Merging algorithms for Meta-Search

Peng-Fei Li

A thesis submitted for the degree of
Master with Honours in Computing of
The Australian National University

March, 2013

Declaration

To be done.

Acknowledgements

I would like to thank my supervisors, Professor David Hawking and Doctor Paul Thomas.

Abstract

Result merging problems have been a significant issues in federated search. However, there exist no general algorithms that can work well in all situations and no particular algorithm can defeat another in every environment. In this paper, we examine several approaches that are widely used in this area. We built a meta-search engine based on the search engines that are distributed among different departments of The Australian National University. Several merging algorithms have been considered their feasibility to fit into the environment and some are implemented as the broker merging function. The algorithms are then be evaluation based on a manually judgement system and NDCG measurement is applied to compare their performance. The result shows that [to be continue]. And then we proposed a [to be continue] algorithm, and experiment shows that[to be continue].

Contents

List of Figures

Chapter 1

Introduction

Conventional search engines use some technologies to index documents and apply some algorithms to retrieve from those index files. But in some cases, resources of original documents are hard to achieve and easy to be ignored. For example, in the web search application, the documents are obtained from the crawlers, or named spiders. Most of these spiders can crawler information that exists on the internet, however, some webpages are not statically presented. These pages are generated by fetching data from their internal database. And in another scenario, in the desktop search [29], the search engine is aimed to provide an integrated interface to access to all of an individual's information. The information is always varies in data structure and are distributed in different applications. In both of the situations, the documents can be extremely hard to achieve and indexing the documents would be a great challenge. To tackle problem in such a case, the technique of meta-search has been proposed. *Meta-search* is a technique that combine the search results from different search engines to a single point for users to access. In a typical meta-search engine architecture, the core component is called *broker*, which get queries from users and allocate the queries to its component search engines and then merge the results retrieved by those search engines. Following this procedures, four challenges of meta-search remains to be *Collection representation*, *Collection selection*, *Result Merging*. And also, *Performance evaluation* turns out to be a great challenge in this area.

Collection representation, Collection representation is a previous process of collection selection. To select which collection is more likely to have documents that is relevant to a particular information needs, the broker need to have some information on the different collections. Those information could be size, content or major topics of the document collections.

Collection selection, In most cases, it is obvious that not all of the collection will contain relevant documents. Due to the cost of bandwidth as well as request time and accuracy of the final result, it is necessary to get rid of some collections that is less possible to have relevant documents. Many efforts have been make to choose among the various collections and many selection algorithms have been proposed.

Result merging, When all the results are retrieved and gathered, it is another challenge to rearrange them in a reasonable order. The page rank algorithms used in different search engines are different in most cases, so the document scores are not comparable to each other. And in other situations, even the document scores are not accessible. The meta-search merging algorithms are trying to output an accurate list of results ranking in order of document relevance.

Performance evaluation, Evaluation of the performance of information retrieval system is always a tough task. Due to the fact that the relevance of documents are based on the relevance of documents to information needs, instead of the query terms, the

performance remains to be hard to measure.

In many cases, the number of collection is pretty limited so the collection selection problem becomes a less significant problem. In this paper, the main focus is put on the merging algorithms. As discussed above, result merging algorithms are utilised to get an integrated list of results retrieved by the component search engines. In the first section, we will introduce different merging algorithms that has been proposed in previous works, and then we will compare the advantages as well as disadvantages of those algorithms. Finally, we will examine the performance of different algorithms using our evaluation framework and discuss the performance of those algorithms in a realistic environment.

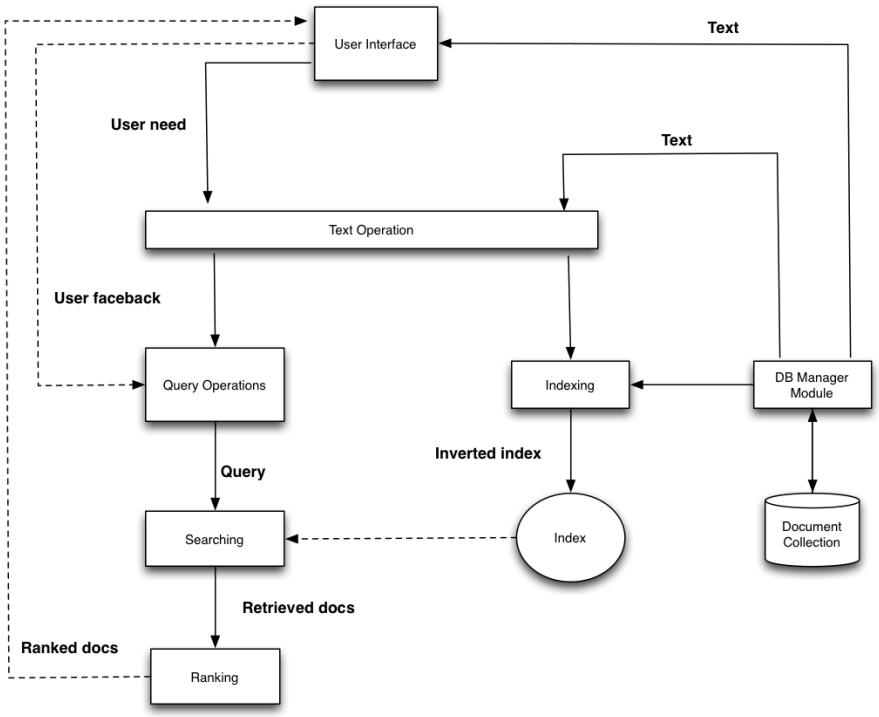
Chapter 2

Literature Review

2.1 Conventional Information Retrieval System

Conventional information retrieval systems provide an interface to users to retrieval documents that are relevant to an information needs from collections which are usually compromised by large volumes of documents. The four major aspects of a conventional information retrieval system are *Information needs*, *Document representation* and *Retrieval model* and *Document ranking*. **Information needs** represent the intended information needs for a searching action and are usually expressed by users using particular query terms. It has usually been referred to as a topic in some other articles. So the whole purpose of an information retrieval system is to meet users' information needs.

Figure 2.1: Typical architecture of information retrieval



2.1.1 Document Representation

The document collection is usually compromised by large amount of documents which usually contain hundreds to thousands lines of worlds or other information formats. It is unrealistic to match strings in each document in the collection. A more practical approach is to represent documents in a particular form, namely index file in the information retrieval area. The indexing process extract every term from documents, conducting tokenizing and stemming process and store them in an index file.

2.1.2 Retrieval Models

There exists several model for information retrieval. Early information retrieval models use boolean methods to retrieve query term

Boolean Model

Vector Space

Probabilistic Model

2.1.3 Document Ranking

2.2 Meta-Search

An alternative approach is to provide an integrated search engine that can send users' query to different search engines and then return the merged result. This method leaves the searching job to each component search engine, which is always built into the document collection and can retrieval its documents through its internal database. This approach is named distributed search [5], sometimes named federated search [**?**] or metasearch. A typical meta-search engine grabs users' query request, do some query expansion, which is optional, and then sends the query to different component search engines, finally the engine collection the results returned by all the collections and merging the resulting in a suitable ranking to display to users (Figure 2.2). People usually separate the meta-search problems into 4 different sub-problems, namely *Resource Description*, *Resource Selection*,*Query Translation*,*Result Merging* [26]. The *Resouce Description* problem is to determine how to represent a collection, it covers the statistic information and topic information of a query. The Resource Selection problem is to select collections with the most possibility to have relevant documents to conduct the search. The *Query Translation* problem is to modify the queries for more appropriate searching among different search engines. The final problem *Result Merging* is to combine the results returned by different search engine into a unified, well-ranked result list.

2.3 Collection Selection

An efficiency issue may arise when the meta-search engine have to search across massive amount of collections. Due to the fact that some collections are mainly focused on one or some particular topics, many of the collections will contain limited number of documents that are relevant to the users' query, or even no documents will return in some extreme cases. And because of the limitation of memory, bandwidth, time and other resources, it would slow down the speed of searching if searches are conducted on all the collections. It

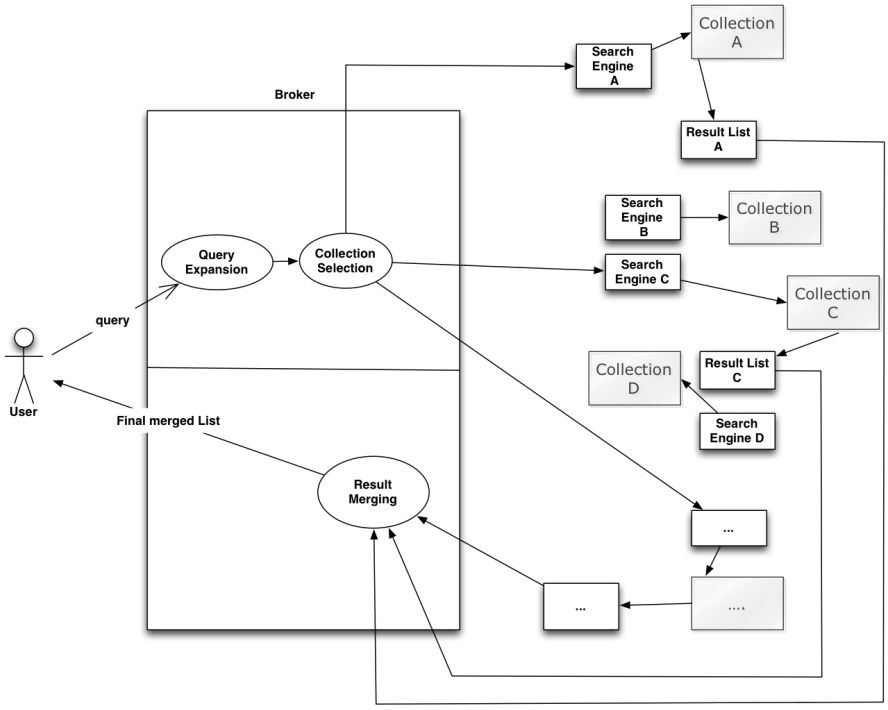


Figure 2.2: A typical architecture of metasearch, the users' requests are fetched by the main component, which is named broker, some query expansion may be executed and the processed query will send to different collections. The collections execute searching on the query and return the retrieval results to the broker

would be a good idea to ignore some collections that are likely to be the last ones to have relevant documents with respect to a particular query or topic in semantic aspect. The problems has been defined as collection selection problems and have drawn many attentions by expertsin this area. Early meta-search engines apply manually cluster collections into different themes or topics and then assign query to different clusters according to its topic, it has already been tried to group collection automatically [7]. Later attempts have been made to treat each collection as a big document and then calculate the similarity between the collection and the query using the bag of words model. In this model, each collection is treated as a big document and other statistic information is stored for later use. A vector space model can be build using the big bag words model, in which all collections as well as queries are built into vectors. Each element into the vector represent the weight of a term to a collection or a query. The weight is often set by the TF-IDF weight which maybe obtained from the cooperative collection statistics or estimation and is used to calculate the similarity of documents or queries. Cosine similarity and Okapi BM25 [21] are widely used to get the similarity of a collection and a query, which is also regard as the collection score in these algorithms. There are also many algorithms based on the lexicon information of the collection. For example, CORI [4, 5] is developed based on an information retrieval system named INQUERY [6], which builds an inference network to search information in a collection. The CORI system calculate the belief of a collection associated to a query and rank them according to the score. Besides all these lexicon based algorithms, documentsurrogate methods [23, 27, 30] and machine learning approaches [33]have also come into use in this area.

2.4 Result Merging

Collection selection problems have been focused on for years and more than 40 algorithms have been developed to solve problem while result merging algorithms have not been paid enough attention. Although collection selection is quite an important factor in meta-search with related to the speed of the system as well as precision of the final result. However, in some cases where the number of collections is not very big, it may be a fact that the collection selection procedure may decrease the efficiency as well as precision since its incomplete searching and complexity in determining if the collection is possible to contain relevant documents. As the results are returned from different sub-components ranked by different algorithms , it becomes another issue how to put them together into an integrated result list. Documents in these results lists are distributed differently and there is limited information about the result document in most cases, which bring much challenge to merge the final result. This problem has been defined as result merging or ranking fusion problems. A formal definition has been given to define ranking fusion problems by [8] and [20]. Given a set of items denoted by U, τ is an ordered list of the elements in a subset of U, which is denoted by S, for example, $\tau = [x_1 \geq x_2 \geq \cdots \geq x_k]$, where $x_i \in S$. If τ contains all the elements of U, *tau* is said to be a $fulllist$. However, full lists are not possible in most cases because the searching engines will not return all the items in the database as a limitation of resources. Assume a set of rank lists $R=\{\tau_1,\tau_2,\tau_3,\cdots,\tau_n\}$, the result merging or rank fusion problem is to find out a new rank list, denoted by $\hat{\tau}$, which is the result of a rank fusion method applied to the rank lists in R. However, there is a danger of regarding result merging problems as ranking fusion problems because it may be hard to say the different rank list are based subsets of different document sets. For example, $\tau_1=[x_1 \geq x_2 \geq \cdots \geq x_k]$, where $x_1, x_2, x_3, \cdots, x_k$ is a subset of U_x,while $\tau_1=[y_1 \geq y_2 \geq \cdots \geq y_k]$, and $y_1, y_2, y_3, \cdots, y_k$ is a subset of U_y. There are many cases and U_x disjoint U_y, which makes the definition hard to express and implement.

2.4.1 Aggregated and Non-aggregated Meta-Search

People try to solve the problem from different aspect, some treat it as a data fusion problem, some scientists regard it as a normal index centralised rank problem besides that the document score need to be normalised, others use machine learning approaches to model the problem. However, actually not all these algorithms can fit into all the scenarios that meta-search are built. As a matter of fact, the reason behind this is that there exists different environment of meta-search. According to the information provided, meta-search can dived into cooperative meta-search and uncooperative meta-search [24]. In cooperative meta-search environment, the statistics information about the collection and the score of each returned document will accessible to meta-search builders, while in uncooperative environment none information is provided expect a search interface. These two types of meta-search may determine to normalise the score based on whether provided score or estimated score. And uncooperative environment is more likely to be the cases where meta-search engines are built. [17] classified metasearch into two categories, namely *Internal metasearch* and *External Metasearch*. The Internal metasearch was described as an architecture, in which the metasearch engines fuse the results from each of its sub-engine based on the shared Document Set, while in the contrast, the External metasearch engines fuse the results from its different component search engines based on different collections(Figure 2.3). However, This kind of classification is more focused on

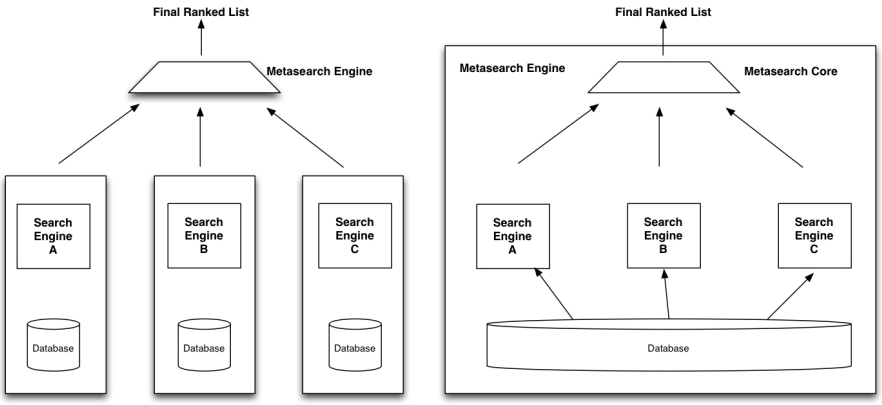


Figure 2.3: Metasearch architectures from [17]

the architecture itself. And not all of the architecture belong to either of the architecture or some architectures are more likely to have characteristics of both categories. It is more practical to classify metasearch engine according to their purposes instead of their architecture. Meta-search is not always used as a way of digging hidden web from the Internet, some times it is used to optimise the performance of search engines by combing the results of different algorithms on a particular source. In that case, there exists large amount of overlap documents between different result list and therefore many linear combination algorithms and data fusion algorithms can be well fitted. So in the case where the purpose is to optimise the performance of multiple search engine, we name it *Aggregated Meta-Search* while *None-aggregated Meta-Search* is a meta-search built to get multiple information from different data collection without considering the document overlap. Actually there is no explicit boundaries between these two types of meta-search engines, for example there may exists many overlap documents between some collections where the purpose of the search is still to dig deep web information. In that case, algorithms are not specific by its definition. The point to point this out is to give a clear image of how the algorithms fit into practise and in what situation can be applied.

2.4.2 General Methods

The term *General Mthods* refers to algorithms that can be well fitted into the both architecture. These algorithms, may cannot outperform some specific algorithms, which will be talked about in the next section, in the particular environments. The algorithms will be talked about in this section is more likely to be a considerable solution in both types of meta-search engines.

Round-Robin

Previous meta-search merge the different lists into a unified one in a simple *Round-Robin* manner [19]. The round-robin algorithm is based on the assumption that the number of relevant documents is approximately the same and distributed evenly in all the collections [18, 19]. But the assumption cannot hold in a real world, where the number of relevant documents varies and the distribution is extraordinarily messy. Rasolofo et.al. [19]also proved that the round-robin methods perform various randomly. It's may be a better idea to give weight to different collection to determine the order of collections when doing

round-robin, which can make the method more stable. So they added some features to each collection to form a biased round-robin algorithm that different collections have collection scores showing their preference in each round-robin circle [19] and saw some precision increase. Round-Robin can fitted into both the Aggregated Meta-search and Non-Aggregated Meta-search, although it can not give a perfect result.

Raw-Score Algorithms

A feasible approach to merge the results is to treat the problem as an index centralised rank problem. In that case, all the documents from different collections are regarded as documents obtained from a big document set which is made up of documents from all the collections. The documents are just ranked according to the documents scores that assigned by their individual search engines. However, because different search engines use different score algorithms according to different criteras, the scores range differently and are always not comparable , so score normalisation is applied to re-arrange the score into a particular range, usually from 0 to 1. For example, [13,20] introduced an algorithm named MinMax to control the range of document score Equation (2.1). The algorithm uses the upper bounds as well as lower bounds of document score in a collection to normalise each document score to the range [0,1].

$$w^{\tau}(i) = \frac{s^{\tau}(i) - min_{j\in\tau}s^{\tau}(j)}{max_{j\in\tau}s^{\tau}(j) - min_{j\in\tau}s^{\tau}(j)} \tag{2.1}$$

$w^{\tau}(i)$ indicates the normalised weight of item $i \in \tau$, and the score of an item assigned by τ is denoted by $s^{\tau}(i)$. They also tried to use the rank assigned by the original search algorithm to get a similarity score, they defined the score *Rank_Sim* :

$$Rank\_Sim(rank) = 1 - \frac{rank - 1}{num\_of\_retrieved\_docs} \tag{2.2}$$

They conducted an experiment on the original score normalisation and rank similarity approach and found the latter algorithm is worse that the previous one in most cases. Then, [20] tried to use $Z - scorenormalisation$ Euqation (2.3) method to normalise the document score.

$$w^{\tau}(i) = \frac{s^{\tau}(i) - \mu_{s^{\tau}}}{\sigma_{s^{\tau}}} \tag{2.3}$$

$\mu_{s^{\tau}}$ denotes the means of scores and $\sigma_{s^{\tau}}$ is the standard deviation. In most cases, the score of each document is not available to meta-search systems. An alternative way is to estimate the score according to accessible information. [19] utilised various document fields to estimate the document score. They generate the document score using a generic document scoring function(Equation (2.4)).

$$w_{ij} = \frac{NQW_i}{\sqrt{L_q^2 + LF_i^2}} \tag{2.4}$$

NQW_i is the number of query words appearing in the processed field of the document i, L_q is the length (number of words) of the query, and LF_i is the length of the processed field of the document i. Then they use the score as the bases for their two new algorithms, namely SM-XX and RR-XX. The SM-XX algorithm use the estimated document score as the unified document score and merge all the documents in the order of the score.

The other algorithm, use the score to re-rank the results in each collection and use the round-robin method to merge the new result lists. The "XX" stands for the document fields such as document title, document summary. Although it shows that the final result of RR-XX is better than original round-robin algorithm, it still leave the distribution of documents in each collection as a problem. In fact normalising the document score is not fair enough to say a document is good enough than others because different collections may have different priorities saying which document is better, this is due to the fact that a collection more focused on a topic may have more weight to judge a document. For example, a user queries some government policy may more likely to intend to get information from a government than to get information from a news website. So, in terms of meta-search merging algorithm, the metasearch system should be a bias decider on different collection both on semantic relationship and the volume of results they returned. Many of the previous algorithms have weighted versions which can improve the efficiency to some degree. Some algorithms use the collection score obtained from the collection selection procedure to schedular a new collection weight, for example,CORI [4, 5] uses a simple linear combination method to normalise the collection score with the collection score get from the collection selection algorithm Equation (2.5),

$$C' = \frac{C - C_{min}}{C_{max} - C_{min}} \tag{2.5}$$

, the normalised score is then used by the raw score based function to estimate the document score by equation (2.6):

$$D' = \frac{D + 0.4 \times D \times C'}{1.4} \tag{2.6}$$

The algorithm turns out to be very stable and efficient,however due to the fact that it requires a meta-search built on the bases of INQUIRY system, the limitation draws quite many problems in implementation in other environments. As an alternative, Rasolofo et.al. [18]proposed a new approach to rank the list from different collections names LMS(using result Length to calculate Merging Score). They estimate the collection score by the The algorithm uses the result length, namely the number of documents retrieved by the collection using Equation (2.7). l_i is the number of documents returned by a collection and K is a constant set to 600 in their paper. C is the number of collections.

$$S_i = \log\left(1 + \frac{l_i \times K}{\sum_{j=1}^{|C|} l_i}\right) \tag{2.7}$$

The collection score was then used to generate a collection weight.

$$w_i = 1 + [(s_i - \bar{s})/\bar{s}] \tag{2.8}$$

where s_i is the ith collection score calculated by the previous formula and $\hat{s}$ is the mean collection score. This approach is like a simplified version of CORI, which just take the size of relevant documents into account. So more weight will be put on the collections with more result documents. However, the algorithm just takes the document length as the only criteria of collection weight, which can get rid of some collections with limited number of relevant documents but very tie-relevant documents. As we can see from all the raw-score algorithms we talked about, none of them can works well on all scenarios, the score

normalised approach ignore the fact that collections are bias in its nature. The generic document function used by [19] may not consider that the title or summary information provided is very limited and different collections return different size of summaries and titles, many sites in fact just return first part of the document. CORI is a stable algorithm, however limited by its architecture. And LMS, on the other hand, is not confident to convince that the size of returned document is the only criteria of collection weight.

Machine Learning Approaches

Apart from those algorithms based on the original scores or ranks, other algorithms tried to build models to fit the problem and train a model by use training data. One of the algorithms is *SSL* [25,26]. The *SSL* algorithm applies a semi-supervised learning approach to train a regression model. The algorithm combine the merging algorithms with the sampling procedure in collection selection. In the collection selection, people always use sufficient sampling queries to get enough documents that can represent each collection. The sampling documents are not discarded, instead, they build a relatively small centralised sample database. Each time SSL executes a query, it sends the query to both the centralised sample database and the original collection. According the the hypotheses that there exists considerable number of overlap documents between the two documents, then a regression model is built to map the individual score to the global centralised index score. For each overlap document $d_{i,j}$, it has a pair of scores from original collection $S_i(d_{i,j})$ and $S_C(d_{i,j})$ from centralised sample database. The mapping function now becomes $S_C(d_{i,j}) = a_i * S_i(d_{i,j}) + b_i$, and the problem now has focused on training a model to get the most suitable parameters to fit the function and eliminating the error. The error is denoted by $\epsilon = \frac{1}{2}\sum_{j=1}^{n}(f(a,b,S_i(d_{i,j})) - S_C(d_{i,j}))^2$. So the problem becomes $(a_i,b_i) = arg_{a_i,b_i}Min\frac{1}{2}\sum_{j=1}^{n}(f(a,b,S_i(d_{i,j})) - S_C(d_{i,j}))^2$. The regression over all training data can been shown in a matrix representation, which is:

$$\begin{bmatrix} S_i(d_{i,1}) & 1 \\ S_i(d_{i,2}) & 1 \\ \cdots & 1 \\ S_i(d_{i,n}) & 1 \end{bmatrix} * [a_i b_i] = \begin{bmatrix} S_C(d_{i,1}) \\ S_C(d_{i,2}) \\ \cdots \\ S_C(d_{i,n}) \end{bmatrix},$$

notice that a_i and b_i is the parameters for collection i to transform its score to the global score. Then we use some linear algebra methods to derivation an easier way to get the parameters. Denote $\begin{bmatrix} S_i(d_{i,1}) & 1 \\ S_i(d_{i,2}) & 1 \\ \cdots & 1 \\ S_i(d_{i,n}) & 1 \end{bmatrix}$ as X, $[a_i,b_i]$ as W and Y is $\begin{bmatrix} S_C(d_{i,1}) \\ S_C(d_{i,2}) \\ \cdots \\ S_C(d_{i,n}) \end{bmatrix}$. Then we get $X * W = Y$, and so $W = (X^TX)^{-1}(Y^TX)$. Still, the algorithm need the original document score form each collection, which makes it difficult to implement in practise.

[1] proposed an algorithm based on the Bayes's theorem. They built a probabilistic model which is used to estimate the probability of the relevance of a document to a query. Given a document d, $r_i(d)$ is the rank assigned by the ith collection to a query. So the probabilities of a document is relevant and irrelevant given the ranking r_1,r_2,$\cdots r_n$ are $P_{irr} = Pr[rel|r_1,r_2,r_3\cdots,r_n]$ and $P_{irr} = Pr[irr|r_1,r_2,r_3\cdots,r_4]$. Odds(Equation (2.9)) of relevance is widely used as a measurement to compute the possibility.

$$O_{rel} = P_{rel}/P_{irr} \tag{2.9}$$

By Byes rules, we can get

$$\begin{aligned} P_{rel} &= \frac{Pr[r_1, r_2, r_3 \cdots, r_n|rel] \cdot Pr[rel]}{Pr[r_1, r_2, r_3 \cdots, r_n]} \\ & \qquad\qquad and \\ P_{rel} &= \frac{Pr[r_1, r_2, r_3 \cdots, r_n|irr] \cdot Pr[irr]}{Pr[r_1, r_2, r_3 \cdots, r_n]} \end{aligned} \tag{2.10}$$

, so we can compute the odds by

$$\begin{aligned} O_rel &= \frac{Pr[r_1, r_2, r_3 \cdots, r_n|rel] \cdot Pr[rel]}{Pr[r_1, r_2, r_3 \cdots, r_n|irr] \cdot Pr[irr]} \\ &= \frac{\prod_i Pr[r_i|rel] \cdot Pr[rel]}{\prod_i Pr[r_i|irr] \cdot Pr[irr]} \end{aligned} \tag{2.11}$$

taking log on both side and we can get a new formula:

$$\log O_{rel} = \sum_i \log \frac{Pr[r_i]|rel}{Pr[r_i|irr]} + \log \frac{Pr[rel]}{Pr[irr]} \tag{2.12}$$

because $\frac{Pr[rel]}{Pr[irr]}$ is a common term for a document, which takes the same value so we can drop it and get the final formula

$$rel(d) = \sum_i \log \frac{Pr[r_i(d)|rel]}{Pr[r_i(d)|irr]} \tag{2.13}$$

$Pr[r_i(d)|rel]$ represent the probability that a relevant document would be ranked at level r_i by system i, while the $Pr[r_i(d)|irr]$is the probability that an irrelevant document would be ranked at level r_i by system i. The model sounds reasonable in theory but hard to implement in reality. In the paper, the author used the trec_eval to calculate these probabilities by human, which definitely cannot be applied to real world scenario. It may be practical to use training data to get these probabilities in real world. But, as the collections keep changes all the time and the training will take a vast mount of time, its hard to guarantee the efficiency and precision.

Another example of machine learning approach is done by [33]. They proposed two algorithms in [33], the first of which is Modeling relevant document distribution. Consider a scenario where meta-search engine is built. For a query Q, each collection I has n_Q^I numbers of relevant documents. As each component search engine execute a search on the query Q, it returns a list of documents ranged by the similarity in a descending order. And there exists a relation between the number of relevant documents and the size of returned documents, denoted by a distribution $F_Q^I(S)$, where S is the size of the returned documents and F is a function mapping the size of returned documents to the number of relevant documents. For C collections $I_1, I_2, \cdots, I_C$, the aim of the algorithm is to find retrieved document sizes of each collection, denoted by $\lambda_1, \lambda_2, \cdots, \lambda_i$ that can maximum the total number of relevant documents, which is $\sum_{i=1}^{C} F_Q^{I_i}(\lambda_i)$, notice that $\sum_{i=1}^{C} \lambda_i = N$ where N is the total number of retrieved documents. In the algorithm, they use training

data to build the distribution on past queries. When a query comes, they use k nearest neighbours based on text similarity to get the average relevant document distribution. A maximisation procedure is applied to find the suitable λ_i for each collection. Then final result is ranked using a so called C-faced die method. The document at rank k is obtained from the top of the collection with the most amount of remaining documents. The document then is removed of the original list. The final result list is formed in the continuing process. Another algorithm based on the past training data is also proposed by [33], which uses past queries to build a clustering model. In this model, the queries are classified into the same cluster if they have many documents in common. Each query in the cluster is represented by the vector space. And the centroid of each cluster is generated by averaging the vectors of every query in the cluster. Each collection in the cluster is assigned a weight w, where w equals the average number of documents that is retrieved by the queries from the collection. So when the meta-search engine passes a new query, the algorithm calculate the distance between the query and each centroid of the cluster and assign the query to the cluster with the shortest distance. Then the size of documents returned from each collection is calculate in the following method: For a query q, the weights for N collections $(C_1, C_2, \cdots, C_n)$ are $(w_1, w_2, \cdots, w_n)$, And there are M documents to be returned, so the number of documents is returned by collection C_i is $\frac{w_i}{\sum_{j=1}^{C} w_j} * N$. And similarly, the final result is arranged using the C-faced die method.
These two methods are more like algorithms learning collection weights during training. So it may can increase the precision and recall of the retrieved results, but the internal order of documents is not conserved during the C-faced die procedure. And on the other hand, all both models are trying to build a static model that can be applied into different queries in different time, and especially the clustering model is based on the hypotheses that particular collections are focused on some particular topics. [8] proposed an algorithm based on Markov chain, however, this algorithm can only be used in the aggregated meta-search.

2.4.3 Aggregated-Meta-Search-Specific Methods

In the environment of aggregated meta-search, there exists lot of document overlaps among different collections and thus can be regarded as a data fusion problems. There exits many algorithms based on the performances of documents on different collections and use aggregated methods to evaluate its final score. Linear combination methods are typical example of aggregated methods. For example, Fox and Shaw [9] developed 6 methods to normalise the overall similarity across all the individual search systems based on SMART by combining the scores across all the collections. These 6 algorithms(Table 2.1), uses

Table 2.1: Comb Algorithms

Name	Similarity
CombMAX	MAX(Individual Similarities)
CombMIN	MIN(Individual Similarities)
CombSUM	SUM(Individual Similarities)
CombANZ	Number of Nonzero Similarities
CombMNZ	SUM(Individual Similarities)* Number of Nonzero Similarities
CombMED	MED(Individual Similarities)

aggregated methods to re-calculate the score of a document among all the item sets. [31] also proposed a linear combination model named LC Model, the idea is to use linear regression to learn a uniformed weight to represent different weights in different collections. The value of a document is calculated by:

$$\rho(w, x, q) = sin(w)\rho_1(x, q) + cos(w)\rho_2(x, q) \tag{2.14}$$

Apart from the aggregated methods based on the raw score, some algorithms use the rank in different collections to conduct the aggregation methods. For exampe, Borda-fuse [1] was a voting procedure popularly used in Election scenario, and was modified to apply to our metasearch systems. In this model, servers are acting as the role of a voter, and each document is just like the candidate. Each server or retrieval system holds its own preference of ranking on all the documents. And then, the top ranked documents are assigned c points for candidates whose size is c, and c-1 points for the next, etc. Then we sum the score of each document and rank them according to the score. Due to the facts that different server may have different weight on different topics, they assigned weights to each server and then modified the method to a weighted borda-fuse algorithm. This model, works quite well for the system, which has many overlap in the documents. However, in reality, the repositories may vary quite differently. In that case, each document may just appears once in each rank list, which means they may distributed evenly among all the searching systems and has a lot of documents with the same scores. So the model cannot work well in the situation when there is little intersection within documents. And as a fact, their experiments showed that the new algorithm cannot outperform the baseline algorithm, but in most case, can perform better than the best-input system. [20] also proved that Borda-fuse algorithm is not competitive with score-based algorithms. Another algorithm using ranks in different collections to get a new rank is named *Condorcet Fusion* by [17]. The algorithm is developed based on a social choice voting model called Condorcet voting algorithm. The condorcet voting algorithm uses the times a candidate defeat others as a criteria to determine the winner instead of the pure position like [1] did in the Borda-fuse algorithm. The algorithm is quite simple:

Algorithm 1 Simple Majority Runoff

```
1: count = 0
2: for each of the k search systems S_i do
3:     if S_i ranks d1 above d2 then
4:         count++
5:     end if
6:     if S_i ranks d2 above d1 then
7:         count--
8:     end if
9: end for
10: if count > 0 then
11:     rank d1 better than d2
12: else
13:     rank d2 better than d1
14: end if
```

And they also tried to use training data to get a sever weight by the average precision of individual search system. [35] modified the weighted condorcet fusion algorithms using

Algorithm 2 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*

a linear discriminant analysis(LDA) technology to train the weights.

2.5 Evaluation

As a matter of fact, it is a tough task to judge the performance of an information retrieval system. Many meta-searchers have trying to persuade others that their algorithms are better, but none of these algorithms can perform stably in every situation. And others are trying to find good ways to do these judgments on these algorithms. For lots of algorithms developed before, it is a common way to test the performance based on the Test Collection like TREC [32]. However, as discussed by [28], the test collection approach is lack of private data and will not evolve as a real data source. Another approach is to judge the performance based on search log like click-through data. This approach is based on the hypothesis that high-ranked documents tend to have more clicks. However, the hypothesis does not always hold in reality. On the other hand, the search log is not easy to achieve in an un-cooperative environment. Other methods like Human experimentation in the lab as well as naturalistic observation are widely used in the practice and they both have their drawbacks. [28] also implemented a tool to evaluate the performance based on embedded comparison and log analysis.

2.5.1 Evaluation on Binary Relevance

Many of the measurements widely used are based on the binary relevance of documents, namely relevant or non-relevant, retrieved or non-retrieved. Several basic measurements are based on binary relevance, for example, *precision*, *recall*. Documents are assigned values of according to the relevance-retrieval metric .

	Relevant	Non-relevant
Retrieved	True Positives(TP)	False Positives(FP)
Not retrieved	False Negatives(FN)	True Negatives(TN)

Precision is the proportion of relevant documents among all the retrieved documents, which equals $TP/(TP + FP)$ while *Recall*, on the other hand, represent the proportion of relevant documents that is retrieved, so $Recall = TP/(TP + FN)$. So recall can be easily achieved by return all the documents, which can bring an unstable evaluation of the performance. *Accuracy* is another measurement that measures the proportion of documents that are correctly judged by the retrieval system, and $Accuracy = (TP + TN)/(TP + FP + FN + TN)$. [15] gave an argument on the measurement saying that accuracy is not an appropriate measure for information retrieval problem because the accuracy can be easily achieved by treating all documents non-relevant due to the fact that most of the documents are irrelevant to a particular query. Precision and recall represent different aspect of the performance of a retrieval system. In some scenarios, precision is more significant than recall, for example, in a web search engine, users are more interested in the top returned documents, so *precisionatk* is usually applied in such a case ,while in a search tool searching documents across the disks, the recall in more

important. So there exists a trade-off between these two criteria. And, $F - measure$ uses weighted harmonic mean of precision and recall to adjust the balance of the two measurements. Another measurement that is very popular in TREC as a criteria is the $MeanAveragePrecision$(widely know as MAP). The MAP precision is the average precision achieved by different queries at each point a relevant document is retrieved. Saying there is N queries represent different topics, $MAP = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{Q_j}\sum_{i=1}^{Q_j}P(doc_i)$, $P(doc_i)$ is the precision when document i is retrieved for a query j. The MAP is proved to be very stable in practise and is widely used in the judgement of a new algorithm.

2.5.2 Evaluation on Non-binary Relevance

However, the relevance of documents to a query or a topic is not just relevant or irrelevant, some documents are much more than others among the judged relevant documents. Traditional binary evaluation methods are much less sufficient to judge the performance of a retrieval system. Some researchers proposed graded relevance judgement method to tackle the drawbacks of the binary evaluation. One of the most famous grade based measurements is call Discount Cumulated Gain (DCG) and its normalisation Normalised Discount Cumulated Gain (NDCG) proposed in [11,12]. The relevance of documents is not binary in this method and is assigned different value according to its similarity to the topic. Saying we allocate the value (0,1,2,3) to documents that is for $Irrelevant$, $Marginally\ relevant$, $Fairly\ relevant$ and $Highly\ relevant$ separately. In DCG, each document is assigned a new relevance score based on its gain value. The gain value is the sum of the document relevance and its previous gain value. For example, given a relevance vector of a list to a query, $G' =< 3, 2, 3, 0, 0, 1, 2, 2, 3, 0, ... >$, for each element in the vector, the cumulated gain CG is calculated using the following equation:

$$CG[i] = \begin{cases} G[1], & \text{if } i = 1. \\ CG[i-1] + G[i], & \text{otherwise.} \end{cases} \tag{2.15}$$

So, the CG' equals $< 3, 5, 8, 8, 8, 9, 11, 13, 16, ... >$ and then the authors added a discount factor to the cumulated gain to reduce the gain of documents that is ranked marginally. The usual approach is divide the original CG by the log of the document rank. So the Discount Cumulated Gain, namely DCG now becomes:

$$DCG[i] = \begin{cases} CG[i], & \text{if } i < b. \\ DCG[i-1] + \frac{G[i]}{\log_b i}, & \text{if } i >= b. \end{cases} \tag{2.16}$$

so the DCG turns out to be $< 3, 5, 6.89, 6.89, 7.28, 7.99, 8.66, 9.61... >$ when the base b is set to 2. There exists another form of the equation,

$$DCG[i] = DCG[i] + \frac{2^{G[i]} - 1}{\log_2(1+i)} \tag{2.17}$$

To evaluate a performance of a document, documents are judged by assessors in advance, for example, following is the distribution of the relevance of documents to a particular query:

Relevance	Number of documents
3	a
2	b
1	c
0	d

, for an ideal information retrieval system, the result returned should form a list in decrease order of document relevance, so the ideal gain vector (IG) should be like this: $IG =< 3, 3, 3, ..., 2, 2, 2, ..., 1, 1, 1, ..., 0, 0, 0, 0, .. >$. According to the ideal gain vector, we can get the value of ideal cumulated gain vector(ICG) as well as ideal discount cumulated gain vector(IDCG). For a new retrieval system that we want to measure, we calculate the relative score of the DCG to the IDCG, which is named NDCG as proposed and get by dividing DCG by IDCG

$$NDCG = \frac{DCG}{IDCG} \tag{2.18}$$

Notice that NDCG is always in the range of 0 to 1. The NDCG method can also be used in binary evaluation, in which the relevance of a document and a query is just 1 for relevant and 0 for irrelevant.

Apart from the NDCG methods, others tried to compare the system rank with the accurate rank of documents. [16] proposed such a method to calculate the distance between *the actual relevance of documents (UREs)* and *the estimates by the Information Retrieval System (SREs)*. The average distance measure(ADM) is calculate using:

$$ADM_q = 1 - \frac{\sum_{d \in D} |SRE_q(d_i) - URE_q(d_i)|}{|D|}. \tag{2.19}$$

$SRE_q(d_i)$ is the actual relevance of a document to a query while $URE_q(d_i)$ is the relevance assigned by the information retrieval system. And D is the number of documents in all the document collection. So this bring a problem in implementing the measurement, it is not possible for assessors to judgement each document for each query. However, it is a practical approach to use the retrieved documents, however, it has not been tested if the accuracy is preserved.

2.5.3 Evaluation on incomplete judgement

Although many of the measurement is said to have a robust evidence to judge a information retrieval system, however, it is usually the case that the judgement of documents is not complete and sometimes imperfect. [3] studied the problem of incomplete information and developed a measurement called *bpref*. The *bpref* method evaluate the chance of a situation when irrelevant documents are ranked ahead of relevant documents in a retrieval system. So we get the value of *bpref* by:

$$bpref = \frac{1}{R} \sum_{r} (1 - \frac{|n \ ranked \ higher \ than \ r|}{R}) \tag{2.20}$$

In the equation, R is the number of relevant document for a specific topic and n is a member of the first R judged irrelevant documents retrieved. To get avoid of the danger that there is very small number of documents, they proposed a $bpref - 10$ method, which

enlarge the number of judged irrelevant documents to 10+R, and now is

$$bpref - 10 = \frac{1}{R}\sum_{r}(1 - \frac{|n\ ranked\ higher\ than\ r|}{10 + R}) \tag{2.21}$$

The experiments showed that the measurement is quite stable even when there is 50 percent or less information is provided. And the other experiment in [3] also showed that even some documents disappeared from the collection, the result is not strongly affected.

Actually other existing measurements can also work fine in the incomplete judgements situation, [2] did experiment on the robustness of three relevance measures, namely bred, NDCG and infAP with incomplete judgements. The result showed that the NDCG, which is modified to binary evaluation outperformed others in such an environment. The experiments also showed that binary NDCG is a little bit better than non-binary NDCG in preserving the system ranking in incomplete judgements, However, this has not been further experimented and non-binary NDCG may still be a good choice in the situation when there is limited amount of incompleteness.

Chapter 3

Experiment

Although many algorithms have been proposed to tackle the meta-search merging problem, not all of them have been tested in real world environments. Some information or statistics that needed by an algorithm is not available in some situations. So the feasibilities of these algorithms remain unknown. On the other hand, many algorithms proposed with an ambition to promote the performance of the system on TREC test collection. So even if an algorithm can outperform other algorithms within TREC among any topic, it is not sufficient to say the algorithm is good enough to solve the problem. In this section, we develop an evaluation platform based on a real world system to test the feasibility of different algorithms as well as the performance of these algorithms.

3.1 Test Collection and Queries

Many of previous works in meta-search were tested based on some standard test collections like TREC and GOV2. Typically, these test collections are composed by *document collections*, *information needs* expressed by the queries as well as the *relevance score* of the queries and documents. And the evaluation procedures judge the ranking results following a particular algorithm compared to relevance score. However, the test collection approach is very different from a real world situation in some sense. First of all, the documents exist in the collections are static and do not change over time, which is quite different from the situation when we search documents in a web search engine. And documents format in real world varies, and information provided by different search engines are different from each other. For example, when search information of a person, the ANU contact web returns a list of contacts with names and contact numbers while the ANU website returns the title of related documents as well as a summary of the documents.So the retrieved information is quite biased. The other information provided by the test collection is the relevance score assessed by the experts. TREC comes with a set of natural language statements of information needs, named topics [34]. These topics that carefully defined by an assessor who would also evaluate the relevance of the topic and documents. Topics are expressed by query terms that are carefully constructed to make sure that it is clear. However, the information needs may be ambiguous in a real world situation due to the fact users may not be able to express their needs in accurate queries, which may bring inaccuracy results to the relevance judgement [28]. And what is more significant issue is that many of previous merging algorithms have not been tested in a real world environment. Some of the algorithms will perform variously in a different situation or even cannot work in such a situation. So in this paper, we built a platform based on a real world environment.

Collection	Description	Location
ANU Search	The main entry of the university, containing information of people, news as well as some information in research	http://search.anu.edu.au/
ANU Contact Search	The site contains information of contacts as well as positions of a stuff	http://www.anu.edu.au/dirs
ANU Map	The site provides information of locations on campus	http://campusmap.anu.edu.au/
ANU Library	The library catalogue of ANU	http://anulib.anu.edu.au/
ANU Study	Course information of ANU	https://studyat.anu.edu.au
ANU Research	The site provide researcher, research project and research publication information of ANU, the three types of resources are actually separated into different collections, so it has been built into three different collections	https://researchers.anu.edu.au/
ANU Dspace	Document information related to ANU	https://digitalcollections.anu.edu.au/

Table 3.1: Internal Collections

3.1.1 Experiment Environment

To construct a real world environment for evaluating different merging algorithms, we built an evaluation test bed based on the websites of the Australian National University. There exists many sites of different departments of the Australian National University. However, some websites are totally distributed from others and there exists no access point to the resources from other sites. And the university do not provide an integrated interface for users to get access to all the resources.

Document Source

We picked 7 sites that are distributed from each other, each of which has their own documents repository and has a search interface for users to access the resources.

Apart from these collections existing inside the university, we also import other two collection which can regard as a special case of meta-search. We add two social media site that is related to ANU as shown in Table 3.2.These two social media site, on the other hand, gave us an opportunity to examine the performance of different merging algorithms in on a real time environment. Also, the contents in these site are not text only, many other formats of information exist in both internal and external collections. However, we claim that all the algorithms that we applied is based on the text information.

Document Overlap

Although the document collections are distributed from each other, there still exists some documents overlap between these document repositories. For example, there exists limited

Collection	Description	Location
ANU Youtube Channel	The official Youtube Channel of ANU	http://www.youtube.com/user/ANUchannel
ANU Twitter Channel	The official Twitter Channel of ANU	https://twitter.com/ANUmedia

Table 3.2: External Collections

amount of documents overlap between ANU website and ANU study at website. Also, the results in ANU website collection may contain some documents retrieved form ANU research. And some other types of document overlap also exists, documents are accessible to many collections but formatted differently. For example, the ANU website may contain videos from youtube site together with some text description. However, the amount of document overlap is very limited and we ignored the little overlap in our experiments.

3.1.2 Building test collection

Previous evaluation of new algorithms are mostly based on a large standard test collection. However, in our case, no test collection is provided. Although there exist some methods that can compare performance of different information retrieval system without a test collection [28]. However, these methods are basic based on the judgement that a user given to say if a system is better than the other, which may not provide sufficient evidence to convince. And as the collections changes over time, we cannot ask assessors to judge documents at a time and conduct the evaluation experiment at the other time. And as the judgement procedure can really take a lot of time, many issues can arise during the procedure, some documents may be deleted while some other pages can be created during the time, especially in the case of social media system.

So the solution is creating a test collection ourselves. There exist large volumes of documents in each collections, so it would be a large collection of documents if we run a full document collection. And also, due the limitation of bandwidth and disk space, it is not feasible to put all documents into the test collection. And on the other hand, the document judgement procedure will take too much time. Later in the experiment, what we care is the ranking of documents that is already retrieved by each collection when search a particular query or topic. So the necessary and useful documents are those document retrieved by the meta-search system on those queries defined in advance. Nevertheless, the size of document that returned by each collection distributes in a quite large range because some of the collections may contain limited resources, this is not like the case when doing collection selection when more relevant collection may contain more numbers of retrieved documents. The difference here is that the total volume of collection is quite different from each, which may bring a biased effect on the ranking. On the other hand, previous study [10] has also shown that users are tend to care about the documents ranking in top and ignore the low-ranked documents, we decide to pick the top 10 documents that is ranked by the original search engines.

A complete test collection is composed by document collection, topic list as well as the judgement of the relevance between documents and the topics. The document collection is initialized when we pass the topic list to the meta-search system. The broker of the system will download the original web page locally as it has been retrieved. And later on, we will use the document judgement system to assess the relevance score between the documents and the topics.

Original users may not be able to express their information needs in a clear and unambiguous way in a form of search term and some topics may extremely biased to some particular collection, this may be useful when evaluating the performance of a collection selection algorithm, however, the focus is on the merging algorithms, regardless of the effect of collection selection problem. So we decided to picked up the topics and defined the query terms by ourselves to reduce the ambiguity and to make sure it can retrieve information from most of the collections. The topics are mostly focused on information of people, organizations and many other subjects that potentially related to ANU. Table 3.3 gives a detailed image of all the categories that we used in the experiment. We used 44 queries in total and were separated into 6 categories.

Table 3.3: Topic categories

Category	Number of queries
People	13
Links with (Orgnisation)	8
Material for a press release	7
Regional Focus	5
Potential for a research collaboration	7
Department details	4
Total	44

Table 3.1.2 and 3.5 describes the document distribution that is retrieved by different component search engines. The figures show that the majority of the queries can retrieve documents from most of the collections. And on the other hand, some collections, such as ANU contact search and ANU Map search contains very limited amount of documents while other collections such as ANU Web search has hundreds times of documents than others. As a result, we use the top 10 documents from each collection and get them ranked in the broker. Table 3.1.2 is the distribution of documents that is actually used in the merging algorithms. Notice the number of retrieved documents in Table 3.6 is an approximation.

3.1.3 Platform

We built a platform to execute meta-search on the collections and run different algorithms on the meta-search engine. The experiment platform is made up of two core system, one of which is the meta-search system, the other is evaluation system. The meta search system is a meta search engine built in Java which provides a broker that separates queries to all the sub-search-engines and merge the returned results.

The main components of the broker are *search interface adapter* which analyse each result page of individual search engine, and *merger* which fuses the results in a particular algorithm and returns the result. Figure 3.1 is an architecture of the meta-search system. We provide a search engine user interface to input the queries. The web will submit the query to the broker when a query search action is taken. The broker read the query and pass the it to the adapter factory. The adapter reads each query and pass it to its parser, the parser analysis the result page of each component search engine and put the result into a result pool. The merger will read result lists from the result pool and execute merging using a particular algorithm and then return the merged list.

The other parts is *Evaluation System*, which is used to evaluate the performance of

Table 3.4: Document distribution in different collection

	0	1	2	3	4	5	6	7	8	9	10
A. D. Hope	0	275	14	5003	0	0	0	1027	2000	23280	140
ACT Health	0	713	106	5071	10	0	0	1314	770	4660	130
Adult education	0	834	0	4521	3	0	0	658	480	810	10
afghanistan	0	1147	13	5158	10	0	20	124	30	120	10
Alistair Rendell	1	0	0	786	1	0	0	5	20	90	10
anglo-australian telescope	3	7	0	138	0	0	0	35	30	50	10
ANU Library	0	37	124	5430	10	0	1	1052	3300	48330	740
ARC	14	510	531	5178	10	176	4	410	290	100	600
Australian sculpture	0	236	21	1991	9	0	0	35	1360	5540	610
big data	0	72	9	5006	9	0	2	668	460	990	10
biodiversity	0	32000	68	5058	9	0	3	205	120	390	10
Brian Schmidt	1	23	0	631	10	0	12	13	120	730	20
Canada	0	15385	12	5510	10	0	0	767	160	170	10
China	3	32000	108	7434	10	3	21	1621	250	930	20
chinese politics	0	1600	46	5009	10	0	0	444	510	2190	10
clear energy	0	51	12	5000	6	0	1	768	310	910	10
climate change	0	1797	200	7278	10	0	20	768	600	1380	40
CSIRO	3	851	36	5022	10	0	2	214	120	160	10
Desmond Ball	1	118	1	769	3	0	0	39	60	280	10
EU	5	1122	452	5035	10	26	2	419	40	50	10
fenner school	1	15	101	3855	10	6	0	62	700	320	10
gender equality	0	382	0	2497	7	0	0	596	210	470	10
H. C. Coombs	0	86	4	3335	10	0	0	128	1380	10640	40
High performance computing	0	393	32	3500	2	0	0	947	890	3360	30
Ian Young	2	120	1	4157	10	0	10	236	330	1190	40
indigenous languages	0	92	42	5027	10	0	1	442	320	1170	20
islamic radicalization	0	10	0	66	0	0	0	101	40	70	10
John C Harsanyi	0	11	0	28	0	0	0	4	1340	8950	80
John Eccles	0	55	0	300	0	0	0	13	450	2540	70
Judith Wright	0	123	0	545	0	1	0	14	70	260	10
Material Science	0	554	2788	5014	7	0	0	1511	1380	6430	20
murray darling basin	0	149	7	1651	4	0	2	54	170	470	10
national botanical gardens	0	8	0	494	0	0	0	32	920	1310	130
NHMRC	0	73	2	1490	10	0	0	81	50	10	130
NICTA	0	1	6	4498	2	0	0	28	50	10	10
Peter Doherty	0	13	0	261	2	0	1	12	310	1710	70
quantum physics	0	1782	22	5000	10	0	0	340	410	6030	30
Rolf Zinkernagel	0	3	0	98	0	0	0	1	20	10	10
rsacs	0	0	0	1	0	0	0	1	0	0	0
Sir Howard Florey	0	4	0	255	0	0	0	7	110	280	10
Sir Mark Oliphant	0	9	0	192	0	0	0	5	160	1080	30
South Korea	0	1959	35	5004	10	0	2	0	560	1080	20
Student Admin	10	1	425	1796	2	11	2	34	410	70	10
teaching mathematic for young young child and the old	0	0	47676	165	0	0	0	13	780	2420	10
The middle East	0	4898	149	5134	10	1	8	698	2230	24310	310

Table 3.5: Collection mapping

Collection ID	Collection Name
0	ANU Contact Search
1	ANU library catalogue
2	ANU Studyat
3	ANU Web Search
4	ANU YOUTUBE Channel
5	ANU MAP Search
6	ANU Twitter Channel
7	DSPACE
8	ANU Researchers
9	ANU Research Publication
10	Research Projects

different algorithms and provide statistical as well graphic form of performance. The essential function of the evaluation system is as follows:

- Initial a test collection using the meta-search engine.
- Provide a relevance judgement interface to assessors and record the graded relevance score on a query and documents.
- Executing result merging of different algorithms and record the fused list of each query.
- Compare the results with user judgements.

The evaluation system use the facilities of the meta-search to initialize the test collection, namely the document pages as well as statistical information of the results. The system also provide a relevance judgement system to assessor the judge the relevance scores between queries and documents. A component named algorithm evaluator is the core function of the system, which execute different merging algorithms on the broker and store the result lists. The algorithm evaluator will output varies performance information based on different criteria.

3.2 Methodologies

We examined many of the algorithms discussed in the literature review and fit in some of the algorithms that is suitable for this environment.

3.2.1 Round-Robin

Round-Robin was used as a common method in many of the previous works in meta-search. The advantage of round-robin is that it preserve the original rank of documents in a collection. However, the relative order of a document of a collection to a document in another collection is quite random. For example, in our environment, we have more than 10 document collections in total. If we have a collection that contains many comprehensive relevant documents but ranked at end of each round-robin cycle, the most relevant document will rank low in such a case. The other disadvantage of round-robin is that it

Table 3.6: Documents used in merger

	0	1	2	3	4	5	6	7	8	9	10
A. D. Hope	0	10	10	9	0	0	0	10	10	10	10
ACT Health	0	10	10	10	10	0	0	10	10	10	10
Adult education	0	10	0	10	3	0	0	10	10	10	2
afghanistan	0	10	10	9	10	0	10	10	10	10	0
Alistair Rendell	1	0	0	9	1	0	0	5	10	10	4
anglo-australian telescope	1	7	0	10	0	0	0	10	10	10	0
ANU Library	0	10	10	10	10	0	1	8	10	10	10
ARC	10	10	10	10	10	1	4	10	10	10	10
Australian sculpture	0	10	10	9	9	0	0	10	10	10	10
big data	0	10	9	10	9	0	2	10	10	10	9
biodiversity	0	10	10	10	9	0	3	10	10	10	5
Brian Schmidt	1	10	0	9	10	0	10	10	10	10	10
Canada	0	10	10	10	10	0	0	10	10	10	1
China	1	10	10	10	10	0	10	10	10	10	10
chinese politics	0	10	10	10	10	0	0	10	10	10	10
clear energy	0	10	10	10	6	0	0	10	10	10	9
climate change	0	10	10	10	10	0	10	10	10	10	10
CSIRO	1	10	10	10	10	0	2	10	10	10	0
Desmond Ball	1	10	1	8	3	0	0	10	10	10	5
EU	5	10	10	10	10	1	2	10	10	10	0
fenner school	1	10	10	10	10	1	0	10	10	10	3
gender equality	0	10	0	10	7	0	0	9	10	10	6
H. C. Coombs	0	10	4	10	10	0	0	8	10	10	10
High performance computing	0	10	10	10	2	0	0	10	10	10	10
Ian Young	1	10	1	10	10	0	10	10	9	10	10
indigenous languages	0	10	10	10	10	0	1	10	10	10	10
islamic radicalization	0	10	0	10	0	0	0	10	10	10	1
John C Harsanyi	0	10	0	10	0	0	0	4	10	10	10
John Eccles	0	10	0	10	0	0	0	10	10	10	10
Judith Wright	0	10	0	9	0	1	0	10	10	10	6
Material Science	0	10	10	10	7	0	0	10	10	10	10
murray darling basin	0	10	7	8	4	0	2	10	10	10	9
national botanical gardens	0	8	0	10	0	0	0	10	10	10	10
NHMRC	0	10	2	8	10	0	0	10	10	3	10
NICTA	0	0	6	10	2	0	0	10	10	0	0
Peter Doherty	0	10	0	10	2	0	1	10	10	10	10
quantum physics	0	10	10	10	10	0	0	10	9	10	10
Rolf Zinkernagel	0	3	0	10	0	0	0	1	10	10	0
rsacs	0	0	0	1	0	0	0	0	0	0	0
Sir Howard Florey	0	4	0	9	0	0	0	6	10	10	7
Sir Mark Oliphant	0	9	0	10	0	0	0	5	10	10	10
South Korea	0	10	10	10	10	0	2	0	10	9	10
Student Admin	10	0	10	10	2	1	2	10	10	10	0
teaching mathematic for young young child and the old	0	0	5	10	0	0	0	10	10	10	9
The middle East	0	10	10	10	10	1	8	10	10	10	10

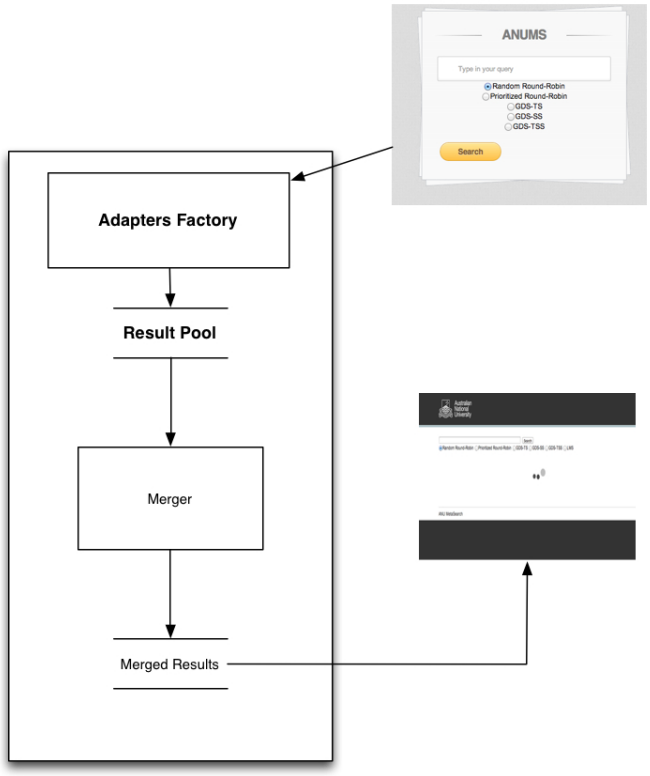


Figure 3.1: The architecture of meta-search

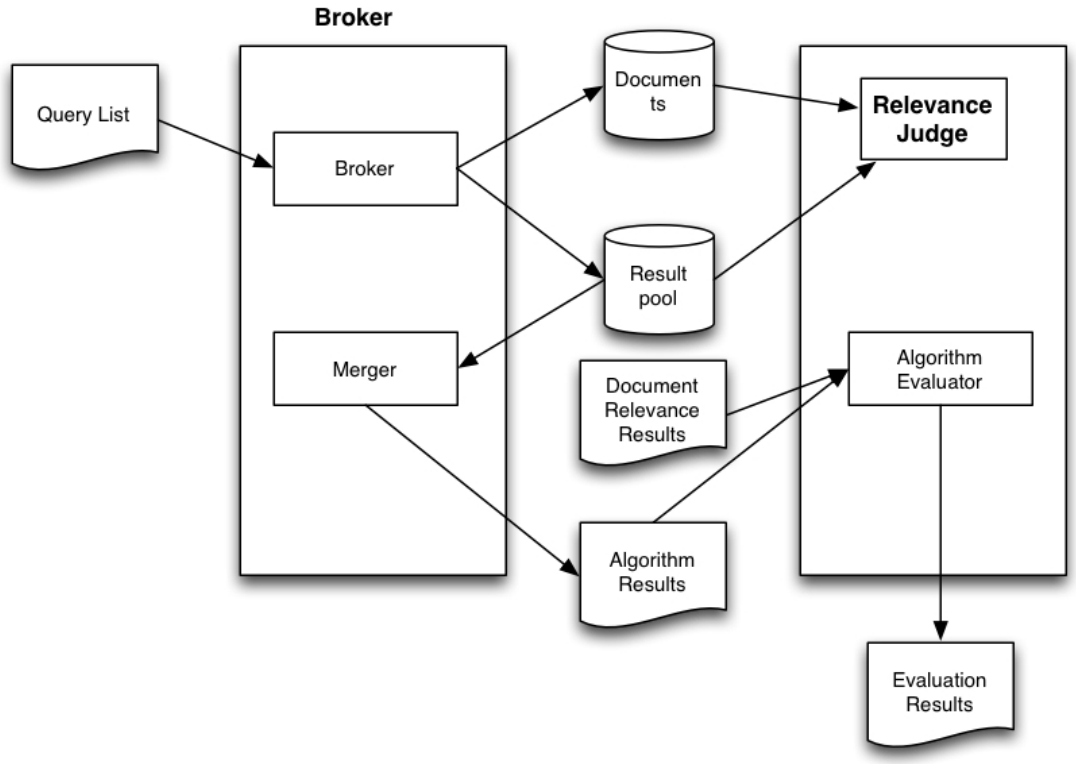


Figure 3.2: The architecture of evaluation system

holds a hypothesis that documents are distributed evenly in each collection, which is not accurate in a real world situation. We implemented a round-robin method in the experiments and use it as the baseline of all the algorithms. The order of documents in each cycle is determined by collections that is randomly ordered.

3.2.2 Prioritized Round-Robin

One drawback of round-robin method is as we talked above that they rank documents randomly in each cycle. To reduce the random effect of round-robin, we try to rank collections in advance according to the relevance of the collection and the query. Collections is biased on a query because a particular collection may focus on some particular topics. There are many algorithms in previous study doing collection selection calculate the collection weight according to some provided statistical information of the collection. However, in our environment, there is limited information on the collection. The only statistical information on collection when doing a search are the length of documents that retrieved by that collection as well as the result page.The length of documents that retrieved by different collections is potentially an indication of collection significance. We used the LMS method to get a collection score and rank collection by the collection score. We have already stored the length of retrieved documents when initializing the test collection. The algorithm first calculate the LMS scoring using the document length before doing the round-robin.

The result page reveals some information on the collection. We develop an algorithm using the average document similarity as a collection score. For documents $d_1, d_2, d_3, ..., d_n$ in collection C_i, the average similarity W_i is calculated as follows:

$$W_i = \frac{\sum_{1}^{N} s_i}{N} \tag{3.1}$$

, where s_i denotes the similarity of d_i and the query.

The same as the LMS based prioritized Round-robin, the merger get each collection's score at first and then rank the collections in the order of collection score. The merged list will be outputted in a round-robin mechanical following the collection order.

3.2.3 Generic document scoring

The score estimation methods proposed by [19] is a more practical approach in such a situation that there exist limited information on the original score assigned by each collection as well as collection statistics. The method used different textual fields that is available to public users as a representation of document topics. This fields are easy and cheap to obtain. And in most of case, these filed are summaries of what the documents talking about, which is a good indication of document focus. The score of a document is calculated by the generic document scoring function of Equation (2.4) using the frequency of query terms that appears in the filed. In this experiment, we used four methods derived from the scoring functions. The first one that we used is the title field, the method is named *SM-TS*, namely the document title scoring. As it is stated by [19], the document title returned by the search engine is more accurate and reliable. And most of search engines return a document summary to display some key contents of the retrieved documents. So we also used *SM-SS*, namely document summary scoring, by getting the document score

using the generic function score. Furthermore, We utilized the combining method using both document title and document summary. The title is sometime too short and query terms may not be contained in the title. Also, in many cases, the query terms and some terms in the field may have the similar semantic information but expressed by different words. So the score for many relevant documents may remains to be zero in such a case. So we use the score of summary field as a supplement when the title score is zero. Algorithm 3 illustrates the scoring function we used in the TSS method. Notice that rank score is applied when the two other scoring function failed. To keep the significance of filed core, we divide the rank score by 10,000 to make sure that documents that has a filed score will not be ranked higher than those document without a zero score. The rank score is simply calculated by $RS = (10 - R + 1)/10$, in which RS denotes the rank score and R denotes the rank of the document in the collection.

Algorithm 3 TSS Scoring algorithm

```
TTS = TS
if TTS == 0 then
    TTS = SS
    if TTS == 0 then
        TTS = RS
    end if
end if
```

Also [19] suggested that the combining method can be implemented by giving different weights to title score and summary score as Equation (3.2). The weight k and $1 - k$ are given to title score and summary score respectively and both in the range from 0 to 1. We also add a rank score to get rid of zero score. The constant k shows the importance of title score and summary score. We'll test the value in our experiments later on.

$$DTSS = k * TS + (1 - k) * SS \tag{3.2}$$

3.2.4 Length-based collection score

As discussed above, the number of documents retrieved by each collection may be a good indication of collection relevance. We tried to use the method [18] proposed, and calculate the collection weight by Equation (2.7) and Equation (2.8). The method is just a way to give the collection weight. It has been used in the round-robin method as a collection score. In this method, however, we combine the weight by the estimated score obtained by the DTSS method to show if the performance has been improved. So we now get a score S_{ij} for a document d_i in collection C_j by Equation

$$S_{ij} = w_j * DTTS_i \tag{3.3}$$

, in which w_j denotes the collection weight of collection C_i.

3.2.5 Locally re-index

A more reliable way to conduct the result merger is to build another index system locally and conduct retrieve based on the index system. So when a search action is taken, the system download the documents documents retrieved by all the component search engines and build an index file based on the documents. The system then will calculate the

document score based on the scoring function and return the documents in a list ordered by document score. In our experiments, all the documents retrieved by all the search engines have been downloaded to the local disk as we build the test collection, so in our locally re-index algorithm, when doing a search on a particular query, the merger retrieves the results from locally stored result pool and reads corresponding documents from test collection into memory. We used the open source Java library Lucene to index and score the documents. As the Lucene library returns just the relevant documents judged by itself, we put other documents at the end of the list.

Locally re-index is said to be accurate and reliable in its nature. However, the process of downloading and indexing is quite time consuming. It might not be a practical approach in a real world situation due to large volumes of documents and the limitation of bandwidth as well as users' patience. However, we implemented the algorithm in our experiment as a benchmark. So if an algorithm can outperform, or be as good as the locally re-index algorithm, it is quite acceptable.

3.2.6 Multiple Weight

As we can see from what we have discussed above, the actual rank of documents is influenced by many factories. But however, it is not clearly for us at the moment that how these factors effect the final ranking. So in this experiment, we try to build a model that take all these factors into consideration and find out the weights for each of these factors. The information can be achieved from a typical uncooperative federal search environment is as follows:

- The similarity between the document title of a document d_i and the query, denoted by TS.
- The similarity between the document summary of a document d_i and the query, denoted by SS_i.
- The number of documents that are retrieved by a collection C_i, denoted by N_i
- The average similarity of the documents in a collection C_i, denoted by $aveS_i$
- The rank score given by the collection for a query R_i

The first two information belongs text field that can be used to estimate the document similarity. A previous research has been done by [19] using a generic document function to estimate the document similarity, as we discussed. And the number of document retrieved as well as the average document similarity reveals the significance of a collection to a particular topic. The last factor, rank score, demonstrate the original ranking of a document in a collection. The original rank of a document in a collection is ignored by most of the score based algorithms, As a matter of fact, although rank score of a document does not show exactly the score of a document assigned by the component search engine, it shows the order of document scores. And whether we should preserve the original order of document ranking remains to be research question. Algorithms like round-robin can outperform other algorithms in some cases is because the original rankings of component search engines are quote good and even. So as round-robin preserve the goodness of the original ranking, it outputs a good ranking. We built a linear regression model utilizing these factors:

$$S_i = w_1 * LMS_i + w_2 * aveS_i + w_3 * Eds_i + w_4 * Rs_i \tag{3.4}$$

, where S_i denotes the score for document d_i, and w_1, w_2,w_3,w_4 denotes the weights assigned to these factors respectively. The document length score, denoted by LMS_i is calculated using the length based scoring function proposed by [18]. Eds_i denotes the estimated document similarity using generic document scoring function and Rs_i is the rank score of document d_i in its original collection.

The remain problem became how to estimate theses weights. In SSL method [26], they proposed a similar approach by building such a regression model to map original document score to a document score in the sampling collection. However, in our experiment, the sampling data is not practical to achieve. Instead, we build a linear regression model which map these four factors to the relevance score. So the intended linear function becomes:

$$f(w, x_i) = w_1 * LMS_i + w_2 * aveS_i + w_3 * Eds_i + w_4 * Rs_i \tag{3.5}$$

And to solve this problem, we used a least square method in linear regression. The aim of the model is to find a set of parameters, which minimise the square difference between the function and the relevance score.

$$w = arg\min_{w} \sum_{i} (f(w, x_i) - rel_i)^2 \tag{3.6}$$

The ideal relevance score is already be assessed by assessors, we'll use a n-fold cross validation method to test the performance of this function. We separate the 44 queries that we experimented into 4 folds, each contains 11 queries. And We pick three of the fold as the training data and keep the remain as the testing data. The testing is comprised of those factors as well as the relevance score assessed.

3.2.7 Unimplemented Algorithms

Not all of the algorithms that we discussed in literature review were implemented due to the limitation of information available. First of all, as we discussed in literature review, some algorithms can only be used in Aggregated meta-search. A fundamental condition of implementing this kind of algorithm is that there is sufficient amount of document overlap among different collections. However, there exist just little or no document overlap in most of the result returned by different collections in our experiments environment. The environment that we use in this experiment is a typical model of non-aggregated meta-search. So those aggregated merging algorithms were not considered in the experiments.

CORI is well-known distributed information retrieval resolution in collection selection. A merging algorithm is also come along with it. The merging algorithm of CORI calculates collection weights by normalizing the collection scores that are obtained in the collection selection step. The original CORI system is built upon the INQUERY system, which constructs an inference network based on probability. And the building of the INQUERY system requires collaboration of the collections. Even if we do not use INQUERY system to get collection score using CORI algorithm, some statistical information on the collections are needed.

For a term t_k, we calculate the belief that term t_k can be observed by the collection, namely $p(t_k|c_i)$ by:

$$T = \frac{df}{df + 50 + 150 \cdot cw/avg\_cw} \tag{3.7}$$

$$I = \frac{\log(\frac{C+0.5}{cf})}{\log(C + 1.0)} \tag{3.8}$$

$$p(t_k|c_i) = b + (1 - b) \cdot T \cdot I \tag{3.9}$$

df is the number of documents in a collection C_i contains a term, cw is the number of terms in C_i, avg_cw is the average cw of all the collections. C is the number of collections, cf is the number of collections that contain the term. b is a constant, usually set to 0.4. So to get a collection score for a query, we need to get the p for each term appears in the query and aggregate them in some methods. And then we use a minmax method to normalize the collection scores to get a collection weight, the collection weight is then combined to other raw score functions. So, as far as we can see, the utilization of CORI algorithm requires basic term size and document frequency of a collection, which is not available in our case.

As we discussed above, machine learning approaches can be promising methods in solving meta-search merging problems. For example, the SSL method using regression method to map the document score in the original collection into an integrated index centralised sampling collection. Issues arise when sampling the document collections. First of all, the size of each collection is unknown, so it becomes a tricky problem how many documents should we keep to form the collection sample. And on the hand other, the documents in each collection are not static due to the characteristic of web collection, it is hard to guarantee the accuracy of the sample. The sample size should large enough so that there exists at least three document overlap in each query result, in which case, the sampling process would be very time and capacity consuming.

There exists other machine learning methods which build clustering or document distribution based on previous queries. So to execute these methods, we need training data to construct the model. Due to the limitation of usage of the platform, it is hard to achieve enough training data. As a matter of fact, users tend to use short and inaccurate query terms to express their information needs, which means that the correlation between queries is very small. So if we put them into a machine learning model, the similarities between new queries and past queries are too tiny.

3.3 Evaluation Criteria

We chose several criteria to demonstrate the performance of the algorithms, both binary and non-binary. Although as we discussed about, binary relevance is not sufficient to prove the efficient of an algorithm, it is still a common criteria that would be used in the evaluation of information retrieval system. Because all the algorithms run on the fixed result pool that is initialized in advance. So the recall as well as precision are the same for all the algorithms. However, different algorithm rank documents in different order, so the precision as well as recall of different algorithms at position k varies. So the basic criteria that we use to measure the performance is *Precision at k*.

$$Prec@10 = \frac{|Top\ 10\ retrived\ Documents \cap Relevant\ Documents|}{10} \tag{3.10}$$

As we preserved the top 10 documents from each collection. It seems like that many of document would be relevant to the query because they are already ranked high by their ranking system. However, for two different documents that are relevant to a topic, the degree of relevance varies. Some documents may mention part of the topic while others may exactly focus on the topic. So if the a system can rank more relevant document higher than less relevant documents, it is better than others. Due to the unbalanced relevance score of different documents, we applied a graded judgement method, NDCG, which use cumulated score with respect to the ideal ranking. We generate the NDCG result for each algorithm of a query and graphic representation of the NDCG.

3.4 Relevance judgement

The document judgement procedure gives a score to the relevance of documents and the query. The relevance score then will be used in the evaluation of merging algorithms as a benchmark or ideal ranking. Conventional test collections invite experts from a specific domain to assess the relevance score, and typically, the relevance of documents with respect to a query is judged as binary score, namely *relevant* and *irrelevant*. However, as we discussed before, binary relevance method is not sufficient to prove that an information retrieval system is better than other. So, in this experiment, we used non-binary relevance judgement method. The relevance of documents and queries were assigned to 5 levels, namely *irrelevant*, *On-topic but useless*,*Somewhat useful*,*Useful*,*Comprehensively useful* and the criteria of relevance is given as follows:

Irrelevant is the score given to documents that are totally irrelevant to the topic.

On-topic but useless is the score given to documents that are related to the topic but is talking on some other aspects of the area.

Somewhat useful is the score given to documents that are relevant to the topic and the topics that the queries are intended to retrieved are mentioned in the documents.

Useful is the score given to documents that most, or at least some part of the documents are talking on the intended information.

Comprehensively useful is the score given to the documents that are talking exactly on the topic.

According to these 5 level of relevance, we set the relevance score in the range from 0, which is irrelevant, to 4, which is given to documents that is comprehensive.

We set up a web-based platform for assessors to conduct the assessing process. The platform provides a solid user interface, in which a particular query and its potential relevant documents are provided. Assessor can pick up a query can assess each document using the platform.

3.5 Experimental Procedure

We separated our experiment into several stages, the first of which is initializing the test collection and experiment platform. In this stage, we initialized the test collection by passing the query list to the broker. The broker retrieved the result documents from each collection and store the original documents as well as the result pool locally. The next step is conducting relevance judgement. We set up a web-based judgement system on-line. We also set-up a relational database (Mysql) bounding to the website to store the query and its corresponding documents. We invited 20 volunteers of different background from

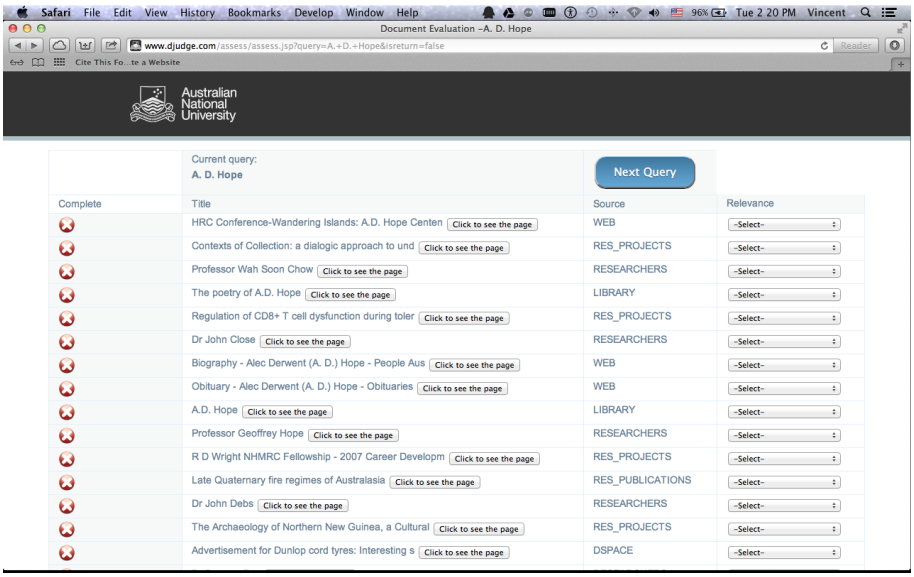


Figure 3.3: User Interface of Document judgement system

the Australian National University, who are also potential users of the system. Although people from different backgrounds may have biased on information needs and may generate different relevance results, we do not record personal information in our experiment for several reasons. First of all, to secure the individual privacy, it is more safe to get rid of these personal information and is a policy of the university to protect personal information in a research project. And on the other hand, the research is focused on the merging algorithm regardless of the influence of personal feedback. And also, the topics that we pick up are very common and are not ambiguous to any person in any background, which is intend to eliminate the personal effect.

We set up a web-based document judgement system on-line, the system is developed using JSP technique and is deployed in an Apache Tomcat sever on a Linux system. The system provide a user interface as the platform of the experiment. To keep the completeness of the document score, we added a feature for user that they can continue assessing the query after a period of time. Assessors were trained to use the application as well as how to score the relevance properly before the execution of the experiments.

After all the queries have been assessed, we gather the document judgement result to local xml file. The final stage is to evaluate the performance of different algorithm according to the relevance score. In this stage, we run different algorithms on the query list and output the ranking results of these algorithms. Again, we store the result of these algorithms in xml file query by query. We pass the algorithm result as well as our user judgement document to an evaluation script that we wrote based on different criteria and get results of their performance.

Chapter 4

Result Analysis

In this section, we analyse the experiment results that we obtained and analyse the underlying reasons for the observation. We'll discuss the overall performance of all the methods that we applied and compare some particular methods with others. Also, we'll discuss how these algorithms would change as environment changes.

4.1 Overall performance

The performance of 10 methods that we used in the experiment various as a whole, even for a particular method, its performance may change for different query. However, the overall performance across 44 queries that we have assessed in the experiment of some methods show a superiority than others, while others cannot outperform others in most cases.We applied both binary and non-binary methods to give an image of how these algorithms performs and how they differentiate from each other.

The binary evaluation method, precision at 10(prec@10) separate documents into two categories, relevant and irrelevant documents. In the experiment procedure, we developed a judgement platform following graded judgement principle, which resulted a list of document score that is non-binary. These score ranges from 0 to 4, from 0, which says the document is totally irrelevant to the topic to score 4, which means the document is comprehensively relevant to the topic. So a straightforward thinking is to regard documents score in the range from 1 to 4 as relevant documents and others as irrelevant documents. So then we get a distribution of the prec@10 of all methods running on all the 44 queries, as it is displayed in the Figure 4.1 The distribution of prec@10 demonstrates that of all these algorithms, there seems not be huge differences between other except the three round-robin methods show disadvantages over others. A obvious phenomenon that we can see from the figure is that mean precisions are all very high than a mean precision we can achieve in a traditional information retrieval system. The reason underlying these results is that different from a conventional search engine. The meta-search system conduct ranking based on the documents that are retrieved by their original search engines, so they are all originally retreated as relevant documents. So that means the overall proportion of relevant documents is larger than a traditional search engine. And also, the prec@10 only cares about the percentage of relevant documents among the top 10 retrieved documents, regardless of the degree of relevance and the rank of relevant documents. The little difference between methods make it tricky to find pattern inside these algorithms, especially a set of similar algorithms, we did a paired t-test to those algorithms to the same group, and we can see the results in Table 4.1 and Table 4.2. As we can see from p-value, only the one paired of algorithms, namely GDS_TS and LMS, rejected the Null Hypothesis,

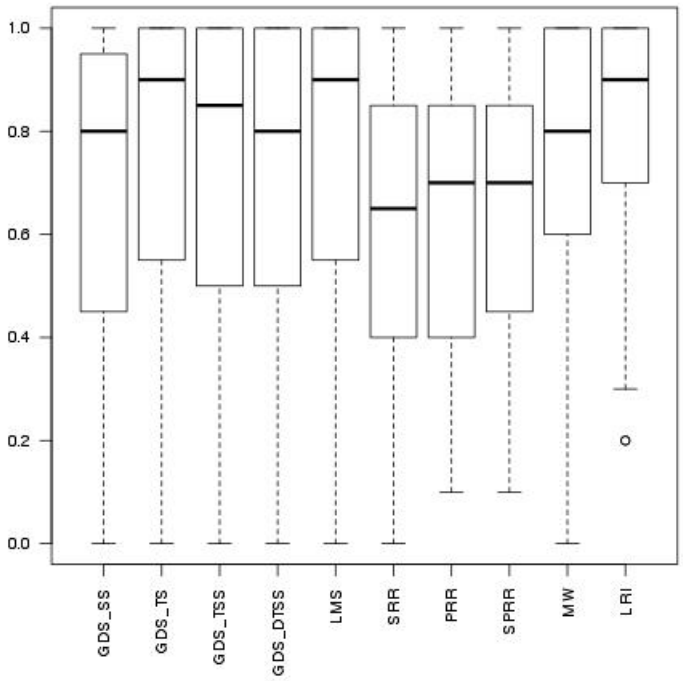


Figure 4.1: precision@10

	GDS_SS	GDS_TS	GDS_TSS	LMS
GDS_SS	-	0.06315	0.2001	0.03484
GDS_TS		-	0.1297	0.5189
GDS_TSS			-	0.08983
LMS				-

Table 4.1: p-value on t-test over GDS-XX methods and LMS, on confidence level of 95%, freedom degree=43

which means there is no sufficient confidence to say a algorithm is better than others.

We can also increase the standard of relevance, which means we restrict the criteria for relevant documents, and thus set the threshold of relevance, if we set threshold to 1, only the documents whose score and higher than 1 can be regarded as relevant documents and vice verse.

So the four graphics in Figure 4.2 illustrates the changes when we increase the threshold from 0 to 4. And one thing can be found is that the whole distribution of all methods has become lower as threshold is increases, however, the relatively distributions of algorithms against others are not significantly differentiates.

The other approach that we used to evaluate these different methods is a non-binary measurement method, NDCG@10, which calculate the gain score cumulatively. So ideally, NDCG@10 can reveal the differences between the resulted rank and the ideal rank according to the relevance scores assigned by the assessors.

Figure 4.3 is the distribution of the NDCG@10 score of all methods among all the 44 queries. Clearly, it makes a distinguishable distribution across different algorithms. Again we did a paired T-test on this algorithms, see Table 4.3

	SRR	PRR	SPRR
SRR	-	0.6748	0.3071
PRR		-	0.2528
SPRR			-

Table 4.2: p-value on t-test over Round-Robin methods and LMS, on confidence level of 95%, freedom degree=43

	GDS_TSS	GDS_TS	GDS_SS	GDS_DTSS	LMS	SRR	PRR	SPRR	MW	LRI
GDS_TSS	—	NH	2.8873	NH	NH	4.7466	4.4734	3.7726	NH	NH
GDS_TS		—	3.2307	NH	NH	4.699	4.4148	3.8922	NH	NH
GDS_SS			—	-3.0256	-3.1842	NH	NH	NH	NH	-3.4549
GDS_DTSS				—	NH	4.8382	4.5063	3.8844	NH	NH
LMS					—	4.7543	4.4829	3.9019	NH	NH
SRR						—	NH	-3.51	-3.7406	-7.2133
PRR							—	-2.6575	-3.7114	-6.6232
SPRR								—	-2.8994	-5.9037
MW									—	NH
LRI										—

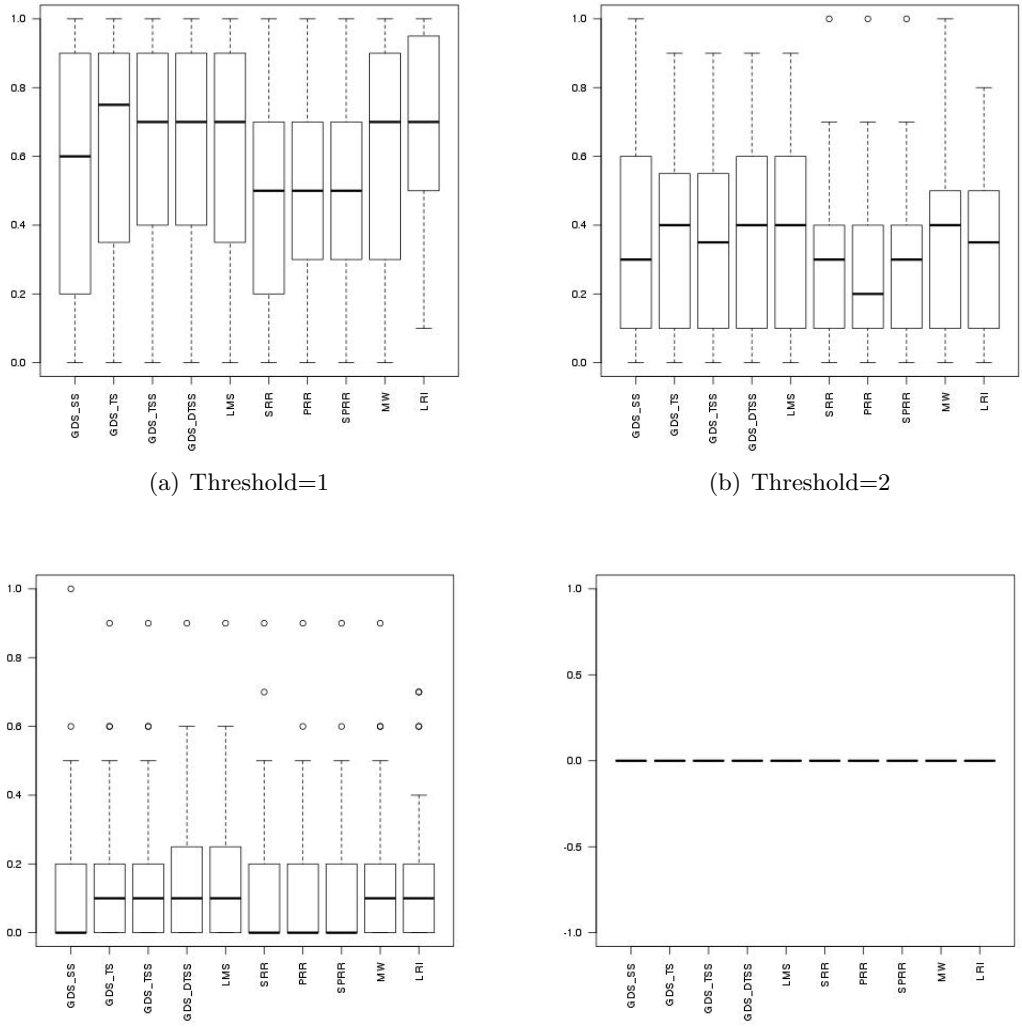
Table 4.3: T-test results for NDCG@10 results for all the methods,NH stands for Null Hypothesis, means t-test has not rejected the Null Hypothesis, the t test conducted from row to column, confidence level=95%, df=43

So now there exists significantly differences among different algorithms. The three Round Robin algorithms are the worst algorithms among all those algorithms been tested while the Local Re-index method and the Generic Document Scoring method based on Document Title (GDS_TS) seems to be two outstanding algorithms among others in this environment. how to use table 4.4

The LRI (Local Re-index) method fetch the original web pages that were retrieved and treat them as a document collection. The method then use an traditional indexing approach on the document collection as what we usually do in a centralised indexing information system. The documents were then ranked based on the scores that were calculated by the similarity between the query and the document text. So the performance of LRI methods is totally depended on the performance of the indexing method as well as the accuracy of the scoring method. If a distributed information retrieved system is no worse than an index centralised information retrieval system, it is considered as an

	GDS_TSS	GDS_TS	GDS_SS	GDS_DTSS	LMS	SRR	PRR	SPRR	MW	LRI
GDS_TSS	—	p=0.2849241, d=0.164888	p=0.2849241, d=0.164888	p=0.6570079, d=0.3138873	p=0.3382643, d=0.1880336	p=2.310635e-05, d=-0.3719031	p=5.558954e-05, d=-0.3399154	p=0.0004892817, d=-0.2528957	p=0.1742379, d=0.1083292	p=0 d=0
GDS_TS		—	p=0.00236984, d=-0.1806192	p=0.3820464, d=0.2059694	p=0.9073363, d=0.4548726	p=2.695421e-05, d=-0.3664063	p=6.6984e-05, d=-0.3329079	p=0.0003407972, d=-0.2682688	p=0.1011739, d=0.05668385	p=0 d=0
GDS_SS			—	p=0.004178282, d=-0.1519879	p=0.002699161, d=-0.1741874	p=0.1019622, d=0.05733364	p=0.2573317, d=0.1521309	p=0.6824074, d=0.3246332	p=0.07800444, d=0.03465159	p=0 d=-0.2
GDS_DTSS				—	p=0.4223028, d=0.2219145	p=1.716752e-05, d=-0.3824303	p=5.005215e-05, d=-0.3438145	p=0.0003489795, d=-0.267274	p=0.1188498, d=0.07107689	p=0 d=0
LMS					—	p=2.254158e-05, d=-0.372781	p=5.393764e-05, d=-0.3410388	p=0.0003309153, d=-0.2695003	p=0.09512011, d=0.05130356	p=0 d=0
SRR						—	p=0.1272298, d=0.077384	p=0.001065238, d=-0.2184404	p=0.0005385612, d=-0.2487635	p=0 d=-0.09, 0.6
PRR							—	p=0.01100594, d=-0.09882246	p=0.000587745, d=-0.2449703	p=4 d=-0.08, 0.5
SPRR								—	p=0.005865978, d=-0.1340298	p=5 d=-0.07, 0.4
MW									—	p=0 d=0
LRI										—

Table 4.4: effect size



(a) Threshold=1

(b) Threshold=2

(c) Threshold=3

(d) Threshold=4

Figure 4.2: Precision at different thresholds

acceptable method. So the LRI is regarded as baseline method for all other methods as a top performance regardless of its efficiency in the aspect of time and space. It is also quite obvious that the LRI methods outperforms others in most of the cases as we can see in both precison@10 and NDCG@10.

We use a pure round-robin method in the experiment and two other prioritized round-robin methods to give different to different collections to rank documents in an order of collection importance. The figures shown that the overall performance of all the round-robin method performs the worst among all other algorithms.

4.2 Round-Robin

Round-robin was the most fundamental mechanical that has been widely used in the early meta-search system. As it has been stated, round-robin method is based on the hypothesis that documents are distributed evenly in different collections, which means that if all the collections have an equally distribution of documents according to their relevance to

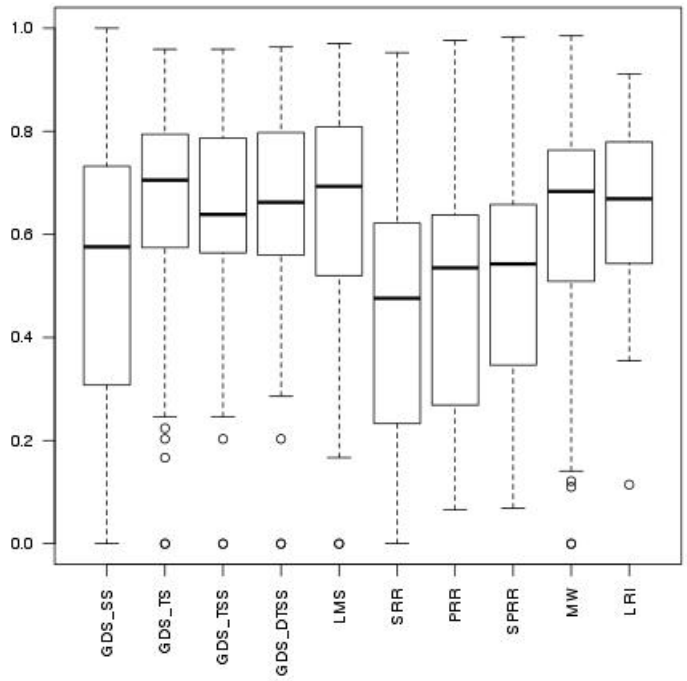


Figure 4.3: NDCG@10 of all algorithms

the query, the method can gain an excellent result. However, due to the fact that the collection may have different weights on a particular topic in reality, and the situation when documents are evenly distributed in collections seldom happens. As a result, round-robin method can not outperform others in most cases, and since the rank of documents in each cycle is random, the performance of the methods various in different runs. We run the round-robin algorithm five times and compare them with the LRI method as well as two other prioritized round-robin methods. Also, to test the optimized performance of round-robin, we designed an optimized round-robin method($OPRR$), which use the final relevance score judged by assessors as a criteria of collection weight and run a collection weight based prioritized round-robin. The results are displayed in Figure 4.2. As been stated, round-robin performs quite randomly in each run. The underlying reason is that documents were ranking randomly in round-robin cycle. But, however, a phenomenon that we can observe from the Figure 4.2 is that although these round-robin runs perform differently, the variation is not significant. The reason is because that we documents that lying in the top 10 list are pretty much the same in each run, and in that case, the difference is the relative order of these documents.

Two other prioritized round-robin methods are also evaluated in the experiments. The original intention of both two algorithms is to rank collections in an order of relevance at the first place and then rank document in order of the collection score in each round-robin cycle. The PPR use the document length, namely the number of document retrieved, as an indication of collection relevance. The second prioritized round-robin method get the collection weight using the average similarity of the retrieved documents. This method is similar to a previous algorithm named *TopSRR* [14], which use title and summary fields of top k documents to calculate the collection score. The original TopSSR method initialized

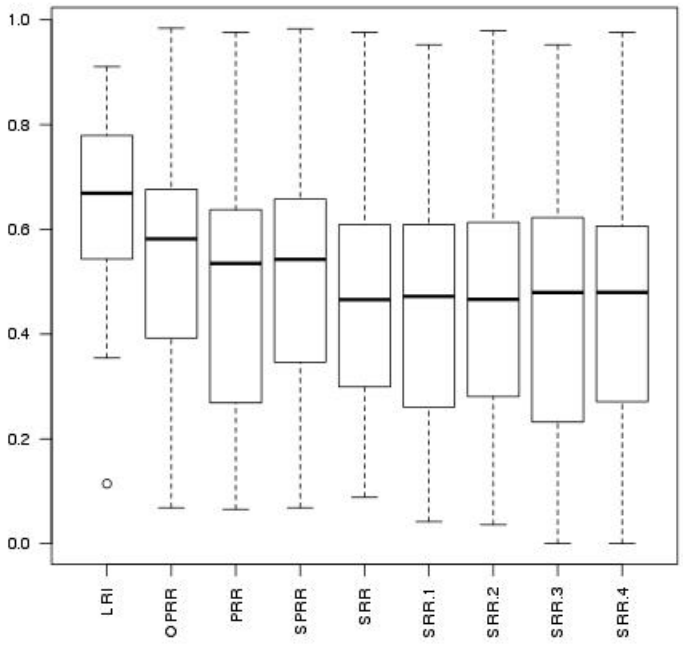


Figure 4.4: NDCG@10 of Simple Round Robin on 5 runs

	PRR	SPRR
OPRR	5.0855	2.9621

Table 4.5: t-value of t-test over OPRR to other prioritized Round-Robin, on confidence level of 95%, freedom degree=43

two vectors TV_j and SV_j by merging all the title and summary fields together respectively and then use Equation to get the collection score.

$$S_j = c * Similarity(Q, TV_j) + (1 - c) * (Q, SV_j) \tag{4.1}$$

In our method, we applied an average function on the similarity of all the top k documents that is retrieved. We then use the aggregated document similarity as the collection score in the round-robin method. As we can observe from Figure **??**, both two algorithms have promote the median of NDCG@10 to some degree and make it more stable. We get a mean NDCG@10 by average the NDCG@10 of a query across all methods. Of all the 44 queries, the PRR method can perform better than mean NDCG@10 in 10 queries while SPRR can do better in 13 queries. That may suggest SPRR may get a more reasonable collection score than PRR.

The paired T-test (Table 4.5)suggest that there exists distinguishable difference between the optimized round-robin. So that means neither of the two prioritized round-robin methods are sufficient to be optimized, and that remains a problem of collection weight, we'll discuss the topic in the next section.

Even if we optimized the round-robin by providing an ideal collection score, the method is still not comparable to most of other merging algorithms, as shown in Figure 4.5. The

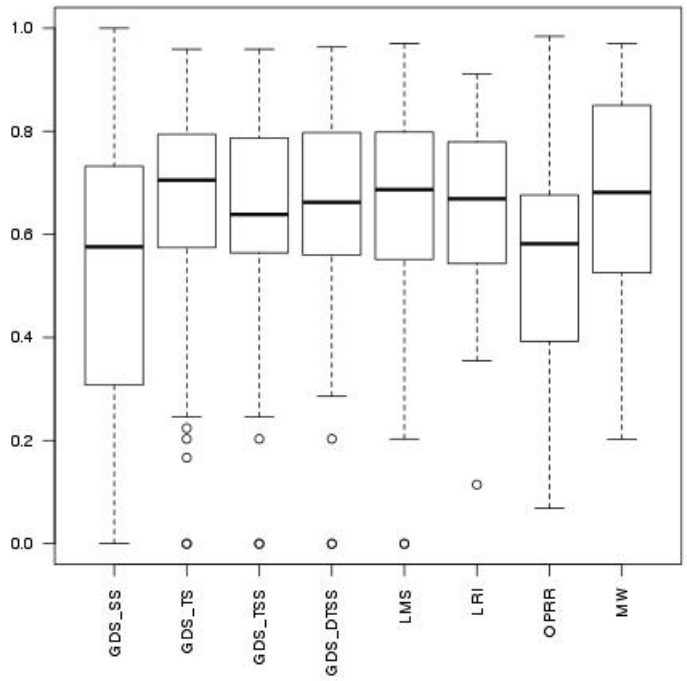


Figure 4.5: NDCG@10 of OPRR with respect to other algorithms

reason is round-robin methods itself is not a suitable method, which rank documents from different collections evenly in a result. Saying we have 10 collections, one of which contains many relevant documents that would be ranked at top 10 in an ideal ranking and none of the remaining collections contain relevant documents. Using a round-robin method, even we rank the first document of the most relevant collection at the top of the final rank, all the other remaining nine documents in the list are irrelevant. As a result, the accuracy of the documents ranking compared to the ideal ranking is quite low.

4.3 Evidence of Collection Weight

Collection weight is always taken into consideration when doing a meta-search engine. The collection weight reveals the significance of a collection with respect to a topic. In an uncooperative environment, there exists limited information on the collection. So what the algorithms performance if we got an ideal or so called optimized collection score. We design a method that get the collection score by cumulate the relevance score that is assessed by assessors. And then we combined the score with the GDS_DTSS method, we name it OGDS, means the optimized GDS. Figure 4.6 illustrates the importance of collection score, if we can add an ideal collection score to the algorithm, it can be improved in a large scale. The OGDS method, has improved the performance of the original GDS and even better than the LRI method. We examine how much difference from the weighted algorithms to the original by applying a Paired T-test(Table 4.6).

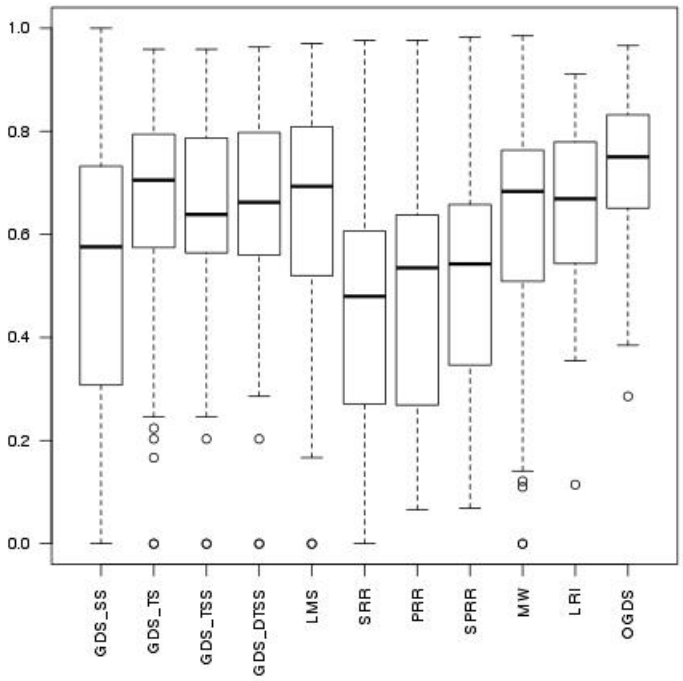


Figure 4.6: NDCG@10 of OGDS with respect to other algorithms

Table 4.6: T-test result for SGDS

	GDS_DTSS	GDS_TS	GDS_SS	LMS	LRI
OGDS	4.4879	3.6568	5.6473	3.4213	2.966

Table 4.7: t-value of NDCG@10 of OGDS tested to other algorithms, confidence level 95%, df=43

4.3.1 Document Length

In our experiments, the limited information can be obtained from the result page. Number of documents retrieved from the search engine is usually accessible in the result page. In LMS, the document length, has been considered as the factor that represent the collection weight. We combined the collection weight get by the LMS with other score based algorithms, here we used the GDS_DTSS method. As we can see from Figure 4.3, the LMS has increased the median of all the NDCG@10 across all the queries. But the total distribution has not been changed so much. Also, in the last section, we used LMS to get a collection weight in the prioritized algorithm PRR, and get the conclusion that LMS method cannot guarantee the optimized round-robin. Again, we used the method in previous section to get an optimized collection weight and combine it with the GDS_DTSS, and compare it with the LMS algorithm.

The LMS is much less than optimized as it is shown in the t-test in the round-robin experiment. The sizes of document collections themselves are various from each other in certain degree, so the number of documents retrieved by these collections are also influenced by the total number of documents as well as the precision of those search engines. For example, some search engines with a bad retrieval algorithm may return

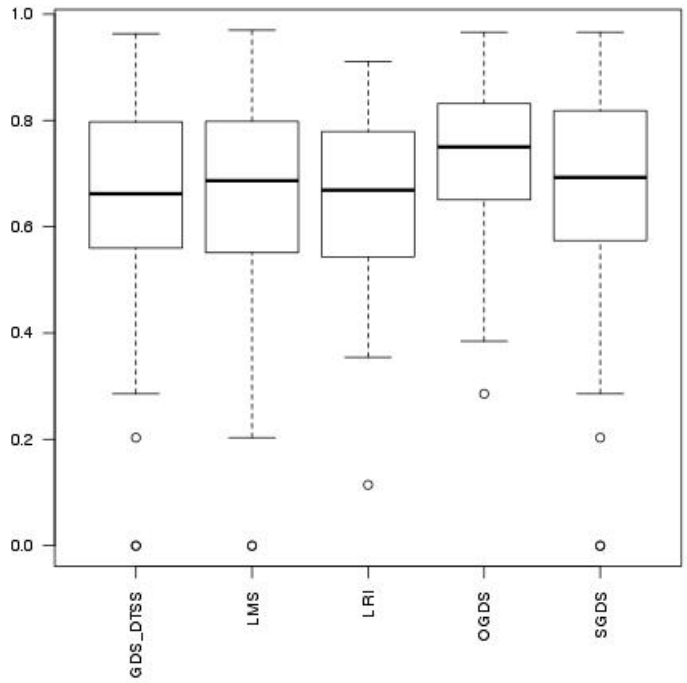


Figure 4.7: NDCG@10 of SGDS with respect to other algorithms

many inaccurate and irrelevant documents while some other search engines with high precision return a limited number of documents that is very relevant to the topic. So document length would not be a sufficient evidence of collection significance.

4.3.2 Average document similarity

Another method of getting the collection is by average the similarity of the text fields of all documents in the first result page with the topic. These text fields, in some way, is a sample of the document collection, which can reveal some information on the particular collection.

Compared to the document length based collection scoring method, LMS, the performance of average document similarity method is more close to the optimized collection weight.

4.4 Value of document fields

The GDS algorithms estimate the document collection score by a general scoring function using the text field provided by the result page. We used two fields, title and summary in our experiment, as text fields. We used four methods to estimate the document score,

- **GDS_TS** Scoring function uses the title field only.
- **GDS_SS** Scoring function uses the summary field only

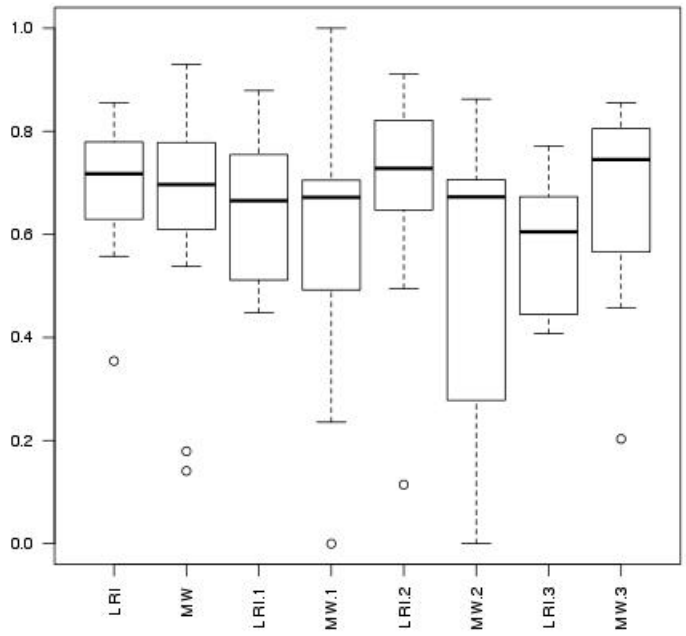


Figure 4.8: 4-fold cross validation

- **GDS_TSS** Scoring function uses the title field as the first priority, if score is not usable, use the summary field
- **GDS_DTTS** Scoring function uses a combination of title field as well as summary field

The performance described by Figure 4.3 shows that GDS_SS performs the worst among all other general scoring functions, while GDS_TS performs the best. The phenomenon demonstrates the fact that the title may be more valuable than summary in representing the main topic of a document. Of all the 44 queries that we evaluated, GDS_SS method can perform better than an average performance in 16 queries while GDS_TS can outperform others in most of the queries, making 33 of all.

4.5 Parameter setting of multi-weight method

To this point, we have taken four factors into our algorithms to get the final comparable document scores to rank the documents. The final method is named multi-weight because we took all these factors into consideration when calculating the document score. As we described in the experiment section, we set 4 weights to these four factors respectively and summarize the weighted factors. We test the performance of this method by a 4-fold cross validation.

Figure 4.8 is a result graphic of 4-fold cross validation compared with the LRI method. Each fold contains an equal number of randomly picked queries. As we can see the method varies among different folds. Sometimes it can outperform LRI in a great degree, for example, MW.3 with respect to the LRI method. However, it can perform very badly, for

example, in MW.2. The purpose of the model is to set weights for those factors, so in a real world system, is the data size too small to learn?

Chapter 5

Conclusion and Future Work

Appendix A

System Architecture

Code

This appendix describes the architecture that we built for the experiment. The entire system was built in two platform as described in chapter 3, the broker together with the document judgement system.

On top level of the broker, we provide a function named *fetchResult*, which takes two parameters, the query and the intended method. The function is the main controller of the broker, which then separate the query to each search engine and get the result pool as shown in List B.1

Listing B.1: Code for getting ranking result

```
1  public ArrayList<Result> fetchResult(String Query, int method) {
2                  AdapterFactory factory = new AdapterFactory();
3                  ResultTable results = factory.executeQuery(Query);
4                  Merger merger = null;
5                  switch (method) {
6                  // Random Round-Robin
7                  case ALGORITHM.SRR:
8                          merger = new SimpleRRMerger();
9                          break;
10                 case ALGORITHM.PRR:
11                         merger = new PRRMerger();
12                         break;
13                 case ALGORITHM.GDS_TS:
14                         merger = new GdsMerger(Query, TS);
15                         break;
16                 case ALGORITHM.GDS_SS:
17                         merger = new GdsMerger(Query, SS);
18                         break;
19
20                 ....
21
22                 default:
23                         break;
24                 }
25                 return merger.executeMerging(results, factory.
                       ServerTable);
26
27         }
```

The query is set to the *AdapterFactory* in the first place. *AdapterFactory* is a factory of all the adapters for all the search engines. Due to the limitation of bandwidth, we adapted the adapters into a multi-thread environment. Query is sent to the *ParserFactory* first to parse each webpage that is retrieved by the component search engines to DOM Trees. *We*

also downloaded these pages for later use. The parsed documents are stored in a hashmap mapped to their original source. And then in the next step, which is the core function of the adapter factory, is to allocate those parsed documents to their own adapters.

Listing B.2: Code for AdapterFactory

```
1  public void allocateAdapters(String query) {
2                  this.query=query;
3                  query = StringFormat.toURL(query);
4                  ParserFactory parserFactory = new ParserFactory(query);
5                  parserFactory.initialDocuments();
6                  //reset DocumentSet
7                  DocumentSet.reset();
8                  HashMap<Integer, Document> documents = parserFactory
9                                  .getDocumentCollection();
10                 CountDownLatch countDownLatch = new CountDownLatch(
                       documents.size());
11                 for (Map.Entry<Integer, Document> entry : documents.
                       entrySet()) {
12                         int server = entry.getKey();
13                         Document document=entry.getValue();
14                         Thread t=null;
15                         switch (server) {
16                         case ServerSource.CONTACT:
17                                 t = new Thread(new ContactAdapter(
                                       countDownLatch,document, results,
                                       ServerTable,"http://www.anu.edu.au/
                                       dirs",server));
18                                 break;
19                         case ServerSource.DSPACE:
20                                 t = new Thread(new DspaceAdapter(
                                       countDownLatch,document,results,
                                       ServerTable,"https://
                                       digitalcollections.anu.edu.au",
                                       server));
21                                 break;
22                         ...
23
24                         default:
25                                 break;
26                         }
27                         t.start();
28                 }
29                 try {
30                         countDownLatch.await();
31                 } catch (InterruptedException e) {
32                         // TODO Auto-generated catch block
33                         e.printStackTrace();
34                 }
35
36         }
```

The aim of those adapters is to analyse the result pages, fetch information of each result documents from that page. There already exists some methods to detect search engine interface and get the document entry, however the focus of this thesis is to evaluate the performance of merging algorithms. As an alternative, we used the Xpath to analyse the result pages instead. All the adapters are inherited from the base class *Adapter*, the *Adapter* class implements the Runnable class for the purpose of multi-thread. And the

Adapter class also has a abstract function named *query*, which analyse the result page and store them into a list of results. List B.3 is an example of the adapter for the anu library catalogue. As we can see, the main purpose of the function is to analyse the result page and parse each document list in the result page, fetch the information of each document and store it in a particular result object.

Listing B.3: Code for the adapter of library catalogue

```
1  public class LibraryCatalogAdapter extends Adapter {
2
3          public LibraryCatalogAdapter(CountDownLatch countDownLatch,
4                          Document document, ResultTable results,
5                          HashMap<Integer, Server> serverTable, String
                               hostUrl, int source) {
6                  super(countDownLatch, document, results, serverTable,
                       hostUrl, source);
7                  // TODO Auto-generated constructor stub
8          }
9
10         public RankList query() {
11                 // TODO Auto-generated method stub
12                 if (document == null)
13                         return null;
14                 // get the size of retrieved documents
15                 Pattern pattern = Pattern.compile("\\d+\\s+(results|
                       result) found");
16                 Node body;
17                 Matcher matcher = null;
18                 try {
19                         body = (Node) xpath.evaluate("//BODY", document,
20                                         XPathConstants.NODE);
21                         matcher = pattern.matcher(body.getTextContent())
                               ;
22                 } catch (XPathExpressionException e1) {
23                         // TODO Auto-generated catch block
24                         e1.printStackTrace();
25                 }
26                 int size = 0;
27                 while (matcher.find()) {
28                         String match = matcher.group();
29                         size = Integer.parseInt(match.substring(0, match
                               .indexOf("result"))
30                                         .trim());
31                 }
32                 Server server = new Server();
33                 server.setServer(source);
34                 server.setResult_size(size);
35                 sTable.put(source, server);
36                 RankList ranklist = new RankList();
37                 try {
38                         NodeList nodeList =
39                                 (NodeList) xpath.evaluate("//TD[@class
                                       =\"briefCitRow\"]", document,
                                       XPathConstants.NODESET);
40                         int length = nodeList.getLength();
41                         // no more than 10 results returned
42                         int resultsize = 0;
43                         for (int i = 0; i < length; i++) {
```

```
44                                  Element ROW = (Element) nodeList.item(i)
                                        ;
45                                  LibcataResult result = new LibcataResult
                                        ();
46                                  Node Title =
47                                          (Node) xpath.evaluate(
48                                                  "TABLE//SPAN[@class=\"
                                                      briefcitTitle\"]/A",
                                                       ROW,
49                                                  XPathConstants.NODE);
50                                  if(Title==null)
51                                          continue;
52                                  String title = Title.getTextContent().
                                        trim();
53                                  result.setTitle(title);
54                                  String Link = hostUrl
55                                                  + ((Element) Title).
                                                      getAttribute("href")
                                                      .trim();
56                                  Node Summary = (Node) xpath.evaluate(
57                                                  "TABLE//SPAN[@class=\"
                                                      briefcitTitle\"]",
                                                      ROW,
58                                                  XPathConstants.NODE);
59                                  String textarea = Summary.getTextContent
                                        ().trim();
60                                  String summary = textarea.substring(
                                        title.length() + 1).trim();
61                                  if (!DocumentSet.contains(Link)) {
62                                          result.setLink(Link);
63                                          result.setSummary(summary);
64                                          result.setSource(source);
65                                          result.setDsumary();
66                                          ranklist.addResult(result);
67                                          resultsize++;
68                                  }
69                                  DocumentSet.AddDocument(Link);
70                                  if (resultsize >= 10)
71                                          break;
72                          }
73
74                  }
75                  catch (XPathExpressionException e) {
76                          // TODO Auto-generated catch block
77                          e.printStackTrace();
78                  }
79                  return ranklist;
80          }
81  }
```

All the result classes are inherited from the base class which is named *Result*(List **??**), common properties are stored in the base class. And we implement the Comparable interface for comparing two results according to their scores.

Listing B.4: Code of class Result label

```
1  package com.results;
2
3  import org.w3c.dom.Document;
```

```
4
5  import com.util.DocumentSet;
6
7  public class Result implements Comparable<Result>{
8
9          protected String Title;
10         protected String Link;
11         protected int Source;
12         protected String imgLink="";
13         protected String displaySummary;
14         protected double score;
15         protected String docID;
16         protected int rank;
17
18         .....
19
20         @Override
21         public int compareTo(Result o) {
22                 // TODO Auto-generated method stub
23                 if(o.getScore()>this.getScore())
24                         return 1;
25                 else if(o.getScore()==this.getScore())
26                         return 0;
27                 else
28                         return -1;
29         }
30 }
```

The last step in the broker controller is merging the results of different collections. The result tables are passed to the *Merger*, the controller will choose a particular merger according to the method wanted. All the mergers are implemented from the interface *Merger* which defines a function called *executingMerging*, which take the result tables and output the merged list.

Bibliography

[1] Javed a. Aslam and Mark Montague. Models for metasearch. *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '01*, pages 276–284, 2001.

[2] Tanuja Bompada, Chi-Chao Chang, John Chen, Ravi Kumar, and Rajesh Shenoy. On the robustness of relevance measures with incomplete judgments. *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '07*, page 359, 2007.

[3] Chris Buckley and Ellen M. Voorhees. Retrieval evaluation with incomplete information. *Proceedings of the 27th annual international conference on Research and development in information retrieval - SIGIR '04*, page 25, 2004.

[4] James P Callan, Zhihong Lu, and W Bruce Croft. Searching Distributed Collections With Inference Networks. *ACM SIGIR-95*, pages 21–28, 1995.

[5] Jamie Callan. Distributed information retrieval. In *Advances in Information Retrieval*, pages 127–150. 2000.

[6] JP Callan, WB Croft, and SM Harding. The INQUERY retrieval system. *Database and Expert Systems ...*, 1992.

[7] PB Danzig, Jongsuk Ahn, John Noll, and Katia Obraczka. Distributed Indexing : A Scalable Mechanism for Distributed Information Retrieval. *... in information retrieval*, pages 220–229, 1991.

[8] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the Web. *Proceedings of the tenth international conference on World Wide Web - WWW '01*, pages 613–622, 2001.

[9] Edward A Fox and Joseph A Shaw. Combination of Multiple Searches. In D K Harman, editor, *The 2nd Text Retrieval Conference TREC2 NIST SP 500215*, volume 500-215 of *NIST Special Publication*, pages 243–252. NIST, National Institute for Standards and Technology, NIST Special Publication 500215, 1994.

[10] Zhiwei Guan and Edward Cutrell. An eye tracking study of the effect of target rank on web search. *Proceedings of the SIGCHI conference on Human ...*, pages 417–420, 2007.

[11] K Järvelin and J Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (...*, 20(4):422–446, 2002.

[12] Jaana Kekäläinen. Binary and graded relevance in IR evaluationsComparison of the effects on ranking of IR systems. *Information Processing & Management*, 41(5):1019–1033, September 2005.

[13] Joon Ho Lee. Analyses of multiple evidence combination. *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval SIGIR 97*, 31(SI):267–276, 1997.

[14] Yiyao Lu, Weiyi Meng, Liangcai Shu, Clement Yu, and King-lup Liu. Evaluation of Result Merging Strategies for Metasearch Engines. *Search*, 3806:53–66, 2005.

[15] CD Manning, P Raghavan, and H Schütze. Introduction to Information Retrieval. In *Introduction to Information Retrieval*, number c, chapter 8, pages 151–175. 2008.

[16] Stefano Mizzaro. A NEW MEASURE OF RETRIEVAL EFFECTIVENESS(OR : WHAT S WRONG WITH PRECISION AND RECALL). *International Workshop on Information Retrieval (IR' . . .*, 2001.

[17] Mark Montague and Javed a. Aslam. Condorcet fusion for improved retrieval. *Proceedings of the eleventh international conference on Information and knowledge management - CIKM '02*, page 538, 2002.

[18] Yves Rasolofo, F Abbaci, and J Savoy. Approaches to collection selection and results merging for distributed information retrieval. *. . . international conference on Information . . .*, 2001.

[19] Yves Rasolofo, David Hawking, and Jacques Savoy. Result merging strategies for a current news metasearcher. *Information Processing & Management*, 39(4):581–609, July 2003.

[20] M Elena Renda, Via G Moruzzi, and Umberto Straccia. Web Metasearch : Rank vs . Score Based Rank Aggregation Methods Categories and Subject Descriptors. 2003.

[21] SE Robertson and S Walker. Okapi at TREC-3. *Proceedings of the Third Text REtrieval Conference (TREC 1994). Gaithersburg, USA, November 1994.*, 1994.

[22] Tetsuya Sakai. On the reliability of information retrieval metrics based on graded relevance. *Information Processing & Management*, 43(2):531–548, March 2007.

[23] Milad Shokouhi. Central-Rank-Based Collection Selection in Uncooperative Distributed Information Retrieval. *Advances in Information Retrieval*, pages 160–172, 2007.

[24] Milad Shokouhi and Luo Si. Federated Search. *Foundations and Trends in Information Retrieval*, 5(1):1–109, 2011.

[25] Luo Si and Jamie Callan. Using sampled data and regression to merge search engine results. *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '02*, page 19, 2002.

[26] Luo Si and Jamie Callan. A semisupervised learning method to merge search engine results. *ACM Transactions on Information Systems*, 21(4):457–491, October 2003.

[27] Luo Si and Jamie Callan. Relevant document distribution estimation method for resource selection. *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval - SIGIR '03*, page 298, 2003.

[28] Paul Thomas and David Hawking. Evaluation by comparing result sets in context. *Proceedings of the 15th ACM international conference on Information and knowledge management - CIKM '06*, page 94, 2006.

[29] Paul Thomas and David Hawking. Metasearch tools for desktop search. *... of the SIGIR workshop on desktop search*, 2010.

[30] Paul Thomas and M Shokouhi. SUSHI: scoring scaled samples for server selection. *Proceedings of the 32nd international ACM ...*, 2009.

[31] CC Vogt and GW Cottrell. Fusion via a linear combination of scores. *Information Retrieval*, 27:1–27, 1999.

[32] E Voorhees and DK Harman. *TREC: Experiment and evaluation in information retrieval*. 2005.

[33] EM Voorhees, NK Gupta, and B Johnson-Laird. Learning Collection Fusion Strategies. *Proceedings of the 18th ...*, pages 172–179, 1995.

[34] EM Voorhees and Donna Harman. The Text REtrieval Conference TREC: History and Plan for TREC-9. *ACM SIGIR Forum*, 1999.

[35] Shengli Wu. The weighted Condorcet fusion in information retrieval. *Information Processing & Management*, 49(1):108–122, January 2013.