Condorcet Fusion for Improved Retrieval [*]

Mark Montague
Department of Computer Science
Dartmouth College
6211 Sudikoff Laboratory
Hanover, NH 03755
montague@cs.dartmouth.edu

Javed A. Aslam
Department of Computer Science
Dartmouth College
6211 Sudikoff Laboratory
Hanover, NH 03755
jaa@cs.dartmouth.edu

ABSTRACT

We present a new algorithm for improving retrieval results by combining document ranking functions: *Condorcet-fuse*. Beginning with one of the two major classes of voting procedures from Social Choice Theory, the Condorcet procedure, we apply a graph-theoretic analysis that yields a sorting-based algorithm that is elegant, efficient, and effective. The algorithm performs very well on TREC data, often outperforming existing metasearch algorithms whether or not relevance scores and training data is available. Condorcet-fuse significantly outperforms Borda-fuse, the analogous representative from the other major class of voting algorithms.

1. INTRODUCTION

Metasearch is the term usually applied to techniques that combine final results from a number of search engines in order to improve 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. It then computes a single ranked list as output, which is usually an improvement over any of the input lists (as measured by standard IR performance metrics).

Current metasearch algorithms can be characterized by the data they require: whether they need relevance scores or only ranks, and whether they require training data or not. See Figure 1.

External metasearch, one use of fusion, takes existing, "complete" search systems and tries to improve upon them by combining them. External metasearch engines include the Web metasearch engines, for example MetaCrawler, ProFusion, SavvySearch, MetaFerret, InFind, etc. They combine the output of "complete" search engines, as a form of post-processing, value-adding stage. See Figure 2.

Internal metasearch, on the other hand, puts fusion at the heart of a retrieval system. Metasearch offers a system-

	no training data	training data
ranks only	Condorcet-fuse Borda-fuse rCombMNZ	Weighted Condorcet-fuse Weighted rCombMNZ
relevance scores	CombMNZ	Weighted CombMNZ

Figure 1: Metasearch techniques may or may not require relevance scores, and may or may not require training. Listed are metasearch algorithms that we shall discuss.

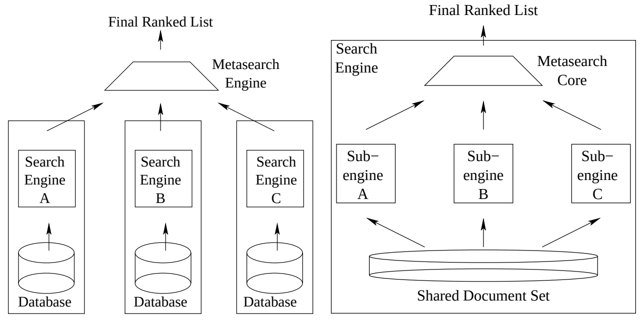

Figure 2: *External metasearch* (left), where a metasearch engine is used to combine three complete and independent search engines, versus *internal metasearch* (right), where a metasearch algorithm is used to combine three cooperating sub-engines over the same set of documents.

atic way of incorporating all of the various types of evidence available to a search engine. Fusion techniques have the potential of combining effectively the information contained in

*This work generously supported by NSF grants EIA-9802068, BCS-9978116, and Career Award CCR 00-93131.

disparate components. 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, keywords, and structure tags. For each of these elements, numerous indexing and searching algorithms may exist. Metasearch can elegantly incorporate all of this information by merging the results of specialized sub-engines.

In the case of internal metasearch, relevance scores are likely available, but in external metasearch often only ranks are available. But perhaps the biggest distinction between internal and external metasearch is the amount of overlap among the document sets of the input systems. In any use of metasearch the document sets of the input systems may be:

- *disjoint,* as in the case of distributed retrieval,
- *overlapping,* as in the case of external metasearch, or
- *identical,* as is likely in the case of internal metasearch.

In this paper we address only the "data fusion" scenario, when the input systems have identical document sets. The application which inspires this study is that of internal metasearch: using fusion as a technique to improve retrieval within one search engine. For simplicity, and in accordance with common usage, in the remainder of this paper we refer to this as the metasearch problem.

Metasearch offers several advantages over traditional monolithic approaches to retrieval. First, metasearch improves upon the performance of individual search engines because different retrieval methods often return very different irrelevant documents, but many of the same relevant documents [19]. Second, metasearch provides more consistent and reliable performance than individual search engines. Since metasearch aggregates the advice of several systems, the fusion tends to smooth out the idiosyncracies of any one system, yielding a more reliable search system [20]. Third, the metasearch architecture is inherently modular. A highly specialized "sub-engine" module can be developed and fine-tuned for each information source about the documents in the collection (such as word frequencies, textual structure within a document, hyperlink structure between documents, etc.) Each sub-engine could be used alone as a search engine. But, by querying all the engines in parallel and combining their results using metasearch, performance is improved. Finally, metasearch leads to focused ranking algorithms that can take advantage of novel, highly specific information sources within documents. To prove the worth of a new ranking technique, one can use a standard metasearch algorithm to fuse their results with standard ranking techniques. If the fused result significantly improves upon the standard techniques alone, then they have indeed tapped a source of novel information.

In this paper we present a new algorithm appropriate for internal metasearch called *Condorcet-fuse*, which is (1) inspired by the Condorcet majoritarian voting algorithm, (2) analysed in terms of graph theory, and (3) reduces to a simple and efficient sorting technique. We present experimental results for Condorcet-fuse that show that it usually outperforms standard metasearch algorithms.

2. RELATED WORK

The use of data fusion to combine document retrieval results has been active in IR research since 1972 when Fisher and Elchesen [10] 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. This work has been followed by more extensive studies, especially in the recent past (for example, [2, 22, 31, 26, 29, 14, 33, 12, 4, 18, 16, 19, 23, 32, 5, 36, 28]. Many systems entered in the TREC competition have taken advantage of metasearch effects [13, 27, 24]. For an excellent survey of the literature see [6].

Key contributions to understanding metasearch include those made by Thompson [29, 30] who proposes a Bayesian model called the Combination of Expert Opinion (CEO) model; Fox, Shaw et al. [12, 13, 27] who developed the CombSum and CombMNZ algorithms; Lee [18, 19] who performed more experiments with these Comb algorithms; and Bartell [3, 2] and Vogt [34, 33, 31, 32] who experiment with linearly combining the normalized relevance scores given to each document.

CombSum computes a new relevance score for each document d as the sum of the normalized relevance scores given to d by the input systems. In terms of ranking the documents, this is equivalent to averaging the scores d is given. CombMNZ, one of the most effective methods to date and thus standard, assigns d the same relevance score as CombSum, but multiplied by the number of input systems that retrieved d. This weighting makes CombMNZ outperform CombSum slightly on most data sets. In this work we compare our new algorithms against CombMNZ (and rCombMNZ—the same algorithm, but with ranks used to simulate scores).

In previous work (Aslam and Montague [1]), we first proposed adapting voting algorithms to solve the metasearch problem. We developed a metasearch algorithm based on the Borda count voting procedure. The Borda count is the central representative of one class of voting procedures: positional algorithms. The Condorcet-fuse model that we propose here is based on the other major class of voting procedures: majoritarian algorithms. In this paper we show that, at least for TREC data, the Condorcet algorithm is the superior technique.

3. THE CONDORCET MODEL

There is a natural analogy between metasearch and voting [1]. In an election, the candidates are ranked as a function of the votes. We can view the ranking of documents from different systems by a metasearch algorithm as a special instance of the voting problem where documents correspond to candidates and input retrieval systems are the voters. The ranking of documents is done according to some function of the input system rankings. To apply a given voting algorithm to metasearch, however, we need to generalize the voting algorithms slightly; voting procedures usually only compute the top candidate while in metasearch we are interested in the entire ranked list of candidates.

3.1 The Social Choice Voting Model

Our inspiration for the metasearch model used in this paper comes from the field of *Social Choice Theory* [25, 21, 17], which studies voting algorithms as techniques to make group (social) decisions (choice). More specifically, we apply to metasearch the ideas from voting algorithms that emerged in the 18th century to address the shortcomings of simple majority voting when there are more than two candidates. The work on these more sophisticated algorithms was spearheaded by Borda [8] and Condorcet [9], whose two very different approaches founded the two main schools of thought in voting. Their algorithms for voting have been at the center of a large corpus of research in Social Choice in this century.

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:

$$\boxed{\begin{array}{l} 3:\ a,b,c,d,e \\ 3:\ e,b,c,a,d \\ 2:\ c,b,a,d,e \\ 2:\ c,d,b,a,e \end{array}}$$

A *social choice function* is a function that maps voting profiles to a set of candidates—the winners. For the example profile, the simple majority rule (or *plurality* voting) dictates that candidate c wins, since it received four first place rankings, more than any other candidate.

Riker [25] distinguishes between *majoritarian* voting algorithms (which are based on a series of pairwise comparisons of candidates), and *positional* methods (which are based on the ranks a candidate receives.) Positional algorithms compute a score for each candidate based on the positions, or ranks, given to each candidate by the voters. The common plurality algorithm (simple majority), for example, gives one point for each time a candidate is ranked first, and zero points for any other position. The Borda count is perhaps the most sensible positional voting procedure. In the Borda count, for each voter, the top candidate receives n points (if there are n candidates in the election), the second candidate receives $n-1$ points, and so on. The candidate with the most points wins. In our example, according to the Borda count, candidates b and c tie for first place, with a total of 38 points each.

The Condorcet voting algorithm is a majoritarian method which specifies that the winner of the election is the candidate(s) that beats or ties with every other candidate in a pair-wise comparison. In the example profile, candidate b is ranked ahead of c in six of the ten profiles. Thus in a simple majority run-off election between these two candidates, b would receive six of the ten votes cast. Indeed, in this example, b beats every other candidate in a head-to-head comparison, so b is the Condorcet winner.

An important result from the field of Social Choice is May's theorem, which states 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)" [21]. This lends support to the Condorcet algorithm, since the Condorcet winner is that candidate that wins (or ties in) every possible pairwise majority contest.

3.2 Condorcet Metasearch

As we have seen, intuitively, 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 by modeling the election with a directed graph of candidates which we call the *Condorcet graph*. A Hamiltonian traversal of this graph will produce the election rankings. Algorithm 2 shows an instantiation of this idea to metasearch. Unfortunately, just to generate the graph takes $O(n^2k)$ time where

Algorithm 1 Simple Majority Runoff.

```
1: count = 0
2: for each of the $k$ search systems $S_i$ do
3:     If $S_i$ ranks $d_1$ above $d_2$, count++
4:     If $S_i$ ranks $d_2$ above $d_1$, count−−
5: If count > 0, rank $d_1$ better than $d_2$
6: Else rank $d_2$ better than $d_1$
```

Algorithm 2 Theoretic Condorcet Metasearch.

```
1: for all pairs $(d_1, d_2)$ of documents do
2:     Use Algorithm 1 to compute the Condorcet Graph
3: Compute Hamiltonian path of Condorcet graph
```

n is the number of documents in the metasearch pool and k is the number of search engines, which makes it too slow for real document collections. For the rest of this section we formalize this algorithm, examine its correctness, and derive an efficient version called *Condorcet-fuse*.

3.2.1 The Condorcet Graph

Given a voting profile for an election with n candidates, its corresponding Condorcet graph $G = (V, E)$ has one vertex for each of the n candidates. For each candidate pair (x, y), there exists an edge from x to y (denoted by $x \to y$) if x would receive at least as many votes as y in a head-to-head contest. In other words, $x \to y$ if x is ranked above y by at least as many voters as ranked y above x. For the candidates that tie there is an edge pointing in each direction (denoted $x \leftrightarrow y$).

The Condorcet graph of any profile contains at least one edge between every pair of candidates. We call graphs with at least one edge between any two nodes *semi-complete*.

3.2.2 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 $n-1$. For some voting profiles, instead of a single winner there is an entire equivalence class of winners. This phenomenon is called *the voting paradox* and is due to the fact that the Condorcet graphs may contain cycles. The simplest example of a "paradoxical" profile has three voters:

$$\boxed{\begin{array}{l} 1:\ a,b,c \\ 1:\ b,c,a \\ 1:\ c,a,b \end{array}}$$

In this profile, a beats b twice, b beats c twice, and c beats a twice. The resulting Condorcet graph is shown in Figure 3.

Although within the field of Social Choice Theory the existence of cycles in the Condorcet graph has been viewed as a paradox (see especially Arrow's famous theorem), we submit that the possibility of cycles is not an artifact or an indicator of an inferior voting method. On the contrary, algorithms that cannot generate cycles ignore the actual complexity of the profile. Fortunately, we have the freedom in the context of metasearch to simply view cycles as ties. For us, the relative ordering of candidates within a cycle is only of secondary importance, whereas their ordering with respect to the rest of the candidates is of primary importance. We

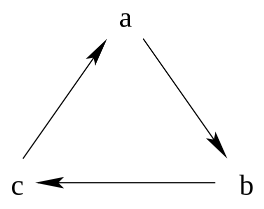

Figure 3: The "voting paradox:" a simple case of cyclic preferences.

can clarify this distinction by considering the Strongly Connected Components of the Condorcet graph.

3.2.3 Strongly Connected Components of Condorcet Graphs

The strongly connected components (SCC) of a graph partition the vertices of the graph. Two vertices are in the same equivalence class if there exists a cycle in the graph containing them. Each SCC of the Condorcet graph contains a set of equivalent nodes. In the strongly connected component graph (SCCG) of a graph, each node represents one SCC and for X, Y in the SCCG, $X \to Y$ if there exists an $x \in X$ and $y \in Y$ such that $x \to y$. In fact, in the case of semi-complete graphs, if $X \to Y$, then $x_i \to y_j$ for all $x_i \in X$ and $y_j \in Y$. Finally, note that SCCGs are acyclic by design.

The strongly connected components of the Condorcet graph can be unambiguously sorted. Since the original graph was semi-complete, so is the SCCG. Together with the fact that the SCCG is acyclic, this implies that 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 m nodes have been exhausted. The unique ordering is $X_0 \to X_1 \to \ldots \to X_m$.

Thus the Condorcet graph is "totally ordered" at the level of the SCCs. Each SCC represents a "pocket" of cycles, within which we say each candidate is tied. The strongly connected components partition the graph into sets of tied candidates.

Condorcet graphs provide a natural model for metasearch but they are costly to compute. We proceed by showing an efficient way for capturing the information in the Condorcet graph without explicitly computing the graph.

3.2.4 Condorcet Paths

We define a *Condorcet-consistent Hamiltonian path* (or *Condorcet path*) to be any Hamiltonian path through the Condorcet graph. Our goal is to efficiently find such a path. Condorcet paths have two properties relevant to metasearch.

Theorem 1. *Every semi-complete graph contains a Hamiltonian path.*

Proof. We prove Theorem 1 by induction. The base case is a graph with one node and is trivial.

For the inductive step, suppose every semi-complete graph with $n-1$ nodes contains a Hamiltonian path. For a problem of size n, let H be the Hamiltonian path for a sub-problem containing only an arbitrary $n-1$ of the nodes. Now the n^{th} node x is introduced, along with its $n-1$ edges to or from the other nodes. There are three cases: (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 doesn't 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. $\Box$

Theorem 2. *If candidate x is in an SCC ranked above the SCC containing y, then x is ranked above y in every Condorcet path.*

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. Thus we say Condorcet paths *properly order* the SCC's. $\Box$

In other words, a randomly chosen Condorcet path orders the candidates that don't tie correctly, and breaks ties arbitrarily.

3.2.5 Efficiency: Condorcet-fuse

We know that we could find an ordering of nodes that properly orders the SCC's by generating the entire Condorcet graph, computing its SCCs, sorting the SCCs, and ordering candidates within each SCC arbitrarily. This algorithm is $O(n^2k)$ for an election with n candidates and k voters, since it takes time $O(n^2k)$ to generate the semi-complete Condorcet graph. This is impractical for large document sets. But Theorems 1 and 2 show that we can also find a reasonable ordering (one that properly orders SCC's) by computing a Condorcet path. We derive an algorithm called *Condorcet-fuse* (see Algorithm 3) that finds a Condorcet path in $O(nk \lg n)$ time. The key idea is to sort the nodes, resulting in a Condorcet path, without ever generating the entire Condorcet graph.

To derive Condorcet-fuse, note that the inductive step in the proof of Theorem 1 resembles InsertionSort: incoming items are compared to all previous items until their proper location is found. Indeed, InsertionSort can be used to find a Condorcet path. The key is to use Algorithm 1 as the comparison function in the sorting algorithm. This comparison function can also be used with MergeSort or QuickSort to find a Condorcet path—we formally prove this for QuickSort as an example.

Theorem 3. *Given a semi-complete graph, if pairwise comparisons between nodes are made according to their edge directions (e.g., $a \to b$ means $a > b$), then sorting the nodes with QuickSort will yield a Hamiltonian path.*

Proof. Recall first how QuickSort works, given a list L: first a pivot element p is chosen from L. Next L is partitioned into two lists A and B such that for all a and b in A and B, respectively, $a < p$ and $p < b$. Then A and B are sorted recursively, and the final sorted list is ApB (that is, A followed by p followed by B).

We may obtain the desired result by strong induction. The base case, a semi-complete graph with one node, is trivial. For the inductive step, assume the elements of L are

Algorithm 3 Condorcet-fuse.

```
1: Create a list $L$ of all the documents
2: Sort($L$) using Algorithm 1 as the comparison function
3: Output the sorted list of documents
```

nodes of a semi-complete graph, which, after sorting can be written ApB where A and B are, by our inductive hypothesis, Hamiltonian paths of their respective semi-complete subgraphs. Then $a \rightarrow p$ and $p \rightarrow b$ for all a and b in A and B, respectively. In particular, the last element of A points to p, and the first element of B is pointed to by p, so ApB is itself a Hamiltonian path. $\square$

In other words, we can compute a Condorcet path by calling QuickSort with Algorithm 1 as the comparison function; the entire graph need not be computed since a comparison between any two nodes can be made when needed by QuickSort. Condorcet-fuse is shown as Algorithm 3.

THEOREM 4. *Condorcet-fuse runs in $O(nk \lg n)$ time.*

PROOF. To efficiently compute a run-off election between two candidates (that is, to compare them) we choose a representation so that instead of having k ranked lists of up to n candidates, we have n lists of the k ranks given to each candidate in some canonical order. This preprocessing step takes time $O(nk)$. A binary comparison can then be made between any two candidates in time $O(k)$, and sorting can be done with $O(nlgn)$ comparisons, yielding total time $O(nk \lg n)$. $\square$

In theory, the algorithm requires $O(nk)$ space. In practice however, not all of this space is needed since not every document is returned by every input system.

3.2.6 Within the SCC's

We know that Condorcet-fuse, by finding a Condorcet path, is guaranteed to properly order the SCC's of the Condorcet graph. But, its behavior within an SCC is of interest as well. An SCC A may have many possible Condorcet paths through it, and any of them may be found by QuickSort, depending upon the sequence of pivots chosen randomly by the algorithm. If a node $a \in A$ has a low in-degree, say 10% of the nodes in A, then it has at least a 90% chance of being ranked above the first pivot—it can only be ranked below it if the pivot chosen is one of those that pairwise-beat a. Thus QuickSort will tend to rank a highly, in a probabilistic sense. The key factor appears to be the relationship of out-degree and in-degree. In this way, Condorcet-fuse appears to approximate the *Copeland Rule* Condorcet extension, which ranks candidates by out-degree minus in-degree.[1] This behavior is likely an important part of the performance of Condorcet-fuse, at least on standard IR test sets which tend to generate large SCCs. It is an interesting open question to determine the probabilistic approximation to the Copeland rule that Quicksort yields.

3.2.7 Weighted Condorcet-fuse

[1]See `www.condorcet.org` for a summary of many such Condorcet extensions.

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

In a simple majority two-candidate election, x beats y if more voters choose x than choose y. We can generalize this to give different voters different sway: 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.

Of course, how you choose the weights is important. It has been shown [1] that even very simple weights based on the performance of each individual input system can be effective. In this work, as a heuristic, we weight each system by its individual mean average precision. We do not suppose this to be optimal, but it illustrates a simple and effective way to take advantage of training data.

3.2.8 Relevance Scores

Interestingly, there does not seem to be an obvious way to make good use of relevance scores in the comparison function. Many metasearch algorithms (like CombSum) can be implemented through careful selection of the comparison function (e.g., in the comparison function, one can calculate the score given a doc a by metasearch algorithm X, the score given a doc b by X, and order them appropriately), but it is not clear how to use the scores in a new way.

4. EXPERIMENTS

4.1 Experimental Setup

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. Each system submits 50 queries, which is enough to allow training and testing when weights are being used. Table 1 shows information about the data sets. We chose the data sets to facilitate comparison with other work: TREC 3 was used by Lee [19], and TREC 5 were used by Vogt [32]. We also chose the more recent TREC 9 because it contains Web data.

Data Set	TREC	No.	Mean Avg Precision			
Name	Topics	Sys	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 1: The data sets used in our experiments. "Topics" are queries and they are numbered consecutively. "No. Sys" contains the number of search systems that submitted results to TREC that year and is the number of systems available for metasearch.

In TREC, each system is allowed to return up to 1000 documents for each query. For TREC 3 and TREC 5, we used the data submitted to the TREC "adhoc" task[2]. For

[2]For TREC 5 we only use runs over the complete data set,

TREC 9, the adhoc task had been replaced by the "web" track. Therefore, over that data set we are fusing the results of Web search engines.

The Vogt data set consists of a subset of the TREC 5 data set as defined by Vogt [32]. In particular, it contains only 10 of the 61 TREC 5 systems, and only 10 of the 50 TREC 5 queries. This subset of retrieval systems was chosen to maximize diversity as measured by nine similarity criteria. 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 the Vogt data because of its diverse inputs and to compare our results to [32]. Note that this data set, due to its diversity, may be the most similar to the internal use of metasearch that we have described.

The last four columns in the table show performance information about each data set: the minimum mean average precision obtained by a system in a given data set, the maximum, the mean, and the standard deviation. The Vogt and TREC 9 data sets 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). 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.

4.1.1 *Training and Testing*

The TREC data used 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 our experiments that require training data (e.g., weighted Condorcet-fuse), 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 five.

4.2 Procedure

We use two experimental procedures: one selects random sets of input systems, and the other combines the best input systems. In our first 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, then the top three systems, and so on, up to the final combination of the top 20 input systems. This experiment is designed to test how the metasearch algorithm performs when combining the best input systems.

In our second 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 trials (or as many as are combinatorially possible) performed as follows. Randomly select a set of k (for $k \in \{2, 4, \ldots, 12\}$) 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 of the k input systems in order to meaningfully assess the improvement gained, if any.) This experiment is designed:

1. to simulate a real-world situation in which one has a set of k techniques available, and must try to obtain the best performance possible from those k. And,

2. to see how the number of input systems affects the performance of the algorithm. A good system will consistently improve on the *best* of its inputs, no matter how many input systems are available.

Although we believe these experiments do test the ability of metasearch to improve upon the best input system being combined, they are designed primarily to compare the performance of one metasearch algorithm against the performance of another.

4.3 Results for Condorcet-fuse

Figures 4 and 5 show experimental results for Condorcet-fuse. The left column of graphs shows results of the random-sets experiment, while the right column shows the best-to-worst experiment. Each row is for a different data set: TREC 3, TREC 5, TREC 9, and Vogt. Each graph shows the mean average precision acheived by a metasearch system—the x-axis shows the number of systems being combined.

The title of each random-sets graph contains confidence-interval information. If the title says "(max err: x)," this means we are at least 90% confident that the actual value of each data point in the graph lies within plus or minus x of the value shown. The low values of x shown in the graphs is an indication of the accuracy of our random-sets experiments. The best-to-worst experiments, however, are inherently less accurate: each data point is the result of only one fusion, not 200 as in the random-sets experiment. For this reason, we consider the random-sets experiment a much more reliable method of comparing metasearch algorithms.

As can be seen in Figure 4, in the random-sets experiment, when only ranks are available the Condorcet-fuse algorithm outperforms rCombMNZ (CombMNZ based on ranks not relevance scores) on each of the four data sets. When relevance scores are available, Condorcet-fuse outperforms CombMNZ on three of the four data sets, despite the fact that Condorcet-fuse doesn't take advantage of the relevance scores.

Table 2 shows that our results for the random-sets experiment are usually significant. After gathering the results of the 200 fusion experiments[3] that went into each data point, we used the sign test to compare Condorcet-fuse to each other metasearch algorithm. From the second column in the table it is clear that Condorcet-fuse usually outperforms (indicated with a "+") the best input system being combined, often significantly (indicated with a "*"). Similarly, it outperforms CombMNZ significantly in almost every case

that is those systems in Category A, not B.

[3]Where combinatorially possible. On the Vogt data set, there are only 45 ways of choosing two systems to combine out of the ten. And as there is only one way to choose ten systems out of ten, we cannot meaningfully compute statistical significance when combining all ten systems.

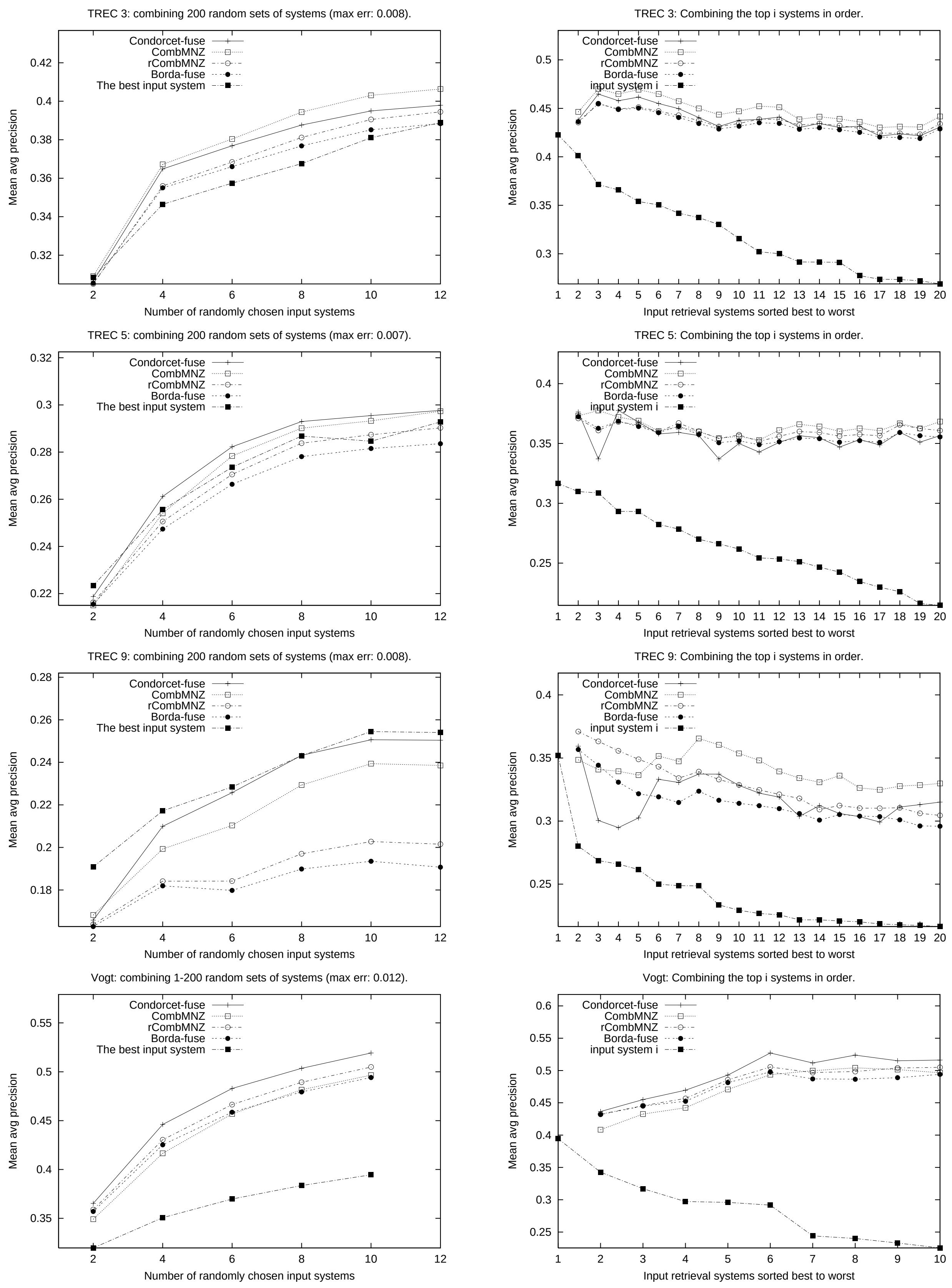

Figure 4: Condorcet-fuse versus CombMNZ and Borda-fuse on four data sets. Also shown is the best input system. If relevance scores are available or not, Condorcet-fuse outperforms CombMNZ in most cases.

except in TREC 3. And it easily outperforms rCombMNZ and Borda-fuse. Not surprisingly, Condorcet-fuse is usually outperformed by its own weighted version that can take advantage of training data. We discuss a dependence-filtered variant of Condorcet-fuse below. Note that this kind of analysis cannot be performed on the best-to-worst experiment where each datapoint represents a single fusion experiment.

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 3. Systems two, three,

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

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

and four are runs by the 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 similar sets of top 1000 documents, they *probably* return similar sets of top 100 documents, and similar sets of top 10 documents, etc.

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 4. 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 4: 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 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. Condorcet-fuse seems particularly sensitive to input system dependence 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).

4.3.1 Dependence Filtering

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 $|\mathcal{S}| = k$, our brute-force implementation of dependence filtering performs $O(k^2)$ similarity comparisons between input systems, and each similarity comparison takes $O(nq)$ time for q queries each with n 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 $O(k^2)$ values. This has not been prohibitively expensive for us since we never fuse more than $k \approx 100$ 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.

Figure 5 shows the results of two variants of Condorcet-fuse: with performance weights, and after dependence filtering. Both improve performance significantly: weights help especially in the random-sets experiment (except over the Vogt data set, where all of the systems have similar performance), while dependence filtering essentially eliminates the performance dips in the best-to-worst experiment.

5. CONCLUSION

We have defined a new model for the metasearch problem, the Condorcet model, adapted from majoritarian voting from the field of Social Choice Theory. We have presented an efficient algorithm, Condorcet-fuse, which permits a simple, sorting-based implementation of the model. When document relevance scores are not available, Condorcet-fuse consistently outperforms the standard CombMNZ. Even when relevance scores are available (which Condorcet-fuse cannot take advantage of), Condorcet-fuse still outperforms CombMNZ on three of the four TREC data sets that we tested. Finally, Condorcet-fuse easily outperforms Borda-fuse, a previously-proposed, voting-based fusion technique.

num sys's	Best input system	CombMNZ	rCombMNZ	Borda-fuse	Weighted Condorcet-fuse	Dependence-filtered Condorcet-fuse
			TREC 3			
2	− 0.4438	− 0.0024*	+ 0.0000*	+ 0.0170*	+ 0.1292	− 0.0000*
4	+ 0.0002*	− 0.0000*	+ 0.0000*	+ 0.0000*	+ 0.1613	− 0.0170*
6	+ 0.0000*	− 0.0000*	+ 0.0000*	+ 0.0000*	− 0.0239*	− 0.2399
8	+ 0.0001*	− 0.0000*	+ 0.0000*	+ 0.0000*	− 0.0085*	+ 0.5000
10	+ 0.0002*	− 0.0000*	+ 0.0000*	+ 0.0000*	− 0.0015*	− 0.2399
12	+ 0.0601	− 0.0000*	+ 0.0000*	+ 0.0000*	− 0.0000*	− 0.2860
			TREC 5			
2	+ 0.1292	− 0.2860	+ 0.0000*	+ 0.0000*	− 0.3358	− 0.0000*
4	+ 0.0331*	+ 0.0000*	+ 0.0000*	+ 0.0000*	− 0.0000*	+ 0.1613
6	+ 0.0601	+ 0.0008*	+ 0.0000*	+ 0.0000*	− 0.0000*	+ 0.1014
8	+ 0.2399	+ 0.0002*	+ 0.0000*	+ 0.0000*	− 0.0000*	− 0.3887
10	+ 0.0002*	+ 0.0085*	+ 0.0000*	+ 0.0000*	− 0.0000*	− 0.3887
12	+ 0.0121*	+ 0.3358	+ 0.0000*	+ 0.0000*	− 0.0000*	− 0.0239*
			TREC 9			
2	− 0.0057*	− 0.0038*	+ 0.0000*	+ 0.0000*	− 0.0000*	− 0.0000*
4	− 0.0449*	+ 0.0000*	+ 0.0000*	+ 0.0000*	− 0.0000*	− 0.0788
6	+ 0.2399	+ 0.0000*	+ 0.0000*	+ 0.0000*	− 0.0000*	− 0.0057*
8	+ 0.0331*	+ 0.0000*	+ 0.0000*	+ 0.0000*	− 0.0000*	− 0.1613
10	+ 0.1984	+ 0.0000*	+ 0.0000*	+ 0.0000*	− 0.0000*	− 0.1613
12	+ 0.2860	+ 0.0000*	+ 0.0000*	+ 0.0000*	− 0.0000*	− 0.0085*
			Vogt			
2	+ 0.0000*	+ 0.0011*	+ 0.0000*	+ 0.0000*	+ 0.0000*	− 0.0000*
4	+ 0.0000*	+ 0.0000*	+ 0.0000*	+ 0.0000*	+ 0.0000*	− 0.0239*
6	+ 0.0000*	+ 0.0000*	+ 0.0000*	+ 0.0000*	+ 0.0000*	+ 0.0601
8	+ 0.0000*	+ 0.0000*	+ 0.0000*	+ 0.0000*	+ 0.0000*	− 0.2283
10	na	na	na	na	na	na

Table 2: Statistical significance results for the random-sets experiment, as calculated by the sign test. A "+" indicates that Condorcet-fuse outperforms the given method. Entries with a "*" are significant above the %95 level. No meaningful significance can be computed when combining all 10 systems for the Vogt data set.

6. REFERENCES

[1] J. A. Aslam and M. Montague. Models for metasearch. In Croft et al. [7], pages 276–284.

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

[3] B. T. Bartell, G. W. Cottrell, and R. K. Belew. Automatic combination of multiple ranked retrieval systems. In W. B. Croft and C. 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.

[4] N. Belkin, P. Kantor, C. Cool, and R. Quatrain. Combining evidence for information retrieval. In Harman [15], pages 35–43.

[5] N. Craswell, D. Hawking, and P. Thistlewaite. Merging results from isolated search engines. In *Proceedings of the Tenth Australasian Database Conference*, Aukland, New Zealand, Jan. 1999. Springer-Verlag.

[6] W. B. Croft. Combining approaches to information retrieval. In W. B. Croft, editor, *Advances in Information Retrieval: Recent Research from the Center for Intelligent Information Retrieval*, The Kluwer International Series on Information Retrieval, chapter 1. Kluwer Academic Publishers, 2000.

[7] W. B. Croft, D. J. Harper, D. H. Kraft, and J. Zobel, editors. *SIGIR'01, Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New Orleans, Louisiana, USA, Sept. 2001. ACM Press, New York.

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

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

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

[11] 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.

[12] E. A. Fox, M. P. Koushik, J. Shaw, R. Modlin, and D. Rao. Combining evidence from multiple searches. In D. Harman, editor, *The First Text REtrieval Conference (TREC-1)*, pages 319–328, Gaithersburg, MD, USA, Mar. 1993. U.S. Government Printing

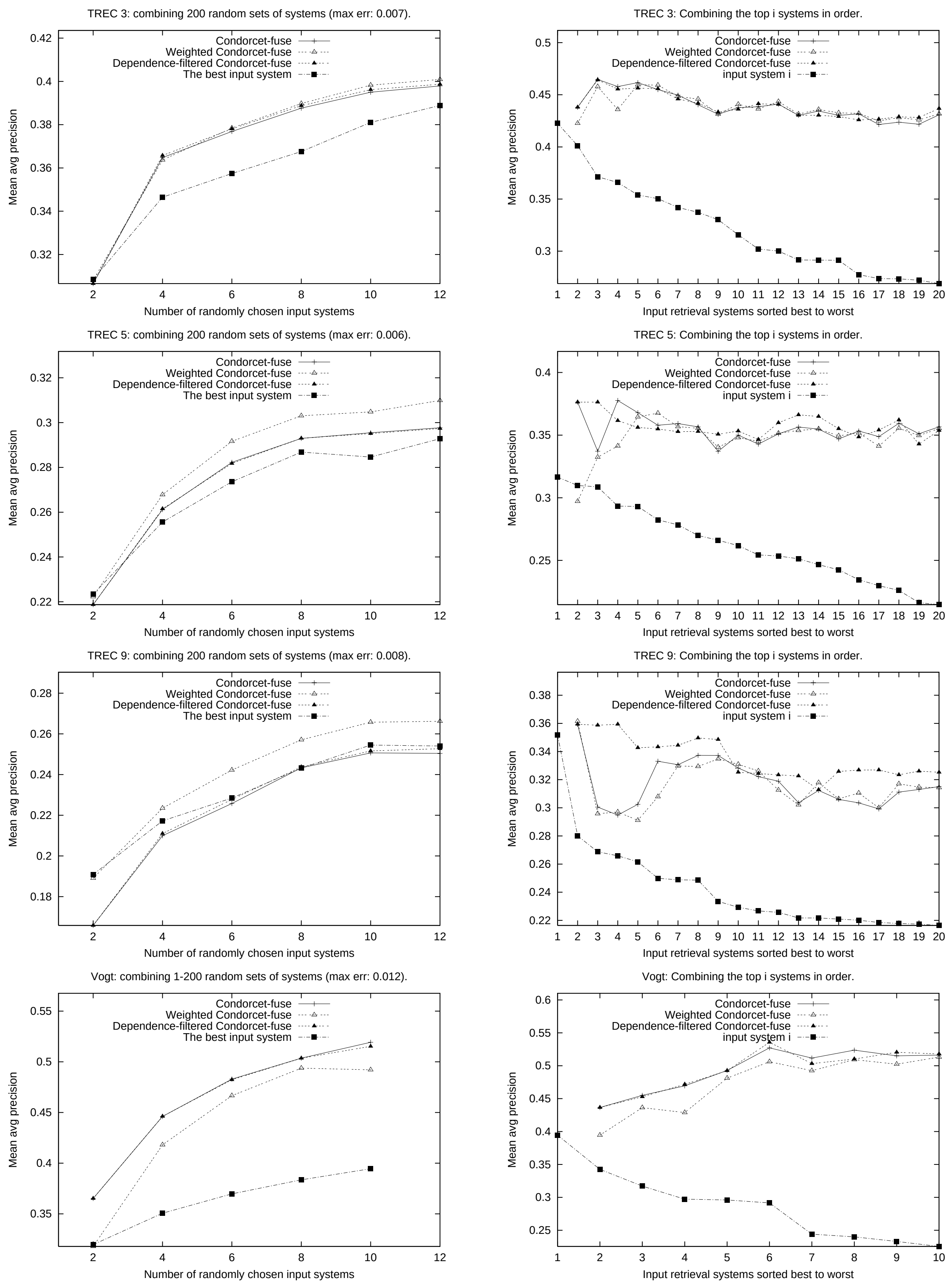

Figure 5: Weighted Condorcet-fuse and dependence-filtered Condorcet-fuse. Performance weights usually improve performance, and filtering out dependent input systems is crucial in the best-to-worst experiment.

Office, Washington D.C.

[13] E. A. Fox and J. A. Shaw. Combination of multiple searches. In Harman [15], pages 243–249.

[14] K. L. Fox, O. Frieder, M. Knepper, and E. Snowberg. SENTINEL: A multiple engine information retrieval and visualization system. *Journal of the ASIS*, 50(7), May 1999.

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

[16] D. A. Hull, J. O. Pedersen, and H. Schütze. Method combination for document filtering. In H.-P. Frei, D. Harman, P. Schäuble, and R. 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, Aug. 1996. ACM Press, New York.

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

[18] J. H. Lee. Combining multiple evidence from different properties of weighting schemes. In Fox et al. [11], pages 180–188.

[19] J. H. Lee. Analyses of multiple evidence combination. In N. J. Belkin, A. D. Narasimhalu, and P. 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.

[20] M. Montague and J. A. Aslam. Metasearch consistency. In Croft et al. [7], pages 386–387.

[21] H. Moulin. *Axioms of Cooperative Decision Making*. Cambridge University Press, 1988.

[22] K. B. 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.

[23] K. B. Ng and P. 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.

[24] K. B. Ng, D. Loewenstern, C. Basu, H. Hirsh, and P. B. Kantor. Data fusion of machine-learning methods for the TREC5 routing task (and other work). In Voorhees and Harman [35], pages 477–487.

[25] W. H. Riker. *Liberalism Against Populism*. Waveland Press, 1982.

[26] E. W. Selberg. *Towards Comprehensive Web Search*. PhD thesis, University of Washington, 1999.

[27] J. A. Shaw and E. A. Fox. Combination of multiple searches. In D. Harman, editor, *Overview of the Third Text REtrieval Conference (TREC-3)*, pages 105–108, Gaithersburg, MD, USA, Apr. 1995. U.S. Government Printing Office, Washington D.C.

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

[29] P. 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.

[30] P. 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.

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

[32] C. C. Vogt. How much more is better? Characterizing the effects of adding more IR systems to a combination. In *Content-Based Multimedia Information Access (RIAO)*, pages 457–475, Paris, France, Apr. 2000.

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

[34] C. C. Vogt, G. W. Cottrell, R. K. Belew, and B. T. Bartell. Using relevance to train a linear mixture of experts. In Voorhees and Harman [35], pages 503–515.

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

[36] E. M. Voorhees, N. K. Gupta, and B. Johnson-Laird. Learning collection fusion strategies. In Fox et al. [11], pages 172–179.