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. [2]

Mickey M. Mouse
October, 2004

Acknowledgements

I would like to thank my lucky stars, and the cat, for not eating me.

Abstract

This thesis tells a great story about what I achieved in my research project. The abstract is short, but informative. it makes clear the general area in which I worked, and what I achieved.

Contents

List of Figures

Chapter 1

Figures and References

Here's an example of including a figure. Figures need to have size information in them. The only ones I know how to make work are encapsulated postscript files, or eps files. If your figures are not in eps form you may need to find a graphics program which can convert formats. Notice how I can refer to the figure, Fig. 1.1.

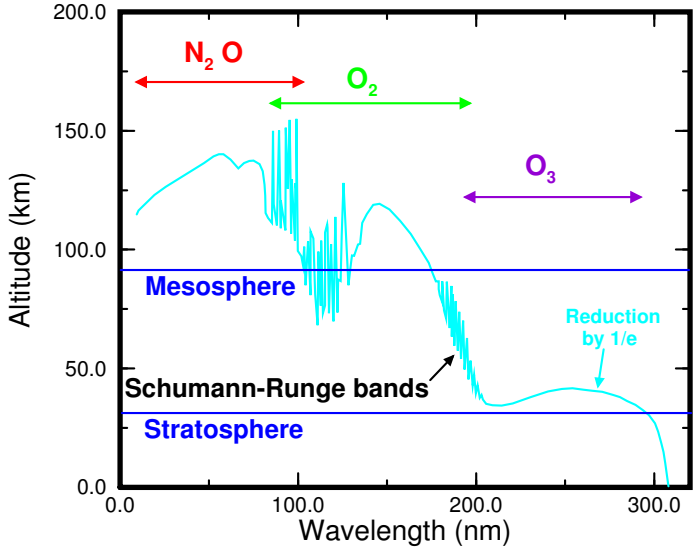

Figure 1.1: This is an example of how to include an eps (encapsulated postscript) figure. Maths is OK in the caption $\epsilon^2 = \Delta$.

Chapter 2

Literature Review

People took years to create information on the Internet, now they have to deal with this huge amount of information. Discovering useful information among the massive amount of resources existing on the Internet has been a hot topic for years, which brought the rise of the field of information retrieval. In early days, companies like Yahoo supplied people with a manually developed directory that classify web sites into different clusters. This is convenient for users to navigate to the website they intend to visit in the time when there are limited numbers of websites existing online. However, as time goes on, such navigation websites are limited by their poor abilities to discover information among the huge amount of websites. Then the more widely used index centralised search engines came into public's eyes. These search engines use crawlers or spiders to crawl pages across the Internet and then index the fetched pages. Companies like Google maintain large amount of data of most of the website online. And they provide very good algorithms to rank all these pages to fit users' requirements. However, this procedure has a disadvantage due to its inability to retrieval information from Deep Web [2]. Traditional index centralised search engines can crawl pages that already existing on the Internet. But as a matter of fact, most of the sites existing on the Internet nowadays are generated dynamically by fetching data from its background database, these data are deep hidden from the surface web that index centralised searching engines can analysis. The size of deep web is much larger than surface web, as pointed out by Bergman [2], and is still growing rapidly. So these difficulties in digging hidden web information has brought many challenges to the index centralised search engines.

2.1 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 [4], sometimes named federated search [11] 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 2.1). People usually separate the meta-search problems into 4 different sub-problems, namely *Resource Description*, *Resource Selection*,*Query Translation*,*Result Merging* [23]. 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

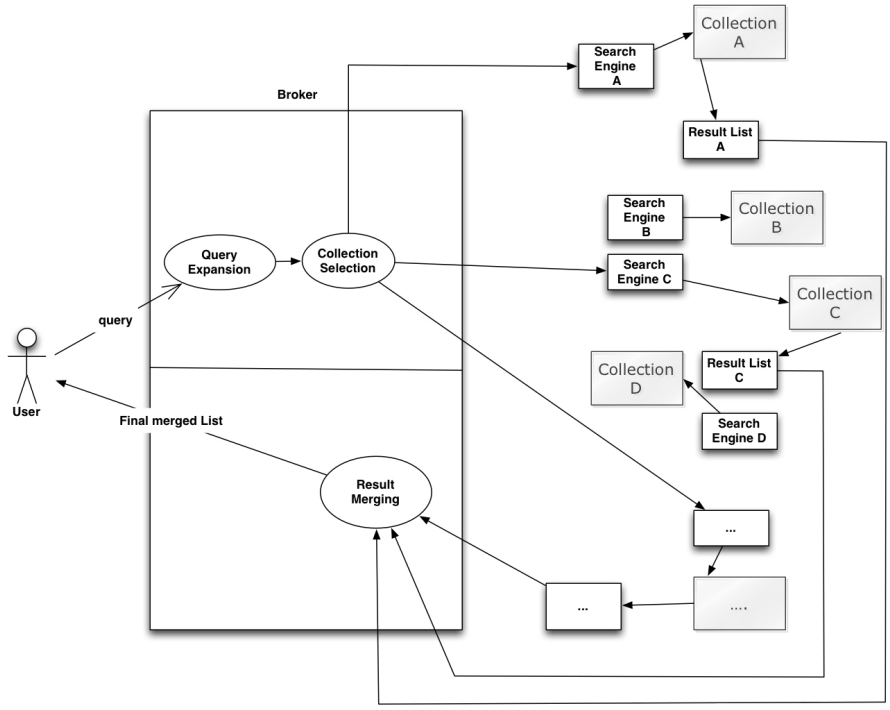

Figure 2.1: A typical architecture of metasearch, the users' requests are fetched by the main component, which is named broker, some query expansion may be executed and the processed query will send to different collections. The collections execute searching on the query and return the retrieval results to the broker

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.

2.2 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 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 [6]. 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 [19] 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 [3, 4] is developed based on an information retrieval system named INQUERY [5], 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 [20, 24, 26] and machine learning approaches [29]have also come into use in this area.

2.3 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 [18]. 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, τ=$[x_1 \geq x_2 \geq \cdots \geq x_k]$, 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, τ_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 τ_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.

2.3.1 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

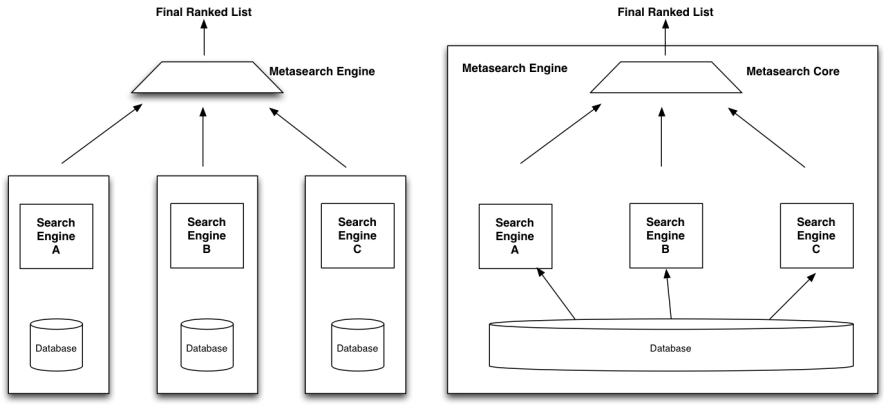

Figure 2.2: Metasearch architectures from [15]

there exists different environment of meta-search. According to the information provided, meta-search can dived into cooperative meta-search and uncooperative meta-search [**?**]. 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. [15] 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 2.2). However, This kind of classification is more focused on 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* and *None-aggregated Meta-Search* is a meta-search built for the purpose of crawling hidden information from the target server. 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.

2.3.2 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 [17]. 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 [16, 17]. 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. [17]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 [17] 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, 18] introduced an algorithm named MinMax to control the range of document score (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, [18] tried to use $Z - scorenormalisation$ (2.3) method to normalise the document score.

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

μ_{s^τ} denotes the means of scores and σ_{s^τ} 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. [17] utilised various document fields to estimate the document score. They generate the document score using a generic document scoring function(Equation 2.3).

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

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 [3, 4] uses a simple linear combination method to normalise the collection score with the collection score get from the collection selection algorithm(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 metasearch built on the bases of INQUIRY system, the limitation draws quite many problems in implementation in other environments. As an alternative, Rasolofo et.al. [16]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 2.6. 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.

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

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 [17] 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* [22,23]. 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 $W = (X^T X)^{-1}(Y^T X)$. 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 2.8) 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 collec-

tions 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 [29]. They proposed two algorithms in [29], 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. CLUSTERING(To be continued). [18]:Markov chain(to be continue)

2.3.3 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. [27] 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. [18] 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 [15]. 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 Si do
3: If Si ranks d1 above d2, count++
4: If Si ranks d2 above d1, count−−
5: If count ¿ 0, rank d1 better than d2
6: Else rank d2 better than d1

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

And they also tried to use training data to get a sever weight by the average precision of individual search system. [30] modified the weighted condorcet fusion algorithms using a linear discriminant analysis(LDA) technology to train the weights.

2.4 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 [28]. However, as discussed by [25], 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. [25] also implemented a tool to evaluate the performance based on embedded comparison and log analysis.

ALL: [**?**, 1–21, 24–29, 31]

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] MK Bergman. The deep web: Surfacing hidden value. *Bright Planet*, pages 1–17, 2001.

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

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

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

[6] Peter B Danzig, Jongsuk Ahn, John Noll, and Katia Obraczka. Distributed Indexing : A Scalable Mechanism for Distributed Information Retrieval 1 Introduction and Motivation. pages 220–229, 1991.

[7] Jürgen Dorn and Tabbasum Naz. Structuring meta-search research by design patterns. *and Technology Conference 2008*, 2008.

[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] Dzung Hong and Luo Si. Mixture model with multiple centralized retrieval algorithms for result merging in federated search. *... on Research and development in information retrieval*, pages 821–830, 2012.

[11] Péter Jacsó. Thoughts about federated searching. *Information Today*, 21(October 2004):7–9, 2004.

[12] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 60(5):493–502, 2004.

[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] Ilya Markov, Avi Arampatzis, and Fabio Crestani. Unsupervised linear score normalization revisited. *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval - SIGIR '12*, page 1161, 2012.

[15] 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.

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

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

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

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

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

[21] Milad Shokouhi and Justin Zobel. Robust result merging using sample-based score estimates. *ACM Transactions on Information Systems*, 27(3):1–29, May 2009.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

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

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

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

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

[31] Ke Zhou, Ronan Cummins, Mounia Lalmas, and Joemon M. Jose. Evaluating aggregated search pages. *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval - SIGIR '12*, page 115, 2012.