms all use the sum or ZMUV score normalization. And the top

	Average Improvement (%)		
Metasearch Algorithm	Rand-Sets	Best-Worst	Avg of Both
Naive Bound	213.72	161.45	187.58
Pareto Bound (approx.)	161.13	126.4	143.77
Majoritarian Bound (approx.)	41.62	20.14	30.88
swCombSum (ZMUV norm)	12.51	10.75	11.63
swCombMNZ (sum norm)	11.73	11.49	11.61
swCombSum (sum norm)	10.34	10.65	10.50
rwCombSum	10.23	9.94	10.09
swCombMNZ	9.62	9.82	9.72
suCombMNZ (sum norm)	8.14	11.12	9.63
suCombSum (ZMUV norm)	8.00	10.30	9.15
druCondorcet-fuse	8.13	9.29	8.71
swCombSum	8.39	8.15	8.27
suCombMNZ (2MUV norm)	6.52	9.73	8.12
ruCondorcet-fuse	8.02	7.85	7.94
suCombSum (sum norm)	5.66	10.08	7.87
suCombMNZ	5.35	9.32	7.33
rwCondorcet-fuse	8.62	5.86	7.24
rwCombMNZ	5.72	8.53	7.13
drwBayes-fuse	5.54	8.50	7.02
ruCombSum	4.53	9.42	6.98
rwBayes-fuse	5.64	8.06	6.85
rwBorda-fuse	6.20	6.63	6.42
dsuCombMNZ	3.94	8.80	6.37
swBayes-fuse	3.61	7.71	5.66
suCombSum	2.87	7.54	5.21
ruCombMNZ	2.25	8.14	5.19
ruBorda-fuse	0.56	6.24	3.40
suCombMed	0.85	3.40	2.13
suCombMNZ (ZMUV norm)	-5.01	-0.59	-2.80
suCombMax	-24.93	-7.45	-16.19
suCombMin	-51.41	-27.07	-39.24
suCombANZ	-57.18	-34.23	-45.70

Table 9.4: Summary of the performance of metasearch techniques that apply when scores and training data are available. The upper bounds are also shown.

five all use performance weighting. Note that the standard technique, suCombMNZ, places 13th.

9.2 Contributions

The key contributions of this work are:

1. We performed the first study of *metasearch consistency*. We have shown that metasearch improves not only the performance of document retrieval, but also its consistency over the queries.

2. We have shown that *performance weighting* is an important technique for improving metasearch performance.

3. We proposed two *relevance score normalization* algorithms that, together with performance weighting, yielded the best-performing metasearch algorithms we know of (11.6% average improvement over the best input system). This indicates that score normalization is an important problem for further study. The problem with the standard technique appears to be its dependence upon statistical outliers.

4. We developed a *naive Bayes* metasearch algorithm based on a simple mathematical model that yields similar performance to the standard CombMNZ algorithm (7.2% average improvement over the best input system).

5. We introduced the use of *Social Choice Theory* to the metasearch problem, modeling it as a democratic election.

6. We proposed *Condorcet-fuse*, based on a majoritarian voting model, and developed a near-linear time, sorting-based algorithm for computing it. Even without relevance scores or training data Condorcet-fuse is one of the highest performing algorithms in all four input classes (8.7% average improvement over the best input system).

7. We proposed the first three *upper bounds* for the metasearch problem. One of them tightly bounds a small class of algorithms that includes Condorcet-fuse.

8. We thoroughly tested all proposed algorithms with two experimental protocols over four data sets.

Perhaps the most pressing question that we are left with is this: given a set of input systems to combine, how should we properly account for dependence among them? We have theoretical evidence that input dependence is a problem: the derivation of the Bayes-fuse algorithm requires the input systems to be independent. And we have empirical evidence that it is a problem: we identified clear dips in the performance of Condorcet-fuse due to dependence. We proposed the dependence filtering algorithm and showed that it does improve the performance of Condorcet-fuse and Bayes-fuse, but it only removes the most obvious offenders and is certainly too simplistic to fully address the problem. The difficulties include that (1) it is easy to eliminate too much useful information while trying to purge out dependence, and (2) it is easy to wind up focusing on the worse input systems, which, being worse, and therefore more like random, are naturally more independent of each other. If two inputs largely agree, it might be because they are both right. Or, it might be because they are dependent. Discerning between these two cases requires a delicate balance. This seems a rich area for further study.

Bibliography

[1] Avi Arampatzis and André van Hameren. The score-distributional threshold optimization for adaptive binary classification tasks. In Croft et al. [16], pages 285–293.

[2] Brian T. Bartell, Garrison W. Cottrell, and Richard K. Belew. Automatic combination of multiple ranked retrieval systems. In W. Bruce Croft and C.J. van Rijsbergen, editors, *SIGIR'94, Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 173–181, Dublin, Ireland, July 1994. Springer-Verlag, London.

[3] Brian T. Bartell, Garrison W. Cottrell, and Richard K. Belew. Learning to retrieve information. In L. Niklasson and M. Boden, editors, *Current trends in connectionism: Proceedings of the 1995 Swedish Conference on Connectionism*, 1995.

[4] Brian Theodore Bartell. *Optimizing Ranking Functions: A Connectionist Approach to Adaptive Information Retrieval.* PhD thesis, University of California, San Diego, 1994.

[5] Mandis Beigi, Ana B. Benitez, and Shih-Fu Chang. MetaSEEk: A content-based metasearch engine for images. In *Storage and Retrieval for Image and Video Databases (SPIE)*, pages 118–128, 1998.

[6] N.J. Belkin, C. Cool, W.B. Croft, and J.P. Callan. The effect of multiple query representations on information retrieval system performance. In Robert Korfhage, Edie Rasmussen, and Peter Willett, editors, *SIGIR'93, Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 339–346, Pittsburgh, PA, USA, June 1993. ACM Press, New York.

[7] N.J. Belkin, P. Kantor, C. Cool, and R. Quatrain. Combining evidence for information retrieval. In Harman [32], pages 35–43.

[8] K. Bharat and A. Z. Broder. A technique for measuring the relative size and overlap of public Web search engines. In *The 7th WWW Conference*, pages 379–388, 1998.

[9] Chris Buckley and Ellen M. Voorhees. Evaluating evaluation measure stability. In Nicholas J. Belkin, Peter Ingwersen, and Mun-Kew Leong, editors, *SIGIR'00, Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 33–40, Athens, Greece, July 2000. ACM Press, New York.

[10] Jamie Callan. Distributed information retrieval. In Croft [14], chapter 5.

[11] J.P. Callan, Z. Lu, and W.B. Croft. Searching distributed collections with inference networks. In Fox et al. [22], pages 21–28.

[12] William S. Cooper. Some inconsistencies and misnomers in probabilistic information retrieval. In *SIGIR'91, Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Chicago, Illinois, October 1991. ACM Press, New York.

[13] Nick Craswell, David Hawking, and Paul Thistlewaite. Merging results from isolated search engines. In *Proceedings of the Tenth Australasian Database Conference*, Aukland, New Zealand, January 1999. Springer-Verlag.

[14] W. Bruce Croft, editor. *Advances in Information Retrieval: Recent Research from the Center for Intelligent Information Retrieval.* The Kluwer International Series on Information Retrieval. Kluwer Academic Publishers, 2000.

[15] W. Bruce Croft. Combining approaches to information retrieval. In *Advances in Information Retrieval: Recent Research from the Center for Intelligent Information Retrieval* [14], chapter 1.

[16] W. Bruce Croft, David J. Harper, Donald H. Kraft, and Justin Zobel, editors. *SIGIR'01, Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New Orleans, Louisiana, USA, September 2001. ACM Press, New York.

[17] W. Bruce Croft, Alistair Moffat, C.J. van Rijsbergen, Ross Wilkinson, and Justin Zobel, editors. *SIGIR'98, Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia, August 1998. ACM Press, New York.

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

[19] Marquis de Condorcet. Essai sur l'application de l'analyse à la probabilité des decisions rendues à la pluralité des voix, 1785.

[20] Daniel Dreilinger and Adele E. Howe. Experiences with selecting search engines using metasearch. *ACM Transactions on Information Systems*, 15(3):195–222, 1997.

[21] H. Leonard Fisher and Dennis R. Elchesen. Effectiveness of combining title words and index terms in machine retrieval searches. *Nature*, 238:109–110, July 1972.

[22] E. Fox, P. Ingwersen, and R. Fidel, editors. *SIGIR'95, Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Seattle, Washington, July 1995. ACM Press, New York.

[23] Edward A. Fox, M. Prabhakar Koushik, Joseph Shaw, Russell Modlin, and Durgesh Rao. Combining evidence from multiple searches. In D.K. Harman, editor, *The First Text REtrieval Conference (TREC-1)*, pages 319–328, Gaithersburg, MD, USA, March 1993. U.S. Government Printing Office, Washington D.C.

[24] Edward A. Fox, Gary L. Nunn, and Whay C. Lee. Coefficients for combining concept classes in a collection. In Yves Chiaramella, editor, *SIGIR'88, Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 291–307, Grenoble, France, June 1988. ACM.

[25] Edward A. Fox and Joseph A. Shaw. Combination of multiple searches. In Harman [32], pages 243–249.

[26] Kevin L. Fox, Ophir Frieder, Margaret Knepper, and Eric Snowberg. SENTINEL: A multiple engine information retrieval and visualization system. *Journal of the ASIS*, 50(7), May 1999.

[27] W Garbe. BINGOOO—transforming the world wide web into a virtual database. *Wirtschaftsinformatik*, 43(5):511, October 2001.

[28] S. Gauch and G. Wang. Information fusion with profusion. In *WebNet'96, Proceedings of the World Conference of the Web Society*, San Francisco, CA, October 1996.

[29] S. Gauch, G. Wang, and M. Gomez. Profusion: Intelligent fusion from multiple, distributed search engines. *The Journal of Universal Computer Science*, 2(9):637–649, September 1996.

[30] Luis Gravano, Chen-Chuan K. Chang, Héctor García-Molina, and Andreas Paepcke. STARTS: Stanford proposal for Internet meta-searching. In *Proceedings of the 1997 ACM SIGMOD Conference*, pages 207–218, 1997.

[31] Luis Gravano and Yannis Papakonstantinou. Mediating and metasearching on the Internet. *Data Engineering Bulletin*, 21(2):28–36, 1998.

[32] D.K. Harman, editor. *The Second Text REtrieval Conference (TREC-2)*, Gaithersburg, MD, USA, March 1994. U.S. Government Printing Office, Washington D.C.

[33] D.K. Harman, editor. *Overview of the Third Text REtrieval Conference (TREC-3)*, Gaithersburg, MD, USA, April 1995. U.S. Government Printing Office, Washington D.C.

[34] Adele E. Howe and Daniel Dreilinger. SAVVYSEARCH: A metasearch engine that learns which search engines to query. *AI Magazine*, 18(2):19–25, 1997.

[35] David A. Hull, Jan O. Pedersen, and Hinrich Schütze. Method combination for document filtering. In Hans-Peter Frei, Donna Harman, Peter Schäuble, and Ross Wilkinson, editors, *SIGIR'96, Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 279–287, Zurich, Switzerland, August 1996. ACM Press, New York.

[36] J. Katzer, M. J. McGill, J. A. Tessier, W. Frakes, and P. Dasgupta. A study of the overlap among document representations. *Information Technology: Research and Development*, 1:261–274, 1982.

[37] Jerry S. Kelly. *Social Choice Theory: An Introduction.* Springer-Verlag, 1988.

[38] J. G. Kemeny. *Random Essays on Mathematics, Education, and Computers.* Prentice Hall, 1964.

[39] Steve Lawrence and C. Lee Giles. Context and page analysis for improved web search. *IEEE Internet Computing*, pages 38–46, July 1998.

[40] Steve Lawrence and C. Lee Giles. Searching the world wide web. *Science*, 280:98–100, 1998.

[41] Steve Lawrence and C. Lee Giles. Accessibility of information on the web. *Nature*, 400:107–109, 1999.

[42] Joon Ho Lee. Combining multiple evidence from different properties of weighting schemes. In Fox et al. [22], pages 180–188.

[43] Joon Ho Lee. Analyses of multiple evidence combination. In Nicholas J. Belkin, A. Desai Narasimhalu, and Peter Willett, editors, *SIGIR'97, Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 267–275, Philadelphia, Pennsylvania, USA, July 1997. ACM Press, New York.

[44] Joon Ho Lee. Combining multiple evidence from different relevant feedback networks. In *Database Systems for Advanced Applications*, pages 421–430, 1997.

[45] David D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 4–15, Chemnitz, Germany, 1998. Springer Verlag, Heidelberg, DE.

[46] Dennis V. Lindley. Reconciliation of probability distributions. *Operations Research*, 1983.

[47] R. Manmatha, T. Rath, and F. Feng. Modeling score distributions for combining the outputs of search engines. In Croft et al. [16], pages 267–275.

[48] R. Manmatha and H. Sever. A formal approach to score normalization for meta search. In *Proceedings of the Human Language Technology Conference (HLT2002) (to appear), also Center for Intelligent Information Retrieval (CIIR) Tech Report IR-242*, March 2002.

[49] M. Catherine McCabe, Abdur Chowdhury, David A. Grossman, and Ophir Frieder. A unified environment for fusion of information retrieval approaches. In *Proceedings of the 8th International Conference on Information Knowledges and Retrieval*, Kansas City, Missouri, USA, November 1999.

[50] M. McGill, M. Koll, and T. Norreault. An evaluation of factors affecting document ranking by information retrieval systems. Technical report, Syracuse University School of Information Studies, Syracuse, NY, October 1979.

[51] W. J. Meng, Z. H. Wu, C. Yu, and Z. G. Li. A highly scalable and effective method for metasearch. *ACM Transactions on Information Systems*, 19(3):310–335, July 2001.

[52] Hervé Moulin. *Axioms of Cooperative Decision Making.* Cambridge University Press, 1988.

[53] Kwong Bor Ng. *An Investigation of the Conditions for Effective Data Fusion in Information Retrieval.* PhD thesis, School of Communication, Information, and Library Studies, Rutgers University, 1998.

[54] Kwong Bor Ng and Paul B. Kantor. An investigation of the preconditions for effective data fusion in IR: A pilot study. In *Proceedings of the 61th Annual Meeting of the American Society for Information Science*, 1998.

[55] Kwong Bor Ng and Paul B. Kantor. Predicting the effectiveness of naive data fusion on the basis of system characteristics. *Journal of the American Society for Information Science*, 51(13):1177–1189, 2000.

[56] Kwong Bor Ng, David Loewenstern, Chumki Basu, Haym Hirsh, and Paul B. Kantor. Data fusion of machine-learning methods for the TREC5 routing task (and other work). In Voorhees and Harman [81], pages 477–487.

[57] Justin Picard. Modeling and combining evidence provided by document relationships using probabilistic argumentation systems. In Croft et al. [17], pages 182–189.

[58] W. H. Press, S. A. Teukolsky, W. T. Vettering, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1995.

[59] T. B. Rajashekar and W. Bruce Croft. Combining automatic and manual index representations in probabilistic retrieval. *Journal of the American Society of Information Science*, 46(4):272–283, 1995.

[60] *Content-Based Multimedia Information Access (RIAO)*, Paris, France, April 2000.

[61] William H. Riker. *Liberalism Against Populism.* Waveland Press, 1982.

[62] Donald G. Saari. *Basic Geometry of Voting.* Springer-Verlag, 1995.

[63] Donald G. Saari. Explaining all three-alternative voting outcomes. *Journal of Economic Theory*, 87(2):313–355, August 1999.

[64] T. Saracevic and P. Kantor. A study of information seeking and retrieving. III. searchers, searches, overlap. *Journal of the ASIS*, 39(3), 1988.

[65] Jacques Savoy, Anne Le Calvé, and Dana Vrajitoru. Report on the TREC-5 experiment: Data fusion and collection fusion. In Voorhees and Harman [81], pages 489–502.

[66] Erik Selberg and Oren Etzioni. On the instability of web search engines. In RIAO [60], pages 223–235.

[67] Erik Warren Selberg. *Towards Comprehensive Web Search.* PhD thesis, University of Washington, 1999.

[68] Joseph A. Shaw and Edward A. Fox. Combination of multiple searches. In Harman [33], pages 105–108.

[69] Bo Shu and Subhash Kak. A neural network-based intelligent metasearch engine. *Information Sciences*, 120:1–11, 1999.

[70] Paul Thompson. A combination of expert opinion approach to probabilistic information retrieval, part 1: the conceptual model. *Information Processing and Management*, 26(3):371–382, 1990.

[71] Paul Thompson. A combination of expert opinion approach to probabilistic information retrieval, part 2: mathematical treatment of CEO model 3. *Information Processing and Management*, 26(3):383–394, 1990.

[72] Howard Turtle and W. Bruce Croft. Evaluation of an inference network-based retrieval model. *ACM Transactions on Information Systems*, 9(3):187–222, July 1991.

[73] Merijn van Erp and Lambert Schomaker. Variants of the Borda count method for combining ranked classifier hypotheses. In *Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition*, pages 443–452, Amsterdam, September 2000. International Unipen Foundation.

[74] Christopher C. Vogt. *Adaptive Combination of Evidence for Information Retrieval.* PhD thesis, University of California, San Diego, 1999.

[75] Christopher C. Vogt. How much more is better? Characterizing the effects of adding more IR systems to a combination. In RIAO [60], pages 457–475.

[76] Christopher C. Vogt and Garrison W. Cottrell. Fusion via a linear combination of scores. *Information Retrieval*, 1(3):151–173, October 1999.

[77] Christopher C. Vogt, Garrison W. Cottrell, Richard K. Belew, and Brian T. Bartell. Using relevance to train a linear mixture of experts. In Voorhees and Harman [81], pages 503–515.

[78] E. Voorhees, Narendra K. Gupta, and Ben Johnson-Laird. The collection fusion problem. In Harman [33], pages 95–104.

[79] E. Voorhees and D. Harman. Overview of the Eighth Text REtrieval Conference (TREC-8). In D.K. Harman, editor, *The Eighth Text REtrieval Conference (TREC-8)*, Gaithersburg, MD, USA, 2000. U.S. Government Printing Office, Washington D.C.

[80] Ellen M. Voorhees, Narendra K. Gupta, and Ben Johnson-Laird. Learning collection fusion strategies. In Fox et al. [22], pages 172–179.

[81] E.M. Voorhees and D.K. Harman, editors. *The Fifth Text REtrieval Conference (TREC-5)*, Gaithersburg, MD, USA, 1997. U.S. Government Printing Office, Washington D.C.

[82] Jinxi Xu and Jamie Callan. Effective retrieval with distributed collections. In Croft et al. [17].