Engineering Search Algorithms for Web Data

Leonidas Akritidis

Supervisor: Assoc. Prof. Panayiotis Bozanis
Department of Computer and Communication Engineering
University of Thessaly

Doctoral Committee:

Panayiotis Bozanis, Chair
Elias Houstis
Catherine Housti
Dimitrios Katsaros
Yannis Manolopoulos
Athena Vakali
Charalampos Konstantopoulos

In memory of my father, Gervassios
to my wife, Dimitra
to my mother, Maria, and my sister, Efthimia

ACKNOWLEDGEMENTS

At the end of this long endeavor, I would like to express my sincere gratitude to a group of people who provided me with their valuable assistance, guidance, support and encouragement. Without their help it is certain that this dissertation would never have started, and/or it would never have reached the required levels of quality.

First of all, I would like to express my gratitude to my thesis supervisor, Professor Panayiotis Bozanis for his vital guidance and support. Without his enormous encouragement this research effort would never have been achieved. Although I did not graduate from this Department and he was not aware of my skills, Professor Bozanis initially accepted me as a PhD student, and in the sequel he provided me with all the necessary knowledge required to conduct a scientific research at the PhD scale. It was not only the scientific advices and guidance, but also the important financial support (from the projects I participated) which ensured the production of this dissertation. Professor Bozanis is an excellent scientist and a valuable collaborator; but above all he is a true supporter. In case this text is being read by an undergraduate student who desires to start or continue an academic career, I definitely suggest that he/she visits Professor Bozanis.

This effort would never have started without the encouragement of Lecturer Dimitrios Katsaros. He was the first who saw an alpha version of QuadSearch, a search system which I developed to assist a friend of mine in his MSc studies. He brought me to the Department and suggested Professor Bozanis for the supervision of my PhD. Lecturer Katsaros was a co-author in several of our publications; his contribution was really valuable, whereas his notes on my manuscripts assisted me in understanding the most important concepts of technical writing. I definitely appreciate his excellence in science and I deeply thank him for his sincere support.

I also thank my thesis committee members, Elias Houstis, Catherine Housti, Yannis Manolopoulos, Athena Vakali, and Charalambos Konstantopoulos for reading my dissertation and for providing valuable comments which assisted me in enhancing its quality.

I would like to write an entire chapter to describe the sincere support of my wife Dimitra, but since this is an acknowledgements section in a PhD dissertation, I must limit my gratitude to only a small paragraph. Dimitra was always there to accept my disappointment

in the hard moments of my research, but she always had a (usually humorous) comment of encouragement to make. I apologize for these unpleasant moments and also, for the time I spent in studying and researching instead of investing it to her. Undoubtedly, a considerable part of this dissertation belongs to her and I express my thanks for her patience, support and devotion.

This dissertation is dedicated to the memory of my father Gervassios, who deceased during this research. His death was a great loss for me; fortunately, the encouragement of the other members of my family, my mother Maria, and my sister Efthimia helped me continue and finish this work. I really owe them my gratitude.

Finally, but most importantly, I would like to thank God, His Son and The Holy Spirit whose constant presence in my life enlightens my path and gives me hope.

Leonidas Akritidis
BSc in Electrical and Computer Engineering
PhD in Computer and Communication Engineering
Volos, 2013

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Engineering Search Algorithms for Web Data
by
Leonidas Akritidis

Supervisor: Prof. Panayiotis Bozanis

The massive growth of the information produced and disseminated through the World Wide Web (WWW) has rendered Information Retrieval (IR) one of the most important and challenging research fields in modern computer science. As hundreds of Gigabytes are being published on the Web in a daily basis and billions of users require access to this huge amount of data, search engines have to constantly scale up in terms of both efficiency and effectiveness.

In this dissertation we present novel engineering algorithms which contribute to the solution of key problems related to the current Web search engines. These algorithms lead to improvements in the query throughput of these systems (that is, the rate at which they serve the incoming queries), and the quality of the results they produce in response to these queries. In particular, we introduce PFBC, an efficient algorithm for organizing and compressing the positional data stored within an inverted index. In the sequel, we expand PFBC with the aim of supporting additional data within an inverted list posting such as the field (or zone) of a document where a word occurs. The new algorithm, namely TZP, exhibits a wide range of advantages against the current state-of-the-art generic integer compression methods. Based on TZP, we introduce BM25TOPF, a probabilistic ranking function which in contrast to the existing probabilistic functions of the Okapi family, takes into consideration the word ordering in the query, and combines term proximity with zone scoring.

Furthermore, we examine the essential problems related to vertical searching, that is, searching for information by accounting only a specific portion of the Web. In particular, we study the problem of quantifying the influence flow in Blogosphere by taking into consideration the particular features which characterize Blogosphere such as the rapid blog

post production and the temporal instability of this environment. We propose three such metrics: MEIBI, MEIBIX and the BP/BI-index. In the sequel, we examine how the proposed models for quantifying the bloggers' influence can be employed by a vertical blog search engine to improve the quality of the generated results.

Another vertical search system which gained our attention is academic search engines. In this dissertation we conducted a three-way research; The first part includes proposal of new scientometrics that is, metrics which measure the quality of the work of a scientist. We introduce the f-index, a novel metric which embodies coterminal citations and presents them as a generalization of self-citations and of co-citation. In addition, we introduce the topic-sensitive extensions, special versions of the most important scientometrics which attempt to evaluate the work of a scientist in only one particular research field. In the sequel, we discuss four strategies for computing these metrics in large-scale datasets by using a special-purpose algorithm parallelization framework (Hadoop/MapReduce). Finally, our last contribution regards a supervised machine-learning algorithm for classifying research papers. The results of all these three parts of our research can be utilized by all the current academic search engines and digital libraries to enhance their functionality.

The final contribution of this dissertation concerns the problem of rank aggregation, or rank fusion. Here we present a family of algorithms which provide an effective manner for combining and re-ranking the results coming from multiple search engines. The new algorithms, QuadRank and the KE family take into consideration both statistical data (i.e. the individual rankings of each item and the number of its appearances) and document-related information (i.e. zone weighting, URL, etc). All these algorithms have been implemented within QuadSearch, a prototype metasearch engine which we have developed as a testbed for evaluating new rank aggregation methods and generic solutions related to the wider problem of metasearching.

ΠΕΡΙΛΗΨΗ

Αλγοριθμικές Τεχνικές Αναζήτησης Πληροφορίας σε Δεδομένα του Παγκόσμιου Ιστού
Συγγραφή:
Λεωνίδας Ακριτίδης

Επιβλέπων: Παναγιώτης Μποζάνης

Η μεγάλη διόγκωση της πληροφορίας που παράγεται και διακινείται μέσω του Παγκόσμιου Ιστού κατέστησε το επιστημονικό πεδίο της Ανάκτησης Πληροφορίας (Information Retrieval, IR) ένα από τα σημαντικότερα στη μοντέρνα επιστήμη των Υπολογιστών. Καθώς εκατοντάδες Gigabytes δημοσιεύονται στον Παγκόσμιο Ιστό σε καθημερινή βάση και δισεκατομμύρια χρηστών απαιτούν άμεση πρόσβαση στην παραχθείσα πληροφορία, οι σύγχρονες μηχανές αναζήτησης πρέπει να επιτυγχάνουν συνεχή κλιμάκωση τόσο σε αποτελεσματικότητα, όσο και σε αποδοτικότητα.

Σε αυτή τη διατριβή παρουσιάζουμε νέους και καινοτόμους αλγορίθμους οι οποίοι συνεισφέρουν στην επίλυση σημαντικών προβλημάτων που σχετίζονται με τις τρέχουσες μηχανές αναζήτησης. Οι αλγόριθμοι που παρουσιάζονται εδώ οδηγούν σε βελτίωση τόσο της ταχύτητας απάντησης των ερωτημάτων (δηλαδή του ρυθμού με τον οποίο οι μηχανές αναζήτησης εξυπηρετούν τα εισερχόμενα ερωτήματα), όσο και της ποιότητας των αποτελεσμάτων που επιστρέφουν οι μηχανές σε απόκριση αυτών των ερωτημάτων. Πιο συγκεκριμένα, εισάγουμε τον PFBC, ένα αποδοτικό αλγόριθμο για την οργάνωση και τη συμπίεση των δεδομένων θέσης που είναι αποθηκευμένα στο ανεστραμμένο ευρετήριο μιας μηχανής αναζήτησης. Στη συνέχεια, επεκτείνουμε τον PFBC αλγόριθμο με σκοπό την υποστήριξη επιπρόσθετης πληροφορίας μέσα σε μια ανεστραμμένη λίστα του ευρετηρίου. Η επιπρόσθετη πληροφορία αφορά στο πεδίο (ή στη ζώνη) ενός εγγράφου μέσα στο οποίο συναντάται μία λέξη. Ο νέος αλγόριθμος που ονομάζεται TZP, παρουσιάζει ένα μεγάλος εύρος πλεονεκτημάτων έναντι των τρεχουσών, κορυφαίων και γενικών μεθόδων συμπίεσης ακεραίων. Με βάση τον αλγόριθμο TZP εισάγουμε την BM25TOPF πιθανοτική συνάρτηση κατάταξης η οποία σε αντίθεση με τις υπάρχουσες συναρτήσεις της οικογένειας Okapi, λαμβάνει υπόψη της τη σειρά με την οποία διατάσσονται οι λέξεις

στα υποβληθέντα ερωτήματα και συνδυάζει την εγγύτητα όρων (term proximity) και την απόδοση βάρους στις ζώνες (zone weighting).

Επιπλέον, εξετάζουμε ορισμένα από τα πιο ουσιαστικά προβλήματα που σχετίζονται με τις κάθετες αναζητήσεις, δηλαδή τις αναζητήσεις που πραγματοποιούνται λαμβάνοντας υπόψη μόνο ένα συγκεκριμένο τμήμα του Παγκόσμιου Ιστού. Μελετάμε το πρόβλημα της ποσοτικοποίησης της ροής της επιρροής στη Blogosphere συμπεριλαμβάνοντας υπόψιν τα ιδιαίτερα στοιχεία που τη χαρακτηρίζουν όπως την ταχύτατη παραγωγή εγγράφων και τη χρονική αστάθεια του περιβάλλοντος. Προτείνουμε τρία διαφορετικά μετρικά: το MEIBI, το MEIBIX και το δείκτη BP/BI. Στη συνέχεια εξετάζουμε μεθοδολογίες με τις οποίες είναι δυνατή η εκμετάλλευση των προτεινόμενων μοντέλων από μία κάθετη μηχανή αναζήτησης ιστολογίων, ώστε να βελτιωθεί η ποιότητα των παραγόμενων αποτελεσμάτων.

Ένα άλλο κάθετο σύστημα αναζήτησης το οποίο κέρδισε την προσοχή μας είναι οι ακαδημαϊκές μηχανές αναζήτησης. Η έρευνα που πραγματοποιήσαμε σε αυτή τη διατριβή συγκροτείται από τρία διαφορετικά μέτωπα: Το πρώτο μέτωπο περιλαμβάνει προτάσεις νέων βιβλιομετρικών δεικτών, δηλαδή μετρικών τα οποία επιχειρούν να μετρήσουν την αντικειμενική αξία της συνολικής εργασίας ενός επιστήμονα. Εισάγουμε το δείκτη f (f-index) ο οποίος ενσωματώνει την έννοια των συν-τερματικών αναφορών και τις παρουσιάζει σαν μία γενίκευση των αυτό-αναφορών (self-citations) και των συν-αναφορών (co-citations). Επιπροσθέτως, εισάγουμε νέες επεκτάσεις στους πιο διαδεδομένους βιβλιομετρικούς δείκτες, οι οποίες επιτρέπουν την αξιολόγηση του επιστημονικού έργου κάθε ερευνητή κατά επιστημονικό πεδίο. Στο επόμενο στάδιο παρουσιάζουμε τέσσερις στρατηγικές για τον παράλληλο υπολογισμό των βιβλιομετρικών δεικτών σε μεγάλα σύνολα δεδομένων, χρησιμοποιώντας το Hadoop/MapReduce, ένα γενικού σκοπού σύστημα κατανομής αλγορίθμων σε πολυπληθείς ομάδες υπολογιστών. Στο τρίτο και τελικό στάδιο παρουσιάζουμε ένα νέο αλγόριθμο εκμάθησης μηχανής (machine-learning algorithm, MLA) για την κατηγοριοποίηση των ερευνητικών άρθρων. Τα αποτελέσματα και των τριών μετώπων έρευνας που παρουσιάζονται εδώ μπορούν να χρησιμοποιηθούν από τις σύγχρονες ακαδημαϊκές μηχανές αναζήτησης και τις ψηφιακές βιβλιοθήκες ώστε να βελτιώσουν την ποιότητα των παρεχομένων υπηρεσιών τους.

Η τελευταία συνεισφορά που παρουσιάζουμε σε αυτή τη διατριβή αφορά στο πρόβλημα της ενοποίησης και της σύγκρισης λιστών αποτελεσμάτων. Παρουσιάζουμε μία οικογένεια αλγορίθμων οι οποίοι παρέχουν αποτελεσματικό συνδυασμό και ανακατάταξη των αποτελεσμάτων που προέρχονται από πολλαπλές διαφορετικές μηχανές αναζήτησης. Εισάγουμε τη μέθοδο QuadRank και την οικογένεια KE τα οποία λαμβάνουν υπόψιν τους τόσο στατιστικά δεδομένα (όπως τις μεμονωμένες κατατάξεις κάθε αντικειμένου και το πλήθος

των εμφανίσεών του), όσο και πληροφορίες σχετικές με τα ανακτηθέντα έγγραφα (ζώνες εμφάνισης των όρων αναζήτησης, URL, κλπ). Οι αλγόριθμοι αυτοί έχουν υλοποιηθεί μέσα στην QuadSearch, ένα πρωτότυπο σύστημα μετα-αναζήτησης που αναπτύξαμε με σκοπό την αξιολόγηση νέων μεθόδων σύγκρισης κατατάξεων, αλλά και γενικών λύσεων σε προβλήματα που σχετίζονται με το ευρύτερο ζήτημα της μετα-αναζήτησης.

CHAPTER I

Introduction

1.1 Information production, dissemination and searching in World Wide Web

Since its invention at CERN in 1989, the World Wide Web (WWW, Web) has evolved into the largest repository of information worldwide. Its size is reflected by both the huge volumes of data scattered across hundreds of millions servers, and the rapid rates at which this data grows. Furthermore, billions of users connect to the Web daily to gain access to a portion of this data, whereas their demands for fast and accurate retrieval of qualitative information sources increase over time.

However, the explosion which led to the Web's present form was triggered by the participatory concept introduced by Web2.0, where users not only access the provided information, but also they produce it. The birth of Web2.0 contributed to the creation of multiple interactive services including social networks, blog sites and communities, forums, wikis, media sharing sites and countless others. On the other hand, the traditional server-based applications such as search engines evolved with the aim of taking into consideration the users' preferences, click through data, explicit feedback, previous attitude, search history, etc.

In such a huge and highly volatile environment, the wide research area of Information Retrieval (IR) obtained a crucial role in modern computer science. Unless it is easily accessible, the information published on the Web is close to useless. On the other hand, the users require swift and effortless identification of sources which satisfy their information needs. To sustain such heavy workloads and to achieve high user satisfaction, the search systems are required to constantly improve in terms of both efficiency and effectiveness.

Efficiency is a measure which is primarily reflected by query throughput, that is, the speed at which a system responds to an incoming request. Accepting queries at rates touch-

ing tens of thousands per second, it is well understood that the underlying data structures and query processing algorithms of a search engine must be fully optimized. On the other hand, effectiveness mainly concerns the quality of the generated results. Since the Web expands at enormous rates and the user expectations grow, the ranking methods of search engines must constantly evolve to provide unbiased and qualitative results. In this dissertation we describe strategies which contribute to the improvement of both of these attributes.

Moreover, the diversity of the information published on the Web led to the creation of specific-purpose search services called vertical search engines. These systems operate similarly to the traditional Web search engines, however, they limit the range of their functionality to only a particular portion of the Web. For instance, a blog search engine allows its users to search among blog posts, communities and authors. Another example of a rather popular vertical search engine are the digital libraries and the academic search engines which provide focused search capabilities for scientific documents and researchers.

Regarding blog search, one of the most important problems is the identification of the most influential authors (bloggers); the influentials are usually connected in large virtual communities and hence, they can play a special role in multiple ways. For instance, commercial companies and organizations can turn their interest in gaining the respect of the influentials to become their "unofficial spokesmen", instead of investing huge amounts of money and time to advertise their products to thousands of potential customers. The influentials could also be responsible for forging political agendas by affecting the other peoples' voting behavior. Therefore, the identification of the bloggers, who exhibit remarkable activity and influence, is of great importance and can lead to the development of innovative business opportunities (related to the commercial transactions and traveling). In this dissertation we propose three methodologies for identifying the most influential bloggers in community blog sites.

Another field of research conducted within this dissertation concerns the academic search engines, digital libraries, and scientific databases. Following the evolution of the Web search engines, these services have significantly enriched the content of their result pages. Hence, the problems related to the quality of a research article or the estimation of the influence of an author have became particularly important. In this work we introduce several methods which contribute to the issue of evaluating the research work of a scientist.

On the other hand, the constantly growing repositories of the academic search engines posed new issues, such as the parallelization of the algorithmic solutions of the aforementioned problems. Here we study for first time the topic of the parallel computation of the scientists' evaluation metrics by using MapReduce, the well-established parallelization framework of [41]. We propose four different strategies and we show that the usage of

in-Mapper Combiners facilitates efficient execution thus improving both running times and network bandwidth.

We also examine the interesting problem of research articles' classification by introducing a novel supervised machine-learning approach. Our algorithm includes a training phase which employs a set of labeled documents and trains a model according to the articles' keywords, its authors previous papers, co-authorship information, and journal history. In the sequel, it exploits the trained model to assign labels to the unlabeled articles. The proposed classification method was compared against the two state-of-the-art machine-learning algorithms, Support Vector Machines and AdaBoost and was found more accurate in research article categorization by approximately 10%.

This method was used in another research topic of this dissertation, the identification of the scientific fields which are attractive for starting scholars. In this study we carefully examine the characteristics of the new scientists and the attributes which render an area of research "hot" for them. The outcome of this interesting work was that not all popular and emerging fields are suitable for new scientists; instead, we must take into consideration the lack of experience and the lack of trust by the rest of the academic community. Our experimental evaluation was based on a large publicly available dataset and the included articles were classified by exploiting the aforementioned machine-learning algorithm.

The metasearch engines comprise another set of particularly interesting search engines. Designed with the aim of combining the power of the plain Web search engines, these services offer increased Web coverage (since the full coverage of the Web is considered impossible for a single search engine) and improved retrieval effectiveness because they exploit the strengths of their component engines. In the last chapter of this dissertation we discuss the rank aggregation problem in the context of metasearching and we introduce a family of effective algorithms. We also present QuadSearch, a prototype metasearch engine which have been developed in order to attest the usefulness of our proposed methods.

1.2 Contributions of this dissertation

In this section we briefly refer to the most important contributions of this dissertation and we provide a general description of its organization.

In chapter II we study efficiency issues related to the Web search engines with the aim of improving their query throughput, that is, the rate at which they answer queries. We state that one of the strategies for achieving this goal is the introduction of robust organization and compression algorithms for the primary data structure of a search engine, its inverted index. More specifically:

- We state that the most efficient algorithms for compressing integers are the block-based approaches, i.e., those which are capable of encoding entire bundles of integers such as PForDelta and VSEncoding. We report that although these methods work well for the docIDs and the frequency values of an inverted list, they are not practical for encoding the positional data because during query processing we are interested in decoding only limited data, and the decompression of whole blocks of integers is redundant. Furthermore, these methods do not allow direct access to the desired data; a separate and costly look-up operation is required (section 2.2).

- We introduce *PFBC* (Positions Fixed Bit Compression), a method which encodes the positions of an inverted list block by using a fixed number of bits. We show that with this organization and by maintaining a limited number of pointers within the list's skip table, we are allowed direct access to the positional data, and also, we decode only the information actually needed (section 2.3).

- In section 2.4 we study the potential of including additional information within the inverted list in an effective and efficient manner. In particular, we replace the plain positional value within a posting by the notion of *occurrence*. In this dissertation the introduced occurrences consist of both the positional data and the physical location of a document where a word occurs (*field* or *zone*).

- We introduce *TZP*, a method for compressing the aforementioned word occurrences. TZP dictates that each document zone is assigned a unique zoneID value and that each zoneID is accompanied by the corresponding positional value. This position-zoneID pair is encoded by using a fixed number of bits, in a fashion similar to that of PFBC. Due to this strategy TZP exhibits the same advantages over the competitor compression algorithms, as PFBC (section 2.6).

- We conduct exhaustive experiments by using a large document collection comprised of about 50 million Web documents, ClueWeb09. Our experiments demonstrate that both PFBC and TZP outperform PForDelta, an optimized PForDelta variant (OptP4D), and VSEncoding in terms of positional data access and decompression speed by a large margin, whereas they introduce infinitesimal losses in the overall size of the index.

In chapter III we address the problem of identifying the influential bloggers in a community by taking into consideration the special features of Blogosphere. The most important contributions of this chapter are:

- We study an early influence estimation model, the *influence-flow model* and we expose its main drawbacks. We state that for the identification of the most influential bloggers it is required that we consider *all* the publications of a blogger, and not just the best one. In addition, we take into consideration the temporal aspects of the Blogosphere.

- Based on the previous analysis, we introduce two novel metrics for identifying the influentials; *MEIBI* and *MEIBIX*. These metrics address both of the aforementioned drawbacks of the influence-flow model. The former takes into consideration the age of a blog post (in days), whereas the latter depends on the age of the incoming references of a post. Furthermore, the definitions of MEIBI and MEIBIX take into account all the posts of a blogger (section 3.3).

- We introduce the *BP/BI-index*, a two-dimensional metric which combines the influence of a blogger with his/her productivity. This metric is also time-sensitive and classifies the bloggers into four classes: $\mathcal{A}$ (currently influential and productive), $\mathcal{B}$ (influential but not productive), $\mathcal{C}$ (productive but not influential) and $\mathcal{D}$ (nor productive nor influential). See section 3.4 for more details.

- We attest the usefulness of our approaches by experimenting with real-world data obtained from three blog communities; The Unofficial Apple Weblog, Engadget, and Techcrunch technology forum.

The problem of ranking in Web search engines and the vertical blog search services is the main subject of chapter IV. Here we apply the contributions of chapter II (combination of positional and field data) and of chapter III (bloggers' influence evaluation metrics) with the aim of improving the retrieval effectiveness. In short, the contributions discussed in this chapter are:

- We investigate the potential of combining term proximity scoring, correct term ordering, and zone weighting into a single probabilistic ranking function. We introduce *BM25TOPF*, a function which takes into consideration the proximity of the query terms in a document (similarly to BM25TP), the location of a document where the query terms occur (similarly to BM25F), and the ordering of the query terms in a document (section 4.2).

- We perform effectiveness measurements by using data from the WebTrack task of the TREC 2009 conference. Our findings indicate that BM25TOPF outperforms both BM25F and BM25TP by at least 8%.

- We attempt to apply the bloggers influence evaluation metrics of chapter III to a vertical blog search engine in order to enhance retrieval effectiveness. In this case we are mainly interested in retrieving opinionated blog entries, that is, articles which are considered relevant to a given query, only in case they contain an opinion (either positive or negative) about the subject of the query (section 4.3).

- We perform effectiveness measurements by using data from the BlogTrack task of the TREC 2009 conference. Our findings indicate that our retrieval model outperforms other state-of-the-art approaches by around 6%.

The academic search engines and numerous related problems are the primary research issue of chapter V. The contributions of chapter V are summarized into the following list:

- We to propose effective metrics for evaluating the research work of a scientist in a fair and unbiased manner. We introduce the notion of coterminal citations and we show their connection to co-citations and self-citations. We present the f-index, a metric which embodies and quantifies the impact of coterminal citations in scientists ranking (section 5.2).

- We enhance the existing state-of-the-art scientometrics by introducing their Topic-Sensitive extensions. These extensions allow us to estimate the impact of the work of an author in a particular area of research.

- Due to the constantly increasing sizes of the scientific databases, the calculation of the aforementioned scientometris becomes more complex, since the volumes of the involved data cannot be handled by a single workstation. We identify the problem and we propose four strategies to support the parallel computation of scientometrics by using Hadoop/MapReduce, a fault-tolerant framework for data and algorithms distribution (section 5.3).

- We formulate the problem of identifying attractive research areas for new scientists. Initially, we provide a detailed description of the provided data and in the sequel, we formally state the problem itself along with its component issues. Our proposed solution is designed to take into consideration several aspects regarding the attractiveness of a research area and the characteristics of the new scientists (section 5.4).

- We study the issue of research articles' classification, a key component for every successful digital library or academic search engine. We present a supervised machine-learning classification approach which achieves more accurate categorization of the

unlabeled articles compared to the traditional Support Vector Machines and AdaBoost (section 5.5).

Finally, in chapter VI we discuss the problem of metasearching and we introduce a family of rank aggregation methods. More specifically:

- We introduce the *KE* algorithm, a rank aggregation formula which combines the results lists coming from different sources. The method incorporates several features including the individual ranking each item was assigned, the number and length of the component results' lists, as long as the total number of the appearances of each item in all lists. Furthermore, we provide three more KE variants: the *GeoKE* which combines the aforementioned features with the locality of the user and the origination of the item, the *weighted KE* which assigns weights to the component lists and the *URL-Aware KE* which takes into consideration of the URL structure of each item (section 6.3).

- We present another rank aggregation method, namely *QuadRank* which is primarily oriented towards Web metasearching. QuadRank incorporates the most effective features of the KE family and injects new features such as zone weighting. The experimental evaluation of QuadRank demonstrates its superiority against other adversary approaches such as Borda Count, the Condorcet method and the Outranking Approach (sections 6.4 and 6.5).

- We present the primary architectural components and design issues of QuadSearch, our prototype metasearch engine and we perform a detailed benchmark indicating the system's efficiency (section 6.6).

CHAPTER II

Compressing Block-Based Inverted Indexes

2.1 Introduction

Nowadays, the repositories of the major search engines accommodate tens of billions of documents [40] and as the Web expands and the crawling technology evolves, these repositories are expected to grow further. Furthermore, search engines accept and answer thousands of queries per second attempting to quickly retrieve the most suitable documents for each submitted query. In such a dynamic environment where the available information, the workload and the user expectations continuously grow, search engines have to constantly scale up in terms of both efficiency (query throughput) and effectiveness (quality of query results).

The inverted index is the primary data structure used by search engines for storing document related data and metadata. According to [147, 159], an appropriately constructed inverted index can improve the performance of query processing dramatically. Due to the importance of the inverted index in the overall efficiency of a search system, there has been a lot of research conducted towards its optimization. Optimization primarily regards two critical issues: *compression* and *organization*. The former is a key issue for reducing the overall index size and minimizing the transfer costs from either disk or main memory. The latter enables partial access of the index structure, that is, a query can be answered without having to traverse all the available information stored in it.

The benefits of these methods are magnified in the case where we store positional data within the index, because the size of the positions is several times larger than that of docIDs and frequencies and the indexes containing positional values are about 3 to 5 times larger than the non-positional ones [154]. Therefore, it is extremely important to devise an effective mechanism to organize and compress the positional data, since a naive solution could lead to prohibitively large indexes and reduced query throughput.

In this chapter we demonstrate that although the current state-of-the-art compression methods (such as PForDelta [69] and VSEncoding [129]) are both effective and efficient when applied at docIDs and frequencies, they do not perform equally well when they operate on the positional data of an inverted list. We introduce PFBC, a scheme which encodes the positions of an inverted list block by using a fixed number of bits allowing us to a) access the required data instantly and b) decode only the data actually needed, without touching any unnecessary information. Through extensive experimentation, we demonstrate that with a small cost in space, PFBC outperforms all the adversary compression methods in terms of speed when applied on the positional data of the index.

In addition, several engineers (see for instance [40]) have stated that the information stored within the indexes of the modern commercial search engines has tripled during the past few years. However, in the literature we mainly encounter strategies and algorithms concerning typical inverted indexes, which almost always store very limited data: document identifiers, word-document frequencies and word positions in a document. Compared to the hundreds of the parameters employed by the major search engines for ranking their documents [149], this data is apparently inadequate.

In this chapter we also study the potential of including additional information within the inverted index. In particular, we adopt the idea of partitioning a Web document into locations of special interest, namely *fields* or *zones*. The document zones were initially introduced in [92], but to the best of our knowledge, issues regarding the compression and organization of such indexes have never been studied before. We investigate the meaning of a word's appearance within a document and we replace the plain positional data by the *occurrences*, a piece of information which contains both the position and the zone of the document where this specific word appears.

In the sequel, we propose a method which allows compact storage of zones along with the corresponding word positions. Our approach, namely *TZP*, operates in a spirit similar to PFBC and is designed to support all the inverted list partitioning strategies that have been proposed so far (refer to [100, 26, 19, 154, 32]). TZP operates in two steps: In the first step, the compressor packs each position-zone pair of a block into a 32-bit space and in the next phase, these packets are encoded together by employing a fixed number of bits. This scheme enables the direct access of the occurrence data for each posting, by using a limited number of pointers (one pointer per block).

As a summary, the contributions of this chapter are:

- We introduce *PFBC*, an algorithm for efficient compression of the positional data in an inverted index without look-ups.

- We introduce a two-level encoder for positions and zones, namely TZP. TZP compresses the data by using a fixed number of bits for each position-zone pair of the same block of an inverted list.

- We show that the fixed-bit policy adopted by PFBC and TZP allow very fast decompression, whereas they also facilitate direct access to the data actually required for processing a query. Both methods introduce a formula which allow us to *compute* the exact location of the desired data, thus saving the cost of look-ups. Moreover, we do not decode redundant information (i.e blocks of integers), but only the data *actually* needed.

- All our contributions are experimentally evaluated by using the Clueweb09-T09B document collection consisting of roughly 50 million documents.

The rest of the chapter is organized as follows: In section 2.2 we refer to the state-of-the-art methodologies for organizing inverted indexes and we cite the relevant work. In section 2.3 we present the principles of PFBC, whereas in section 2.4 we provide a description of the document zones. In section 2.5 we show how the zones can be integrated within the inverted index and in section 2.6 we present the TZP compression algorithm for zones and positions. The chapter closes with with the experimental evaluation of our methods in section 2.7, and our conclusions in section 2.8.

2.2 Preliminaries and related work

The inverted index is the primary data structure constructed and maintained by the search engines for serving user queries. There is a significant amount of research regarding the efficient organization of the inverted index and in this section we briefly describe some basic elements deriving from the related theory.

A typical inverted index structure consists primarily of two components: (i) the *lexicon*, a list sorted in ascending lexicographical order containing all the distinct words appearing in the collection and (ii) the *inverted file*, that stores all the occurrences of each word in the collection. These occurrences are organized in *inverted lists*. In its simplest form, an inverted list I_t of a term t stores a list of *postings* which contain the integer identifiers of the documents (docIDs) where t appears into.

To support ranked query processing, we store additional information within an inverted list: (i) the term-document frequency (or just frequency) $f_{d,t}$, which reveals how many times a term t appears in document d i.e. each posting S_i is of the form (d_i, f_i) and (ii)

some positional data, $p_{i,k}$ indicating the position of the term in the document. In this case the postings are of the form $(d_i, f_i, p_{i,0}, p_{i,1}, ..., p_{i,f_i-1})$. The inverted lists can be sorted either by docID, or by another attribute (frequency or another scoring value [30]. Here we consider the situation where the inverted lists are sorted by increasing docID order; this setup allows more effective index compression (refer to [147, 32]) and supports the parallel traversal of all of the query terms' inverted lists during query processing [92].

The inverted lists are stored in a highly compressed form either on disk, or in main memory. Compression has multiple advantages since it does not only reduce the storage requirements, but it also decreases transfer costs [147]. However, the compressed data needs to be quickly accessed and decompressed when a query is submitted and in case of improper implementation, decoding could be a serious bottleneck for query throughput in search engines.

There are two primary methods exploiting the inverted index to evaluate a query: *term-at-a-time* and *document-at-a-time* [140]. The first approach initially orders the query terms in increasing frequency order; in the sequel the inverted list of each term is repeatedly merged with the lists of the other terms, leading to the final result list (see [147] for a detailed description). On the other hand, in the latter method, the inverted lists are scanned in parallel sequentially retrieving the documents which are relevant to the query. Document-at-a-time evaluation is essential for large document collections where we can predict the number of documents that could be retrieved (by using statistical methods) and the operation can be terminated as soon as adequate qualitative documents have been retrieved [26].

To perform efficient parallel scanning of several inverted lists, it is beneficial that we maintain a mechanism which allows us to skip large portions of the list by seeking the first docID larger than or equal to a given one. In this way we avoid decoding useless portions of the list and we are able to quickly access the desired information. For this reason, multiple works such as [96, 100, 26, 19, 154, 32] propose the partitioning of the inverted list into a series of adjacent *blocks* and the maintenance of one or more pointers pointing to the beginning of each block. During the evaluation of a query, the processor makes use of these pointers to locate the correct block and decode only the data actually needed.

A block-based inverted list organization is depicted in Figure 2.1. The data in each block is stored in three *chunks*: the first chunk is used to store the docIDs, the second chunk stores the corresponding frequency values, whereas the third chunk contains the positional data. Since the number of the occurrences of a term within a document can be infinitely large, an important issue posed by the usage of a block-based scheme is to identify the location of the positional data for a particular posting. The problem becomes even more

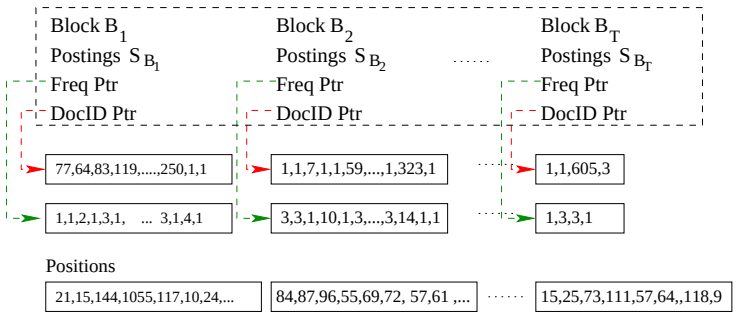


Figure 2.1: Partitioning an inverted list into T blocks of postings (block-based organization). In the upper part we depict the skip table which stores two pointers per block: one pointing at the docIDs and one pointing at the frequency values.

challenging in case the corresponding data is compressed, because we need to locate the desired information within the *compressed* byte sequence.

To address this issue, the researchers have proposed two basic ways to organize the data: (a) interleaving, i.e. the positional data of a particular block are stored after the docIDs and the frequencies of the same block, and (b) creating a completely separate structure for positions with its own lookup mechanism. For instance, [148] propose interleaving, and they introduce a fairly standard hierarchical look-up structure to access the positions. For each block of an inverted list, this structure stores one docID and one pointer to the beginning of the block. Furthermore, within each block the positional data is organized into sub-blocks of $\mathcal{B}$ postings. For each of these sub-blocks, a pointer is used to store its offset from the beginning of the block. To retrieve the positional data of a specific posting we first search for the correct sub-block and then we decode all the positions for the postings of this sub-block. Then the positions of the particular posting are retrieved by using the aforementioned offset value. Nevertheless, this look-up operation is quite expensive and can decelerate query processing especially in case $\mathcal{B}$ is small. Furthermore, this additional structure occupies extra space in memory.

On the other hand, [136] organizes the positions by employing a separate structure, namely indexed list. This list consists of two levels; the upper level contains pointers pointing to the data stored at the lower level. That is, for each posting they store a pointer which shows the location of the corresponding positional values. Although this approach offers direct access to the positional data without look-ups, it requires even more space than the aforementioned look-up structure since storing one pointer per posting is very expensive.

In this chapter we propose a hybrid between interleaving and data structures. In particular, we choose to store the positional data contiguously (i.e. not in blocks), but along

with each block we also record a pointer which in combination with our encoding method, it allows us to access *directly* the positions of a specific posting. This renders our approach much more economic than that of [136], since instead of requiring one pointer per posting, it requires one pointer per block.

On the other hand, the problem of selecting appropriate block sizes has been widely studied: For instance, the work of [96] proposes setting skip pointers every $\sqrt{N_t}$ postings of the inverted list, where N_t is the number of documents containing t. A number of research articles [154, 43, 148] place skips each time a fixed number of postings (i.e. 128) has been encountered. Other papers study the issue of dynamically setting skip pointers in a fashion which maximizes query throughput. For instance, [26] embed compressed perfect skip lists in an inverted list to increase the processing speed, whereas [129] introduced a novel class of encoders which partition the list in an optimal way that maximizes decompression rates by using dynamic programming.

Here we do not examine in depth the issue of inverted list partitioning. We rather focus on the efficient representation of positions and zones which allows fast query evaluation. However, notice that all the aforementioned skipping/partitioning techniques can be used in combination with our proposals with no additional effort.

2.3 PFBC: Positions Fixed Bit Compression

In this section we describe PFBC, a simple, yet efficient approach for encoding and organizing the positional data of an inverted list. Initially we present the ideas and the algorithm that characterize the compression part, and in the sequel, we show how this method allows fast access to the required data.

2.3.1 Compressing the positional data with PFBC

Now let us present the manner PFBC operates during the compression phase; the method is designed to encode the positional values of an inverted list block by using a fixed number of bits. PFBC also requires the existence of a pointer which shows the location of the first position of this block. With this information, we will later demonstrate how we can access the positional data of an arbitrary posting within this block.

Our analysis begins by considering the block-based list organization of Figure 2.1. Suppose that the inverted list I_t of a term t is partitioned into B_{I_t} blocks and each block $B_i \in B_{I_t}$ is comprised of S_{B_i} postings. Here we do not study the manner an inverted list is partitioned into blocks, however to render our proposals compatible with the existing

partitioning approaches, we assume that S_{B_i} is not equal for all blocks (i.e. the blocks of the list may include variable numbers of postings).

In the sequel, we identify the highest positional value $|p_{B_i}|_{max}$ for each block B_i of the inverted list. Based on $|p_{B_i}|_{max}$, we allocate a number of $C_{B_i} = \lceil log_2(|p_{B_i}|_{max} - 1) \rceil$ bits to produce a binary representation of each occurrence in that block, and we store these representations into a compressed storage.

A pseudocode demonstrating how PFBC is used to encode a bundle of K positional values is presented in Algorithm 1.

Algorithm 1 Encoding a bundle of K positional values with PFBC. After the identification of maximum positional value (steps 3-8), PFBC calculates C, which is the smallest number of bits required to encode all K integers (step 9). In the sequel, the function $write()$ in step 13 is used to store each p_i value into a compressed sequence $\mathcal{P}$ by using C bits.

```
     byte PFBC − Encode(K, p[K])
1.   int i ← 0, p_max ← 0, C ← 0
2.   byte P
3.   while (i < K) {
4.       if (p_i > P_max) {
5.           p_max = p_i
6.       }
7.       i++
8.   }
9.   C ← ⌈log_2(p_max − 1)⌉
10.  P ← allocate ⌈KC/8⌉ bytes
11.  i ← 0
12.  while (i < K) {
13.      write(p_i, P, C)
14.      i++
15.  }
16.  return P
```

The fixed bit compression methodology of PFBC is expected to introduce some compression loss in comparison to PForDelta. Actually, the latter encodes the largest integers of a list as exceptions and the rest of them by using a fixed-bit scheme, similar to the one we described. This operation is proved to be very effective in the case of docIDs, because in the docIDs blocks the number of large integers is small (due to the d-gap encoding). However, when P4D is applied at blocks of positional data, the benefits are diminished because in such blocks the number of large integers cannot be predicted. Indeed, as we demonstrate in our experimental section, PFBC is outperformed by P4D in terms of compressed index sizes by only a small margin.

2.3.2 Accessing the positional data with PFBC

Here we describe how PFBC can be employed to locate the positional data of a particular posting. Recall that our primary objective is to avoid any costly look-ups which will decelerate query processing. For this reason, we introduce a mechanism which allows us to *pre-compute* the location of the required data and decode it without touching any unnecessary data. Note that the current state-of-the-art block compression methods such as PForDelta and VSEncoding do not meet this requirement, since they are only capable of decoding bundles of integers.

To achieve this, it is required that we store two values for each block B_i of the list: (a) the aforementioned C_{B_i} value which denotes the number of bits we used to encode the positions belonging to the block B_i, and (b) a pointer R_{B_i} pointing at the beginning of the positional data of B_i. Exploiting this limited amount of information, we are able to calculate the location of the positional data for any posting belonging to B_i. The following equation provides the exact bit $\mathcal{S}_j$ where the positions associated to an arbitrary posting j start from:

$$\mathcal{S}_j = R_{B_i} + C_{B_i} \sum_{x=0}^{j-1} f_{x,B_i} \tag{2.1}$$

where f_{x,B_i} is the x^{th} frequency value stored within B_i. Consequently, to locate the positional data for an arbitrary posting j we first need to dereference the corresponding R_{B_i} pointer value. Then, we sum up all the frequency values of the previous postings; this sum reveals the number of the positional values stored between the beginning of the block and the location of the desired data. Since the compressed positions are stored by using a fixed number of bits, we just need to multiply the sum by C_{B_i} to locate the first compressed position of the posting. The operation ends by decoding the next $f_{j,B_i}C_{B_i}$ bits and the positions are retrieved. The algorithm used by PFBC is illustrated in Table 2.

Our proposed methodology exhibits a wide range of advantages over the adversary compression approaches:

- It facilitates direct access to the positional data by using equation 2.1. No expensive look-ups for positions in tree-like structures are required. Consequently, query processing is accelerated;
- It saves the space cost of maintaining a separate look-up structure [149], since the involved pointers can be stored within the skip table;
- It uses much fewer pointers than the indexed lists of Transier and Sanders [136]; and

Algorithm 2 Using PFBC to decode the positional data of the j^{th} posting of the block B_i. Initially, we sum up all the frequency values of the previous postings (steps 2-5) of B_i, and we locate the bit where the occurrences of the j^{th} posting start from (step 6). Then, we employ the $read()$ function to read each encoded position from the bit-vector $\mathcal{P}$, starting from different points each time (steps 8-12).

```
     int PFBC − Decode(j, B_i, P)
1.   int x ← 0, s ← 0
2.   while (x < j) {
3.       s ← s + f_{x,B_i}
4.       x + +
5.   }
6.   int start ← R_{B_i} + sC_{B_i}
7.   x ← 0
8.   while (x < f_{j,B_i}) {
9.       p[x] ← read(P, C_{B_i}, start)
10.      start ← start + C_{B_i}
11.      x + +
12.  }
13.  return p
```

- It enables decoding of the information actually needed, without the need to decompress entire blocks or sub-blocks of integers (unlike other compression techniques such as PForDelta [69]).

The final issue to address is to determine how the required R_{B_i} and C_{B_i} values should be stored. A straightforward solution is to create a special data structure for this reason. However, in that case additional space and a look-up operation would be required; consequently the benefits of our method would be partially limited. Therefore, we decided to use the skip structure (upper part of Figure 2.1) and for each entry of the table, we also record these two values. This strategy is both effective and efficient; no extra space is required, but only the room occupied by the values themselves. Furthermore, in case the query processor decides that a posting belonging to a particular block should be exhaustively evaluated by decoding its corresponding positional data, we are able to immediately access R_{B_i} and C_{B_i}.

Algorithm 2 includes a pseudocode which demonstrates how PFBC is used to decompress the positional data for a specific posting. Let us describe the basic steps of this algorithm. Assume that we need to access and decode the positions for the j^{th} posting which belongs to the block B_i of the inverted list. Initially, we accumulate all the frequency values of the previous $j-1$ postings of the block. In the sequel, we read the R_{B_i} and C_{B_i} values from the skip table and we locate the required data as indicated by Equation 2.1. If f_{j,B_i} is the associated frequency value of the j^{th} posting, we sequentially read f_{j,B_i} groups of C_{B_i}

bits from the compressed sequence; each group represents an encoded positional value of this posting.

2.4 Document zones

There are several features that differentiate Web documents from plain textual documents: the former include hyper-links allowing the reader to quickly navigate from one page to another, and they also possess a visual structure determined by the usage of HTML tags. A typical Web page usually includes a title, some anchor text associated with its outgoing links, headings and other locations of special interest such as meta-tags and URL. Apparently, the appearance of a word in different locations of such a document is of different importance. For instance, the words used in a document's title usually represent its content and an effective search engine should treat these words in a different way than the ones occurring in the normal text.

The structure of the Web documents have gained very little attention by the inverted index researchers and engineers. The vast majority of the relevant work takes into consideration only the position of a word to describe its occurrence within a document. One exception to this rule is the early work of [30] which introduced plain and fancy *hits* to identify words appearing in a document's text and title respectively. However, this work ignores the rest of the physical locations of a document whereas it limits the maximum position value to an upper bound of 12 bits.

To address this problem, [92] partitioned each document into several parts of special interest called *fields* or *zones*. Zones are distinct, arbitrary locations of a Web document containing free text and delimited by page formatting tags. It is not mandatory to be contiguous (they can span across multiple locations of the document), but they cannot overlap. Unfortunately, the aforementioned work does not address the issue of the efficient representation of zones within the inverted index.

Table 2.1 records some the most typical zones that a Web page can be partitioned into. Each of these zones is assigned a unique integer value, the *zoneID*. Notice that for the case we are studying (Web pages), we consider these eight zones only. Nevertheless, with a slight modification our approach can support more zones and larger zoneIDs.

In this chapter we examine how zones can be used to improve the retrieval effectiveness of a search engine. Moreover, we study efficient methodologies of including them within the inverted index, the primary data structure that search engines employ to answer user queries.

Zone	zoneID	HTML
Body (Normal Text)	0	`<body>...</body>`
Anchor Text	1	`<a>...</a>`
Title Text	2	`<title>...</tile>`
Document's URL	3	–
Headings	4	`<h1>...</h1>,<h2>...</h2>,...`
Page Description	5	`<meta name="Description" Content="...">`
Image Description	6	`<img alt="..." ...`
Label Text	7	`<label>...</label>`

Table 2.1: Zones of a typical Web page

2.5 Integrating zones within the inverted index

We now show how a typical inverted index can be enriched by including zones. Within each posting S_i of the inverted list I_t of a term t, we replace the positional data with a more general quantity $h_{i,j}$, describing the j^{th} *occurrence* of t within a document d_i. Therefore, the new form of the posting S_i is:

$$S_i : (d_i, f_i, h_{i,0}, h_{i,1}, ..., h_{i,f_i-1}) \tag{2.2}$$

Since we desire to describe the occurrence of a term by using both positions and zones, $h_{i,j}$ is of the form:

$$h_{i,j} : p_{i,j}, z_{i,j} \tag{2.3}$$

that is, each term occurrence is now described by a *position-zone* pair and not by just using positional values. From now on, we shall use the word occurrence to refer to such position-zone pairs. Compared to a typical inverted index, this enriched form provides additional features and functionality, since:

- It is capable of answering a wider range of queries, for example "*Retrieve all the documents having the terms University AND Thessaly in title AND the term gr in their URL*"
- We can exploit more sophisticated and effective ranking functions such as the BM25F [89], or fabricate other robust scoring approaches by taking the additional parameters into consideration (see chapter IV).

The inclusion of additional information makes index organization and compression more challenging. The requirements of Web-scale search engines include compactness

(i.e. the new data must be stored as effectively as possible) and speed (i.e. we must be able to quickly access and decode the index data). In the following subsections we discuss further these requirements.

2.6 TZP: compression of zoneID-position pairs

In this section we introduce TZP, an efficient algorithm for encoding the position-zone pairs of the postings of equation 2.3. Initially we develop the encoding algorithm and in the sequel, we show how we can access and decode the occurrence data of a particular posting.

2.6.1 Compressing zoneIDs and positional data with TZP

Inevitably, the inclusion of zoneIDs in an inverted list will lead to an increase of the overall size of the index. In this subsection we study how we can minimize this effect and preserve high decompression rates.

Let us return to the posting scheme described by Equations 2.2 and 2.3. A naive approach for encoding each term occurrence (Equation 2.3) would suggest using two separate integers, one for storing the positional value and one for storing the zoneID. Undoubtedly, this approach is prohibitively expensive since we cannot afford wasting 64 bits[1] for each term occurrence.

For this reason, we suggest a two-level compression scheme, namely *TZP*, which initially encodes one position-zone pair into a 32-bit space, and then, for each block of the inverted list, it employs a fixed number of bits to encode the occurrences of that block. Fixed-bit compression such as Packed Binary [20] has two major advantages; at first, it allows fast decompression since the decoding process is fairly simple and second, it enables access to the positional data of a particular posting without look-ups and without decoding useless data. We shall defend these claims shortly, especially the second one which is extremely important since look-ups could drastically decelerate query processing [148].

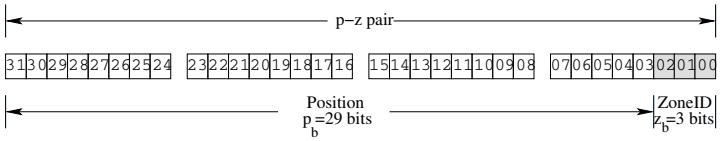


Figure 2.2: First phase of TZP; Encoding a single term occurrence (position-zone pair) in a 32-bit space for $z_b = 3$ and $p_b = 29$.

Our analysis begins again by considering the block-based list organization of Figure

[1] in this work we assume that each integer occupies 32 bits.

2.1. Now we demonstrate how the occurrences $h_{\mathcal{B}_i}$ included in the postings $S_{\mathcal{B}_i}$ of a block $\mathcal{B}_i$ can be efficiently encoded by using our proposed TZP approach. During the first phase and for every term occurrence $h_j \in h_{\mathcal{B}_i}$, we reserve the z_b least significant bits of a 32-bit integer to encode the zoneID z_j, whereas the rest $p_b = 32 - z_b$ bits are used to represent the positional value p_j. In this work we mainly focus on standard Web pages with eight zones at most, thus a setting of $z_b = 3$ suffices. However, for different types of documents alternative setups could be selected. Finally, applying this method to all the occurrences of the block results in a sequence of new integers having values equal to $|h_j| = 2^{z_b} p_j + z_j$.

At the second phase, we first select the highest occurrence value $|h_{\mathcal{B}_i}|_{max}$ for each block $\mathcal{B}_i$ of the inverted list. In the sequel, we use $C_{\mathcal{B}_i} = \lceil log_2(|h_{\mathcal{B}_i}|_{max}) \rceil$ bits to produce a binary representation[2] of each occurrence in that block, and we store these representations into a bit vector. This operation is very similar to the Packed Binary approach.

A pseudocode demonstrating how TZP is used to encode position-zone pairs is presented in Algorithm 3. The operators that we use in our algorithm representation, include bitwise AND ($\&$), bitwise OR ($|$), left-shift of a value v by b bits ($\mathscr{L}_b(v)$) and right-shift of value v by b bits ($\mathscr{R}_b(v)$).

TZP is not designed to operate on single integers; we can rather think of it as a method which effectively compresses two correlated integers from which one of them has an upper bound. In our examined occasion, we restrict zones to occupy only three bits. TZP can be generalized to encode more than two integers. Therefore, if we have N integers which all describe a single phenomenon (i.e. term occurrence in a document) and $N - 1$ of them can be restricted to have an upper bound and relatively small values, TZP provides an efficient mechanism for their compression.

2.6.2 Accessing zoneIDs and positional data with TZP

Now we show how our fixed-bit encoding allows access to the occurrences of a particular posting without look-ups. For each block $\mathcal{B}_i \in \mathcal{B}_{I_t}$ of an inverted list, we associate two values: (a) a pointer $R_{\mathcal{B}_i}$ which points at beginning of the occurrence data of $\mathcal{B}_i$, and (b) a value $C_{\mathcal{B}_i}$ which represents the fixed number of bits used to encode each occurrence of $\mathcal{B}_i$. With this information we can *compute* the location where the occurrences of a particular posting $S^j_{\mathcal{B}_i}$ start from, and avoid searching for it during query processing. The following equation provides the exact bit where the occurrences of $S^j_{\mathcal{B}_i}$ start from:

$$\mathcal{L}^j_{\mathcal{B}_i} = R_{\mathcal{B}_i} + C_{\mathcal{B}_i} \sum_{x=0}^{j-1} f_{x,\mathcal{B}_i} \tag{2.4}$$

[2]Recall that we are able to encode every positive integer ranging from $0 \ldots N-1$ by using $\lceil \log_2 N \rceil$ bits.

Algorithm 3 Encoding a bundle of K position-zone pairs with TZP. The function $Encode\_Occurrence$ encodes a single pair with respect to the number of bits z_b that we reserve for zones (in our case $z_b = 3$). $Encode\_Occurrences$ compresses an entire bundle of K position-zones pairs; after the the maximum occurrence value h_{max} and the number of the required bits C have been determined (steps 9-10), we store the binary representation of each occurrence within the bit vector $\mathcal{H}$ (steps 12-15). $WriteBits$ is a typical bit-writer function which stores an integer into $\mathcal{H}$ by using C bits.

int $Encode\_Occurrence\ (p, z, z_b)$

1. **int** $h \leftarrow 0$
2. $h \leftarrow \mathscr{L}_{z_b}(p)$
3. $h \leftarrow h \mid z$
4. **return** h

byte $Encode\_Occurrences(p[K], z[K], K)$

1. **int** $i \leftarrow 0, h_{max} \leftarrow 0$
2. **while** $(i < K)$ {
3. $\quad h[i] = Encode\_Occurrence\ (p[i], z[i], 3)$
4. $\quad$ **if** $(h[i] > h_{max})$ {
5. $\quad\quad h_{max} = h[i]$
6. $\quad$ }
7. $\quad i++$
8. }
9. **int** $C \leftarrow \lceil \log_2(h_{max} - 1) \rceil$
10. **byte** $\mathcal{H} \leftarrow$ **allocate** $\lceil KC/8 \rceil$ bytes
11. $i \leftarrow 0$
12. **while** $(i < K)$ {
13. $\quad WriteBits(\mathcal{H}, h[i], C)$
14. $\quad i++$
15. }
16. **return** $\mathcal{H}$

where $f_{x,\mathcal{B}_i}$ is the x^{th} frequency value stored within $\mathcal{B}_i$. Equation 2.4 informs us that to locate the occurrence data for a particular posting, we first need to retrieve the associated $R_{\mathcal{B}_i}$ pointer value; then, we sum up all the frequency values of the previous postings of the current block $\mathcal{B}_i$. This sum of frequencies reveals the number of the occurrences between the beginning of the block and the location of the desired data. Since each of these occurrences is represented by a fixed number of bits, we just need to multiply the sum by $C_{\mathcal{B}_i}$ to locate the first *compressed* occurrence of the posting. The operation ends by decoding the next $f_{j,\mathcal{B}_i} C_{\mathcal{B}_i}$ bits and the occurrences are retrieved.

Similarly to PFBC, our proposed methodology has a series of advantages over other organization approaches:

- It allows us to directly access the desired data by using Equation 2.4 without look-

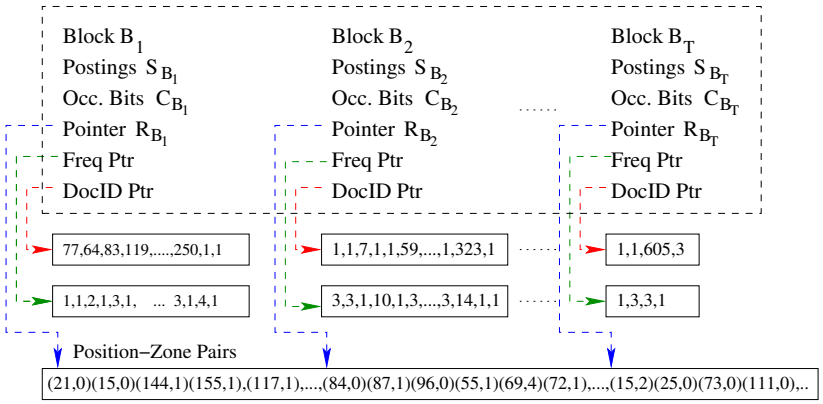


Figure 2.3: Partitioning an inverted list into T blocks of postings (block-based organization) according to TZP. The positional and zone data are packed separately at the end of the inverted list. For each block $\mathcal{B}_i$ of the list, we store within the skip table (a) a pointer $R_{\mathcal{B}_i}$ pointing at the starting bit of the corresponding occurrence data and (b) the number of bits $C_{\mathcal{B}_i}$ used to encode each occurrence of the block.

Algorithm 4 Decoding position-zone pairs for the j^{th} posting of a block $\mathcal{B}_i$ according to TZP. Initially we sum up all the frequency values of the previous postings (steps 2–5) of the current block $\mathcal{B}_i$, and we locate the bit where the occurrences of the j^{th} posting start from (step 6). Then, we employ the $ReadBits$ function to read each encoded occurrence from the bit-vector $\mathcal{H}$, starting from different points each time (steps 8–12). In the second phase, we extract the positional and zone data out of each occurrence (steps 14–18).

int $Decode\_Occurrences(j, \mathcal{B}_i, \mathcal{H})$

1. **int** $x \leftarrow 0, s \leftarrow 0$
2. **while** $(x < j)$ {
3. $\quad s \leftarrow s + f_{x,\mathcal{B}_i}$
4. $\quad x++$
5. }
6. **int** $start \leftarrow R_{\mathcal{B}_i} + sC_{\mathcal{B}_i}$
7. $x \leftarrow 0$
8. **while** $(x < f_{j,\mathcal{B}_i})$ {
9. $\quad h[x] \leftarrow ReadBits(\mathcal{H}, C_{\mathcal{B}_i}, start)$
10. $\quad start \leftarrow start + C_{\mathcal{B}_i}$
11. $\quad x++$
12. }
13. $x \leftarrow 0$
14. **while** $(x < f_{j,\mathcal{B}_i})$ {
15. $\quad z[x] \leftarrow h[x]$ & 0x00000007
16. $\quad p[x] \leftarrow \mathscr{R}_3(h[x])$
17. $\quad x++$
18. }
19. **return** p, z

ups. Consequently, query processing is accelerated.

- It saves the space cost of maintaining a separate look-up structure [148].
- It uses much fewer pointers than the approach of [136] which employs indexed lists.
- It enables decoding of the information actually needed, without the need to decompress entire blocks or sub-blocks of integers (unlike other compression techniques such as PForDelta [69, 160]).

Another important issue is to determine where we should store $R_{\mathcal{B}_i}$ and $C_{\mathcal{B}_i}$. A convenient location is within the skip structure (upper part of Figure 2.3); for each entry of the table, we also record these two values and in case the processor decides that a block should be scanned, we are able to immediately retrieve them.

Similarly to encoding, the decoding process of TZP also involves two phases. Algorithm 4 includes a pseudocode describing the entire operation. Suppose that we need to access the positional and zone data for the j^{th} posting of block $\mathcal{B}_i$. Initially, we locate the data as indicated by Equation 2.4. If $f_{j,\mathcal{B}_i}$ is the associated frequency value of $S^j_{\mathcal{B}_i}$, we sequentially read $f_{j,\mathcal{B}_i}$ groups of $C_{\mathcal{B}_i}$ bits from the compressed sequence; each group represents an encoded occurrence of this posting. In the next phase, we extract the desired zone and positional data out of each occurrence.

2.7 Experiments

In this section we attest the proposed PFBC and TZP approaches against some of the most successful general-case compression methods. The experimentation is divided in two phases where we measure the compression effectiveness and the query processing efficiency of our two approaches. We mainly focus on presenting the space and time benefits deriving from the usage of PFBC and TZP in terms of both compression effectiveness and speed of data access and decompression.

All the results we present in this work are obtained by using a machine equipped with a CoreI7 920@2.66 GHz processor (having the additional processing cores and Hyper-Threading disabled) and 12 GB of RAM. The system was running the 64-bit distribution of Ubuntu Linux 10.04 LTS. The sample document collection we employed in our experiments is the Clueweb09-T09B data set, which is a subset of a larger collection, Clueweb09. This subset consists of 50,220,423 pages written in English and occupies approximately 1.42 TB in uncompressed form.

In the real-world Web search engines, each machine typically searches a subset of the collection, consisting of up to tens or hundreds of millions of Web pages [154]. Each query server maintains its own part of the index, hence the entire index is partitioned into shards [40]. Since in this work we desired to simulate a large-scale search system as precisely as possible, we follow a similar strategy for constructing the index and we provide separate measurements for each shard. In total, the generated index structures in this work consist of ten shards.

2.7.1 Experimental index setups

To achieve unbiased measurements for PFBC we created three separate positional indexes having their docIDs and frequencies organized in the same manner. In particular, we fragmented each inverted list into blocks of 128 postings and we employed P4D to encode the d-gaps and the associated frequencies. In the first two experimental index setups, we applied the state-of-the-art position organization and compression methods encountered in the literature. Namely,

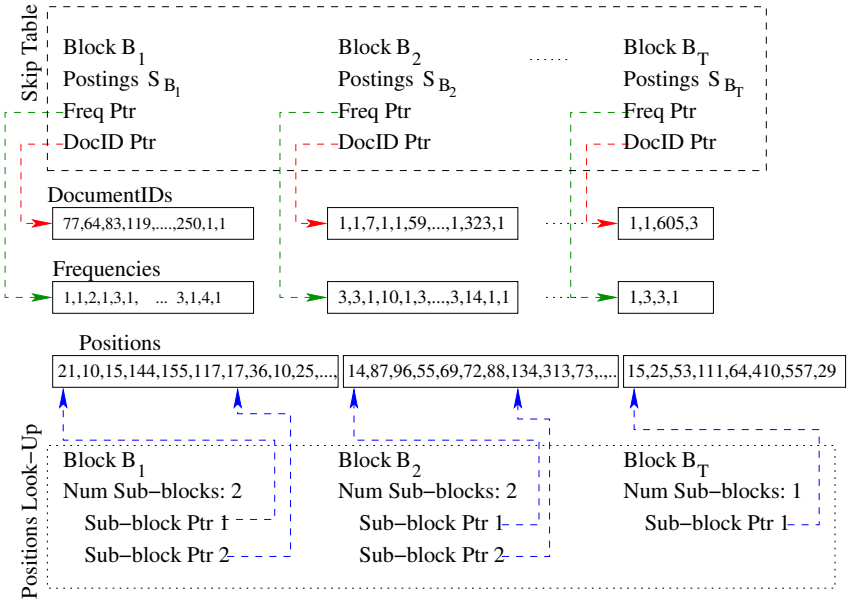


Figure 2.4: The adversary block-based organization approaches. The positional data are encoded by using either OptP4D, or VSEncoding. The positions look-up structure stores pointers which point at the beginning of the positional sub-blocks. Each sub-block consists of 128 positional values, unless it is defective.

- The strategy proposed by [149], i.e. the positions are compressed with OptP4D and accessed by using a separate look-up structure. The original P4D algorithm encodes

integer groups having sizes that are multiples of 32 [69]. Typical implementations employ P4D to encode bundles of 128 values; however, there are occasions where we have fewer than 128 integers to encode. In such cases the encoder constructs a *defective* compressed block of integers [26] by filling up the empty space with dummy entries. These dummy entries introduce a remarkable overhead which in turn leads to compression losses. To reduce these losses, [149] propose a variant of P4D, namely OptP4D, which automatically adapts to under full chunks by using another compression method such as Simple16. To locate the positional data of a particular posting, we need a set of pointers pointing at multiple points in the compressed positions and a look-up structure similar to the one described in [149].

- The positions are encoded according to VSEncoding [129]. In contrast to the other encoding schemes, this one automatically determines the size of each block by computing the optimal manner that a list of integers should be partitioned, with respect to the value of a cost function. In the sequel, VSEncoding encodes the integers of each block by utilizing a fixed number of bits (according to the largest value in the block). Although VSEncoding is found to perform well in decompressing docIDs, in this work we use it to encode the positional data of an inverted list with the aim of comparing it against PFBC. Regarding the issue of organization, we observed that, due to the random distribution of the positional values, VSEncoding constructs a large number of small blocks of integers. In particular, for each block of an inverted list, the algorithm in question organizes the respective positional data by constructing on average 41 sub-blocks; consequently, for the entire inverted index we need to waste more than 20.3 GB of space just to store pointers to each of these sub-blocks. For this reason, we store pointers in a way identical to the one we apply at the OptP4D case (that is, we set pointers per 128 compressed positions).

These organization strategies are illustrated in Figure 2.4. In the following experiments we compare them against our introduced PFBC approach in terms of both space costs and consumed times during query processing.

Regarding TZP, we expanded both of the aforementioned index organization approaches with the aim of supporting zones. In particular, for each block of the inverted list we appended a fourth chunk as illustrated in Figure 2.5. This fourth chunk is used to store the encoded sequence of the zoneIDs. To locate the positional and zone data we employ a look-up structure similar to the one proposed in [149]. The difference is that apart from the regular pointers showing the location of the positional data, it is also required to store an equal number of pointers pointing at the respective zone data.

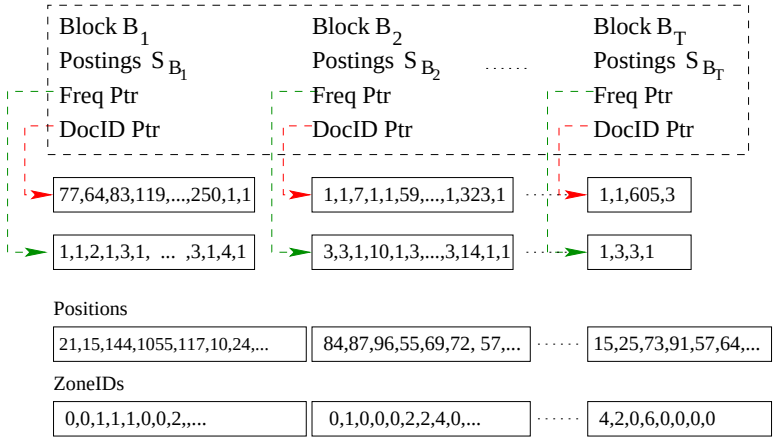


Figure 2.5: Expanding the partitioned inverted list of Figure 2.1 to store zones. Apart from the three standard chunks which store the docIDs, frequencies, and positions, we allocate one more to store the desired values.

Finally, to demonstrate the overall growth in the size of an index caused by the inclusion of zones, we also constructed a standard positional index for our experiments. The setup we selected is similar to the one of Figure 2.1. DocIDs and frequencies are encoded by using the P4D method, whereas the positional data are organized by using interleaving and compressed by applying the OptP4D variant.

2.7.2 Compression effectiveness of PFBC

Now let us evaluate the performance of PFBC against the adversary state-of-the-art approaches in terms of compression effectiveness. Apart from the size of the inverted file, we also measure the space occupied by the accompanying data structures required for efficient query processing (i.e. skip table, pointers to positions, and position look-up structure).

In Table 2.2 we record the overall space requirements of each index setup that we examine. The first row concerns the accumulated inverted file size for all the ten shards comprising our index. The next rows illustrate the sizes of the accompanying data structures. Notice that each organization approach does not make use of all data structures. For instance, our PFBC approach does not require the existence of a positions look-up structure, whereas all strategies employ a skip table. The absence of a data structure is denoted by using a dash symbol.

Among our examined encoding algorithms, VSEncoding achieved the best compression performance; the ten inverted files of all shards occupy in total roughly 90.2 GB. On the other hand, OptP4D performed imperceptibly worse resulting in an inverted file which

Data Structure	OptP4D	VSEncoding	PFBC
Inverted Index	90.8	90.2	92.0
Skip Table	1.7	1.7	1.7
Pointers to positions	-	-	2.1
Positions look-up	4.1	4.1	-
Total	96.6	96.0	95.8

Table 2.2: Overall space requirements (expressed in GB) of our experimental index setups

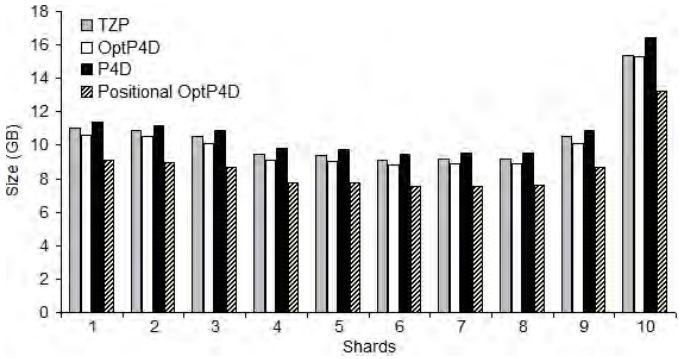
occupied less than 1% more space. As we anticipated, the fixed-bit compression strategy adopted by PFBC introduced some slight losses. Compared to VSEncoding, the inverted files of PFBC occupied in total approximately 2% more space.

Furthermore, we examine the sizes of the data structures which accompany each index organization scheme. The PFBC scheme suggests storing the positional data in a contiguous manner (i.e. without interleaving) and maintaining only one pointer per block to access the positional data of a particular posting. This pointer, along with the corresponding value representing the number of bits we use to encode the positional values of the block, are stored within the skip table itself. However, OptP4D and VSEncoding encode groups of elements hence in an interleaving scheme a block can contain multiple positional sub-blocks. For each of these sub-blocks, it is required that we store a separate pointer to be able to access their positional data. In other words, we are usually obliged to store multiple positional pointers for each block of the inverted list. These pointers are kept within the separate look-up structure we describer earlier.

In Table 2.2 we report the sizes of the auxiliary data structures for all of the 10 shards of our examined indexes. The skip table which includes the positional pointers (recall the upper part of Figure 2.4) is much more economic than the look-up structures themselves. As a matter of fact, this data structure occupies approximately 65% of the space occupied by the data structures of the OptP4D and VSEncoding approaches (skip table plus the positions look-up structure = 5.8 GB). In the last row of Table 3 we present the overall index sizes (inverted file plus auxiliary data structures) for each of the examined schemes. In conclusion, we notice that the superiority of OptP4D and VSEncoding over PFBC in terms of compressed sizes is compensated by the significantly smaller data structures that accompany our proposed index scheme. As a result, PFBC presents marginal savings of 0.02-0.08%.

2.7.3 Compression effectiveness of TZP

In this subsection we record the inverted file sizes for both positional and enriched indexes, to evaluate the benefits deriving from the usage of TZP. In the sequel, we measure



Index Type	Size (GB)
TZP	104.68
OptP4D	101.30
P4D	108.88
Positional OptP4D	92.91

Figure 2.6: Compressed inverted file sizes per shard (Left), and total index sizes for all ten shards (Right) for our four experimental setups.

the space occupied by the accompanying data structures, that is, the skip table and the positions (and zones) look-up structure.

2.7.3.1 Compressed inverted file sizes

The left part of Figure 2.6 illustrates the inverted file sizes expressed in GB for each of the ten constructed shards. Furthermore, in the right part of the same Figure, we present the overall index sizes for each organization approach.

The comparison of TZP against P4D and OptP4D shows that our method is more effective than the plain P4D approach. P4D encodes groups of 128 integers; for each group, we select a parameter b in a manner that the large majority of the data elements (i.e. 90%) can be coded by using b bits. The rest of the elements are called *exceptions* and are coded by employing 8, 16, or 32 bits. On the other hand, TZP encodes the word occurrences by using a fixed number of bits, regardless of their values. Therefore, one could expect that P4D would perform better than TZP.

Nevertheless, this is not valid, since P4D cannot deal effectively with the groups that have fewer than 128 elements each (defective groups). In these groups, the empty space is padded with dummy entries and this leads to significant wasted space. As we already mentioned, OptP4D addresses this problem by using P4D to encode the regular groups containing 128 values, but instead switches to a different code for the defective ones.

In comparison with P4D, TZP produced an index that is about 3.9% smaller. On the other hand, OptP4D outperformed our method by a margin of 3.2%. However, both P4D and OptP4D require a look-up structure in order to identify the positional and zone data of a particular posting. As we will show shortly, this structure has remarkable space requirements, therefore the overall space occupied by these indexes is increased.

The standard positional index occupies in total approximately 92 GB. In comparison

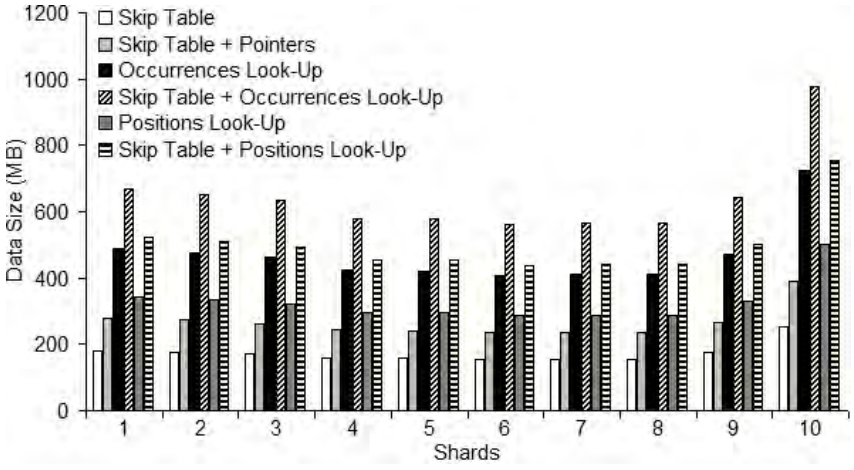


Figure 2.7: Sizes of the auxiliary data structures used by our four examined indexes per shard.

to the other three approaches, we conclude that the inclusion of zones within the inverted index leads to an increase to the occupied space by a margin that fluctuates between 8.3% (for OptP4D) and 14.7% (for P4D).

2.7.3.2 Sizes of the auxiliary data structures

In this subsection we examine the space requirements of the auxiliary data structures. The auxiliary structures include the skip table which allows us to partially access and decode the inverted list data during query processing, and the positions look-up structure which enables us to access the positional and zone data for a particular posting. Recall that P4D and OptP4D require both of these structures, whereas TZP is accompanied only by the skip table of Figure 2.3.

Our TZP approach suggests storing the occurrence data contiguously (i.e. not interleaving) and maintaining only one pointer per block to locate the desired data per posting. However, P4D and OptP4D encode groups of 128 elements, hence, in an interleaving scheme, a block can contain multiple positional and zone chunks. For each of these chunks, it is necessary to store a separate pointer to be able to access their data. In other words, we are usually obliged to store multiple positional and zone pointers for each block of the inverted list.

Figure 2.7 depicts the sizes of the auxiliary data structures for all of the 10 shards of our examined indexes. Moreover, in the Table 2.3 we present the overall auxiliary data structure sizes for all ten shards. In the second column of this Table we record the type of the index which makes use of a specific data structure. For instance, the TZP approach

Data Structure	Index	Size (GB)
Skip Table	All except TZP	1.69
Skip Table + Pointers	TZP	2.60
Positions Look-Up	Positional OptP4D	3.21
Skip Table + Positions Look-Up	Positional OptP4D	4.90
Occurrences Look-Up	P4D/OptP4D	4.58
Skip Table + Occurrences Look-Up	P4D/OptP4D	6.27

Table 2.3: Total auxiliary data structure sizes for all ten shards.

exploits the skip table along with the pointers we described in subsection 2.6.2, whereas OptP4D employs both the skip table and the occurrence look-up structure.

Notice the difference between the occurrence and position look-up structures. The former stores pointers pointing at the positional and zone data, whereas the latter maintains pointers pointing only at the positional data. The inclusion of these additional zone pointers causes an increase of about 30% to the look-up structure (3.21 GB vs 4.58 GB).

The skip table which includes pointers (recall the upper part of Figure 2.3) is much more economic than the look-up structures themselves. As a matter of fact this data structure occupies approximately 53% of the space occupied by the data structures of the positional P4D approach (skip table plus the positions look-up structure, 4.9 GB) and only the 41.4% of the space occupied by the auxiliary structures of the P4D approach (skip table plus the occurrences look-up, 6.27 GB).

In Table 2.4 we present the overall index sizes (inverted file plus auxiliary data structures) for each of the four examined schemes. We notice that the superiority of the OptP4D over TZP in terms of compressed sizes is compensated by the significantly smaller data structures that accompany our proposed index scheme.

Index Type	Inverted Files	Data Structures	Total
TZP	104.68	2.60	107.28
OptP4D	101.30	6.27	107.57
P4D	108.88	6.27	115.15
Positional OptP4D	92.91	4.90	97.81

Table 2.4: Total inverted index sizes expressed in GB, for all ten shards.

2.7.4 Query throughput in positional indexes with PFBC

In this subsection we examine the performance of PFBC against the adversary approaches in terms of speed during query processing. This issue is of critical importance since high query evaluation speeds lead to increased query throughput for the entire search

system. Our experimentation was divided in two parts: on the first part we submitted a set of 50 conjunctive queries and measured several statistics such as the decompression times and the size of the accessed data. On the second part, we repeated the previous experiment by submitting the same 50 queries expressed as exact phrase searches.

For the requirements of the first stage we have implemented the two stage query processing method discussed earlier. Therefore, during the first phase we traverse the inverted lists of the query terms by employing the document-at-a-time strategy, and we quickly identify the most relevant results by accessing docIDs and frequency values only. In the second phase we apply more complex ranking schemes such as BM25TP [31] to the best results determined in the previous stage by retrieving the positional values. We experimented with two values of K ; the first one is $K = 200$ and was selected because in [149] the authors prove that higher values do not lead to any further precision gains. Furthermore, since the major Web search engines return at most 1000 results, we also choose to set $K = 1000$.

For the needs of this experiment we employed the set of queries of the Web Adhoc Task of the TREC-2009 Web Track. This set consists of 50 distinct queries; each query contains on average two terms (102 terms in total). Since the inverted indexes we constructed were comprised of ten shards, we repeated our experiments ten times; each time the query processor was assigned a different index shard.

For all the compression methods, we measure several statistics including the number of the positional values we access and decode, along with the corresponding decompression rates. We also record the latency of the look-up operation; that is, the time consumed by each approach to locate the positional data for each posting. The results of our experiments are illustrated in Table 2.5.

K		**OptP4D**	**VSEncoding**	**PFBC**
$K = 200$	Decompressed positions	2,756,128	2,756,128	374,251
	Decompressed positions/query	55,123	55,123	7,485
	Total access time (msec)	0.11	0.11	0
	Total decompression time (msec)	5.56	5.14	1.01
	Average time per query (msec)	0.11	0.10	0.02
$K = 1000$	Decompressed positions	11,192,608	11,192,608	1,044,234
	Decompressed positions/query	223,852	223,852	20,885
	Total access time (msec)	0.31	0.31	0
	Total decompression time (msec)	23.94	21.10	4.02
	Average time per query (msec)	0.48	0.42	0.08

Table 2.5: Positional data access and decompression times per query and per posting for different values of K.

Table 2.5 is divided in two parts; the upper part contains the results we recorded for $K = 200$, whereas the lower one includes the results for $K = 1000$. The first line repre-

sents the total number of positional values accessed by each method for all the ten index shards, whereas the second line shows the number of the decompressed positions per query. PFBC outperforms the adversary approaches by a significant margin, since the fixed-bit compression scheme allows us to locate exactly the data we need to access and we do not have to decode entire blocks of integers. In total, the organization method with the look-up structure employed by OptP4D and VSEncoding touched approximately 7.4 times more data than the one applied by PFBC for $K = 200$. In case we set $K = 1000$, PFBC is even more efficient, since the other methods decode about 10.7 times more data.

The next line reveals the average position look-up times consumed by each method per query. The algorithm of Table 2 allows PFBC to calculate the location of the positional data without searching for it, consequently, the latency is nullified in this case. Regarding the other two approaches which employ the aforementioned look-up structure, they introduce a latency of about 0.11 msec per query for $K = 200$ and 0.31 msec for $K = 1000$. Note here that the look-up times do not scale proportionally to the value of K (the value of K increases fivefold whereas the latency tripled). This is explained by the fact that for some of our test queries, the query processor returns fewer than 1000 results.

Now let us examine the decompression rates achieved by each method. The lines 3 and 7 of Table 2.5 include the total amount of time required to decode the positional values for all the 50 queries of our experiment. Furthermore, lines 4 and 8 reveal the average decompression time per query. On average, VSEncoding outperformed the OptP4D approach by a margin ranging between 1% and 2% for different values of K. PFBC was the fastest among the evaluated schemes, since it achieved about 5 times faster decompression compared to VSEncoding for both settings of K. This is mainly due to the limited number of positional data it decodes per query and due to the fixed-bit scheme.

In the sequel, we repeated the previous experiment by submitting the same queries as exact phrase searches. In other words, the documents identified as results, not only must contain the query terms, but also, the query terms must appear as an exact phrase (or sentence) within them. To process such phrase queries, it is essential that we retrieve all the positions for each candidate document; then, a candidate document is identified as a result only in the case the difference between the positional values of the postings is equal to the positional difference expressed in the query.

Since we must access all the positional values for all postings, there is no meaning in employing the two-stage query processing method applied in the earlier experiment. Instead, the query is answered in a single step by locating and decoding all the involved positional data. For this reason, these queries are much more intensive than the simple conjunctive queries. However, since as many as 10% of web queries are phrase queries and

many more are implicit phrase queries (such as person names), [92], a search engine must support this functionality in a both effective and efficient manner.

	OptP4D	**VSEncoding**	**PFBC**
Decompressed positions	2,374,318,680	2,374,318,680	89,143,576
Decompressed positions/query	47,486,373	47,486,373	1,782,872
Total access time (msec)	62.86	62.86	0
Total decompression time (msec)	7,194.36	6,874.92	1,066.14
Average time per query (msec)	143.88	137.50	21.32

Table 2.6: Positional data access and decompression times per query and per posting in case the queries are submitted as exact phrase searches.

Comparing the results of Table 2.5 and Table 2.6 we notice the huge increase in the amount of the accessed positional data; in contrast to the two-stage query processing scheme with $K = 1000$, PFBC now touches 85 times more data. On the other hand, the OptP4D and VSEncoding methods have to decode approximately 860 times more positions. PFBC is still much more economic than the adversary approaches, since it is required that we decompress roughly 27 times fewer positional values.

We also notice a great increase in the look-up and decompression times. OptP4D and VSEncoding consumed about 63 milliseconds to locate the required positional values, in contrast to PFBC, which is still free on any look-up latencies. PFBC spent more than 1 second to decode all the positional values for all 50 exact phrase queries, leading to an average decompression time of 21.32 msec per query. The other block-based compression schemes were outperformed by a significant margin. OptP4D and VSEncoding consumed about 7.2 and 6.9 seconds to decompress the involved positional values for all the 50 queries respectively (average decompression times per query are 143.88 and 137.50 msec respectively).

2.7.5 Query throughput in enriched indexes with TZP

In this subsection we discuss the performance gains deriving from the omission of the occurrence look-up structure in enriched indexes. The positional and zone data are required during the second phase of query processing therefore, by avoiding to constantly seek for these values leads to significant benefits during this phase only. Recall that TZP allows direct access to the occurrence data without look-ups and moreover, it enables decompression only of the data actually needed during query processing.

To measure the imposed time penalty during query processing, we have implemented this look-up structure for positions and zoneIDs. In Figure 2.8 we illustrate the average times consumed to search for the desired data per query, when K is assigned different values.

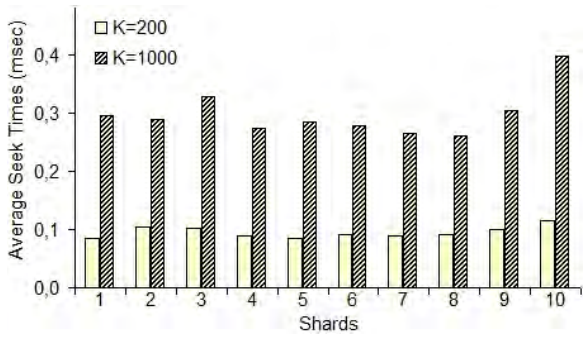


Figure 2.8: Occurrence look-up structure: Average seek times for positional and zone data per query and per shard.

	$K = 200$	$K = 1000$
Queries	50	50
Query Terms	102	102
Postings	19,568	89,969
Postings per Query	391.36	1,799.38

Table 2.7: Number of postings involved in the second phase of query processing for $K = 200$ and $K = 1000$.

The average seek times per query and per shard are 0.1 and 0.3 milliseconds for $K = 200$ and $K = 1000$, respectively. Similarly to the positional index situation, notice that the average times do not scale proportionally to the values of K, since the value of K increases fivefold whereas the times tripled. This is explained by the fact that some of our test queries produce fewer than 1000 results.

The seek penalty illustrated in Figure 2.8 is avoided by employing TZP and the methodology described in subsections 2.6.1 and 2.6.2. Recall that our proposed techniques allow us to pre-calculate the location where the desired data is stored. Hence, no look-up operations are required at the second phase and the speed gains are significant, especially in the case where the system receives thousands of queries per second.

Until now we have shown that the avoidance of occurrences look-ups during query evaluation offers an acceleration to the entire operation. We now demonstrate how our model provides faster occurrence decompression and leads to even better performances. Initially, in Table 2.7 we present the number of postings processed in the second phase of query evaluation, for $K = 200$ and $K = 1000$, for all the 50 queries that we have submitted to the system.

In subsection 2.6.2 we discussed that when encoding occurrences, TZP dominates over the group-based compression schemes such as P4D because we can access only the data

K		**TZP**	**OptP4D**	**P4D**
$K = 200$	Decompressed Occurrences	374,251	2,756,128	2,834,432
	Total Decompression Time (msec)	1.64	8.57	8.34
	Average Time per Query (msec)	0.0328	0.1714	0.1668
	Average Time per Posting (msec)	0.0838	0.4379	0.4262
$K = 1000$	Decompressed Occurrences	1,044,234	11,192,608	12,394,752
	Total Decompression Time (msec)	6.04	35.92	34.07
	Average Time per Query (msec)	0.1208	0.7184	0.6814
	Average Time per Posting (msec)	0.0671	0.3992	0.3787

Table 2.8: Occurrence decompression times per query and per posting, for $K = 200$ and $K = 1000$.

actually required and we do not need to decode entire groups of possibly unnecessary values. On the other hand, in case P4D is selected, we need to decode an entire block of 128 elements, even if only one value is required to process the query.

In Table 2.8 we record the values of several interesting characteristics of the second phase of query processing for our three examined index setups. The first four rows of the table concern the setting $K = 200$, whereas the next four rows represent our measurements for $K = 1000$.

The first row shows the sum of the decompressed occurrences for all 50 queries of our test set. TZP accesses much fewer values than the other approaches since in general, it decodes 7 to 12 times less data that P4D and OptP4D. Hence, we expect that our method would decode the desired information much faster than its opponent setups.

Indeed, in case we set $K = 200$ consumed approximately 1.64 milliseconds to decompress the occurrence data for all the 50 submitted queries, a value which is translated to an average of 0.033 milliseconds per query and 83.8 nanoseconds per posting. On the other hand OptP4D and P4D were roughly 5.2 and 5.1 times slower, respectively. Similar speed differences are also observed in the case of $K = 1000$; TZP decodes the positional and zone data in times which are 5.9 and 5.6 times faster than the respective ones of OptP4D and P4D, respectively.

2.8 Conclusions

In this chapter we studied several efficiency issues concerning the compression and organization of an inverted index. We have stated that although the block-based compression methods such as PForDelta and VSEncoding achieve excellent performance when applied at document IDs and frequency values, they are not equally effective when they are utilized in positional data. This is happens because the access to the positional data during

query processing is particularly expensive, hence, we avoid touching it for all candidate documents; we do so only for the most promising ones. Since the aforementioned methods operate on entire bundles of integers, they decompress redundant information during query processing. In addition, to begin the decompression step, the current state-of-the-art methods require that we first locate the desired data, thus appending an additional look-up cost.

We addressed both of these problems by introducing PFBC, a method which employs a fixed number of bits and a limited number of pointers to encode the positional values of an inverted list. We have demonstrated theoretically and experimentally that PFBC outperforms both PForDelta and VSEncoding in terms of decompression speeds, whereas, it eliminates the additional look-up cost, and decodes *only* the data actually required without touching redundant information.

In the sequel, we studied the possibility of integrating additional information within an inverted list. We replaced the plain positional value by the *occurrence*, a quantity which embodies both the position and the field (or zone) of the document where a word occurs. We showed how we can enrich the inverted list postings with the occurrences and we introduced TZP, an expansion of PFBC which enables direct compression of the positional and zoneID value.

TZP retains the advantages of PFBC over the adversary compression algorithms, whereas it manages to keep the overall inverted index size at reasonable levels. We demonstrated that such an enriched inverted index which has its occurrence data compressed by TZP is only 9% times larger than a plain positional index. Compared to other experimental enriched indexes which had their occurrence data encoded by PForDelta and VSEncoding, our proposed index setup achieves 5 times faster decompression of the occurrence data and also, it exhibits further benefits by eliminating the look-up costs.

CHAPTER III

Influence Flow in Social Networks

3.1 Introduction

During the last years, we have witnessed a massive transition in the applications and services hosted on the Web. The obsolete static Web sites have been replaced by numerous novel, interactive services whose common feature is their dynamic content. The social and participatory characteristics that were included in these services, led to the generation of virtual communities, where users share their ideas, knowledge, experience, opinions and even media content. Examples include blogs, forums, wikis, media sharing, bookmarks sharing and many others, which are collectively known as the Web 2.0.

Blogs are locations on the Web where individuals (the bloggers) express opinions or experiences about a subject. Such entries are called blog posts and may contain text, images, embedded videos or sounds and hyperlinks to other blog posts and Web pages. On the other hand, the readers are provided with the ability to submit their own comments in order to express their agreement or disagreement to the ideas or opinions contained in the blog post. The comments are usually placed below the post, displayed in reverse chronological order. The virtual universe that contains all blogs is known as the *Blogosphere* and accommodates two types of blogs [4]: a) *individual blogs*, maintained and updated by one blogger (the blog owner), and b) *community blogs*, or multi-authored blogs, where several bloggers may start discussions about a product or event. Since in the former type of blogs, only the owner can start a new line of posts, the present article focuses only on community blogs.

In a physical community, people use to consult others about a variety of issues such as which restaurant to choose, which medication to buy, which place to visit or which movie to watch. Similarly, the Blogosphere is a virtual world where bloggers buy, travel and make decisions after they listen to the opinions, knowledge, suggestions and experience of other

bloggers. Hence, they are *influenced* by others in their decision making and these others are defined in [77] as *the influentials*.

The identification of the influentials is of significant importance, because they are usually connected in large virtual communities and thus they can play a special role in many ways. For instance, commercial companies can turn their interest in gaining the respect of the influentials to become their "unofficial spokesmen", instead of spending huge amounts of money and time to advertise their products to thousands of other potential customers. It can also lead to the development of innovative business opportunities (related to commercial transactions and traveling) can assist in finding significant blog posts [48, 67], and can even be used to influence other peoples' voting behavior.

The issue of identifying influential bloggers is very recent and despite it seems similar to problems like the identification of influential blog sites [62] and the identification of authoritative Web pages [79], the techniques proposed for these problems can not be applied to the identification of influential bloggers. The problem of identifying the influential bloggers has been introduced in [5], and the literature lacks other sophisticated solutions. That initial model, mentioned here as *the influence flow method*, explicitly discriminated the influential from the active (i.e., productive) bloggers, and considered features specific to the Blogosphere, like the blog post's size, the number of comments, and the incoming and outgoing links. Nevertheless, this model fails to incorporate temporal aspects which are crucial to the Blogosphere and does not take into account the productivity as another factor which affects the influence.

Motivated by these observations, this chapter proposes two new metrics called *MEIBI* and *MEIBIX*, for identifying influential bloggers in community blogs. The proposed approaches take into consideration both the temporal and productivity aspects of the blogging behavior, along with the inter-linkage among the blogs posts. Furthermore, we expand the problem of categorizing the bloggers according to their recent activity and influence by introducing four representative classes. These classes are formed with respect to the current blogging activity of a writer and the current influence flowing from his posts. To categorize a blogger into a class, we propose time-aware metrics by considering both the temporal and productivity aspects of the blogging behavior, along with the inter-linkage among the blogs posts.

All three proposed methods are evaluated against the aforementioned initial model (which is the only competitor so far) using data from real-world blog sites.

3.2 Related work

The recent explosion of the Blogosphere has attracted a surge of research on issues related to Blogosphere modeling, mining, trust/reputation, spam blog recognition, and many others [4]; these issues though are not directly relevant to the present work. The specific problem of identifying the influential bloggers in a blog site draws analogies from the problems of identifying influential blog sites and identifying authoritative Web pages (Web ranking). The identification of influential blog sites [62] and the related study of the spread of influence among blog sites [63, 64, 72, 83] are orthogonal to the problem considered here, since we are interested in identifying influential bloggers in a single blog site, which might be or might not be an influential blog site. Similarly, the eigenvector-based methods for identifying authoritative Web pages [79], like PageRank and HITS, "*are not useful to our problem, since blog sites in Blogosphere are very sparsely linked*" [78]. Finally, it is obvious that the works which propose methodologies for discovering and analyzing blog communities [86, 157] can not be exploited/tailored to our problem.

The only work directly relevant to our problem is that reported in [5], which introduced the problem. To solve it, the authors proposed an intuitive model for evaluating the blog posts. This model is based on four parameters: Recognition (proportional to the incoming links), Activity Generation (proportional to the number of comments), Novelty (inversely proportional to the outgoing links) and Eloquence (inversely proportional to the post's length). These parameters are used to generate an influence graph in which the influence flows among the nodes. Each node of this graph represents a single blog post characterized by the four aforementioned properties. An influence score is calculated for each post; the post with the maximum influence score is used as the blogger's representative post. The influence score $I(p)$ of a blog post p that is being referenced by ι posts and cites θ external posts, is determined by the following equation:

$$S_{\iota,d} = w(L_d)\Big(w_c C_d + w_{in} \sum_{m=1}^{|D_{c,d}|} S_{\iota,d_m} - w_{out} \sum_{n=1}^{|D_{r,d}|} S_{\iota,d_n}\Big) \tag{3.1}$$

where $w(L_d)$ is a weight function depending on the length L_d of a post d and w_c denotes a weight that can be used to regulate the contribution of the number of comments (C_d). Finally, w_{in} and w_{out} are the weights that can be used to adjust the contribution of incoming and outgoing influence respectively. The calculation of this influence score is recursive (positive reinforcement from incoming links and negative reinforcement from outgoing links), similar to the PageRank definition. This score is the $\iota Index$ metric, which can be later used to identify the most influential bloggers. Isolating a single post to identify

whether a blogger is influential or not, is an oversimplistic approach, and so it would be if they have used gross metrics, like average, median and so on. A blogger may have published only a handful of influential posts and numerous others of low quality, whereas other bloggers may have published several tens of influential blogs only, whose score though is lower than the score of the most influential blog of the former blogger. Therefore, the productivity of bloggers is a significant issue that has been overlooked by this preliminary model.

Another drawback of this preliminary model is that its output depends highly on user defined weights. The value change of the above properties can lead to different rankings. Hence, its outcome is not objective, as tuning the appropriate weights the model identifies influential bloggers with different characteristics. In other words this model cannot provide a satisfactory answer to the question *"who is the most influential blogger?"*; but it can give answers to questions of type *"who is the most influential blogger according to the number of comments that his/her posts received?"*.

But, most importantly, this model (and also the naive models which are based on the k most active bloggers), ignore one of the most important factors in Blogosphere: Time. As already known [4], the Blogosphere expands at very high rates, as new bloggers enter the communities and some others leave it. Hence, an effective model that identifies influential bloggers should take into consideration the date that a post was submitted and the dates that the referencing posts were published, in order to be able to identify the *now-influential bloggers*. Additionally, with such requirements it is mandatory to have fast methods (even on-line methods) for the discovery of the influentials, which precludes the use of demanding and unstable recursive definitions, like that used by the influence-flow method proposed in [5].

3.3 Identifying the influential bloggers

In this section we introduce our first two contributions for identifying the influential bloggers, namely MEIBI and MEIBIX. Initially we discuss the main factors which determine the essence of influence in Blogosphere, and in the sequel we present the aforementioned approaches. In the last part of this section we experimentally demonstrate the usefulness of of these methods by comparing their performance against the existing influence flow model.

3.3.1 Factors measuring a blogger's influence

Beyond any doubt, the number of incoming links to a blog post is strong evidence of its influence. Similarly, the number of comments made to a post is another strong indication that this blog post has received significant attention by the community. The case of outlinks is more subtle. In Web ranking algorithms like PageRank and HITS, the links are used only as a recognition of (or to convey) authority. The influence-flow method of [5] assigns two semantics to a link: it is the means of conveying authority, and it is also a means of reducing the novelty. This mechanism results in two significant problems: a) it misinterprets the intention of the link creators, and b) it causes stability and convergence problems to the algorithm for the influence score calculation. It is characteristic that the authors admit ([5, page 215]) that the presence of outlinks in novel posts is quite common and it is used "to support the post's explanations". Therefore, we argue that the outlinks are not relevant to the post's novelty, and all links should have a single semantic, that of implying endorsement (influence).

It is generally acceptable that the longer documents are more possibly of higher informational value than the shorter ones. This intuition is also present in some of the most successful Web ranking functions such as BM25 , where the length of a document is a factor that determines its score during query processing. Regarding blog communities, although the length of a post is not a safe indication of its influence, we accept here that longer posts are likely to cause stronger reactions from other bloggers or readers, than the shorter ones.

The temporal dimension is of crucial importance for identifying the influentials in rapidly changing environment such as Blogosphere. Time is related to the age of a blog post and also, to the age of the incoming links to that post. Moreover, the age of the comments made to a post is also of significant importance. In the former case, the time involves the age of the post (e.g., in days since the current day) and in the latter case, the time involves the age (e.g., in days since the current day) of the incoming links and of the comments.

An influential blogger is recognized as such if s/he has written influential posts recently, or if the posts have had an impact recently. In the Blogosphere, we identify two forms of impact:

- *Impact inside community:* Denotes the influence that a blogger has on the regular members/readers of the community. It is mainly visible by the comments made to a post.

- *Impact outside community:* Denotes the influence that a blogger has on other bloggers outside the community. The incoming links that a post receives is a strong

indication of this type of impact. There are also some other indications revealing the impact that a post has outside the community (such as the number of Facebook[1] or Twitter[2] shares), but these characteristics are supported only by a small number of blog communities and are not considered in this work.

There is another observation evident by the analysis presented in [5]: many of the influential bloggers were also active (i.e., productive) bloggers (see Table 1 and Tables 3–5 of [5]). Although, productivity and influence do not coincide, there is quite a strong correlation between them. Therefore, productivity should somehow be taken into account when seeking for influential bloggers.

3.3.2 The MEIBI and MEIBIX metrics

Let us begin our analysis by introducing the universe $\mathcal{B}$ which represents Blogosphere. Within Blogosphere we define two sets: The first one is A and contains all bloggers, whereas the second one is D and is composed of all the blog entries. From these main sets we identify two important subsets, $D_a \subset D$ which accommodates the blog posts authored by a blogger a, and $D_b \subset D$ which includes the set of posts published by a blog site b.

For each blog entry $d \in D$ we formulate a set of properties which includes: i) the number of comments C_d submitted by the post readers, ii) the number of other posts $D_{c,d} \subset D$ and $D_{c,d} \subset D$ referring to and referenced by d, and iii) t_d which represents the date and time when the post was published expressed as a time stamp[3]. The elapsed time since the creation of a post d (i.e. the age of a post) is symbolized as Δt_d and is expressed in seconds. In case we need to convert Δt_d in another metric unit, we use the quantity $(t - t_d + \theta)/\theta$ where t is the current time stamp and θ is a constant tuned to express the time difference in the desired unit. For example, in case it is required to express Δt_d in days, we set $\theta = 86400$.

As already mentioned, the map in Blogosphere changes rapidly, in a manner that a blogger who would currently considered as an influential, is not guaranteed to remain influential in the future. New bloggers enter the community and thousands of posts are submitted every day. In section 3.3.3 it is demonstrated that a blogger may submit up to hundreds (or even thousands) of posts yearly. In this dynamic environment, the date that a blogger's post was submitted is crucial, since a blog post becomes "old" very quickly. An issue being

[1] http://www.facebook.com

[2] http://www.twitter.com

[3] The time stamp is a 32-bit integer which represents the number of the elapsed seconds since January 1st, 1970

Symbol	Meaning
A	the set which contains all bloggers
D	the set which contains all blog posts
a	a blogger $a \in A$
d	a blog post $d \in D$
D_a	the set which contains all the blog posts of the blogger a
C_d	the set of comments to the post d
$D_{c,d}$	the set of posts referring (have a link) to the post d
$D_{r,d}$	the set of posts referenced by the post d
L_d	the length (in words) of the post d
t_d	the time stamp of d
S_d	a score value of the post d
h_a	a metric for the evaluation of the blogger a

Table 3.1: Summary of the used symbols

discussed in a blog post at the present time and is now of major importance, may be totally outdated after two months. To account for this, we assign a score S_d to the a post of a blogger as follows:

$$S_{M,d} = \gamma(C_d + 1)|D_{c,d}|\Big(\frac{\theta}{t - t_d + \theta}\Big)^{\delta} \tag{3.2}$$

The parameter γ is not absolutely necessary, but it is used to grant to the quantities $S_{M,d}$ a value large enough to be meaningful. Similarly, the parameter δ does not affect the relative score values in a crucial way, but it is used to allow for fast decaying of older posts. Both parameters do not need complicated tuning, since they are not absolutely necessary; in our experiments, γ and δ are assigned values equal to 4 and 1, respectively. Since a post may receive no comments at all, we add one to the factor that counts the number of comments to prevent null scores.

Using the definition of scores $S_{M,d}$, we introduce a new metric ***MEIBI***[4] for identifying influential bloggers. The definition of MEIBI follows:

Definition 1. A blogger a has MEIBI index equal to $h_{M,a}$, if $h_{M,a}$ of his/her D_a posts get a score $S_{M,d} \geq h_{M,a}$ each, and the rest $D_a - h_{M,a}$ posts get a score of $S_{M,d} < h_{M,a}$.

This definition awards both influence and productivity of a blogger. Moreover, a blogger will be influential if s/he has posted several influential posts recently.

But an old post may still be influential. How could we deduce this? Only if we examine the age of the incoming links to this post. If a post is not cited anymore, it is an indication that it negotiates outdated topics or proposes outdated solutions. On the other, if an old post continues to be linked to presently, then this is an indication that it contains influential

[4]Metric for Evaluating and Identifying a Blogger's Influence.

material. Based on the ideas developed for the MEIBI metric, we work in an analogous fashion. Instead of assigning to a blogger's old posts smaller scores depending on their age, we can assign to each incoming link of a blogger's post a smaller weight depending on the link's age. This idea is quantified into the following equation:

$$S_{MX,d} = \gamma(C_d + 1) \sum_{\forall r \in D_{c,d}} \Big(\frac{\theta}{t - t_r + \theta}\Big)^{\delta} \tag{3.3}$$

Based on equation 3.3 the definition of the ***MEIBIX (MEIBI eXtended)*** metric is formulated as follows:

Definition 2. A blogger a has MEIBIX index equal to $h_{MX,a}$, if $h_{MX,a}$ of his/her D_a posts get a score $S_{MX,d} \geq h_{MX,a}$ each, and the rest $D_a - h_{MX,a}$ posts get a score of $S_{MX,d} < h_{MX,a}$.

The introduction of the MEIBI and MEIBIX generates a straightforward policy for evaluating the influence of both blog posts and bloggers. No user-defined weights need to be set before these metrics provide results, whereas the most sound features of Blogosphere are considered. Moreover, the calculation of the metrics can be performed in an online fashion, since they do not involve complex computation and they do not present stability problem like those encountered when using eigenvector-based influence scores. Note that the developed metrics are similar in spirit with the h-index and its variations (see [146]) that recently became popular in the scientometrics literature, but the challenges in Blogosphere are completely different: there are comments associated with each blog post, the time granularity is finer, the author of a post is a single person, the resulting graph might contain cycles, and many more.

There is also the possibility of taking into account the time that each comment was written, but such an extension does not contribute significantly to the strength of the model, since the time-varying interest to the post is captured by the time-weighting scheme to the incoming links, and moreover, it introduces the problem of having to handle two time scales, i.e., days for the links and the posts themselves, and hours or minutes for the comments. In the sequel, we will evaluate the effectiveness of the proposed metrics to a real-world dataset, comparing it with its only competitor [5].

3.3.3 Experimental evaluation

The evaluation of the methods proposed here, but in general, of a lot others developed in the context of information retrieval, is tricky, because there is no ground truth to compare against; things are more challenging in this case, since there is only alternative [5] to contrast with. Nevertheless, we firmly believe that our evaluation is useful and solid as long as

Year	Posts	Posts with inlinks	Inlinks
2008	3676	3653	53297
2007	4497	662	259
2006	4354	186	18
2005	4307	77	1
2004	997	8	0
Total	17831	4586	53575

Table 3.2: Time distribution of posts and inlinks.

the proposed methods reveal some latent facts that are not captured by the competitor and by some straightforward methods, which result in different rankings for the final influential bloggers. In the sequel of this section, we first describe the real data we collected for the experiments, and then present the actual experiments and the obtained results.

3.3.3.1 Data characteristics

Millions of blog sites exist. The Technorati[5] blog search engine claims to have indexed more than 115 million blogs. Since it is impossible to crawl the entire Blogosphere to obtain a complete dataset, it is essential to detect an active blog community that provides blogger identification, date and time of posting, number of comments and outlinks. The Unofficial Apple Weblog[6] (TUAW) is a community that meets all these requirements; the same source of data was used also in [5]. Although we use data from only one blog, the proposed methods can be appplied to every blog community having characteristics similar to these of TUAW. We crawled[7] TUAW and collected approximately 160 thousand pages, from which we extracted 17831 blog posts authored by 51 unique bloggers. This accounts for approximately 350 posts per blogger on average. Moreover, the posts received totally 269449 comments (15 comments per post on average); only 1761 posts (ratio 10%) were left uncommented. To obtain the incoming links to each blog post, we used the Technorati API[8]. Apart from the number of the incoming links, we also retrieved the date that the referring post was submitted and its author's name. This information is necessary for the calculation of the MEIBI and MEIBIX metrics. From the total 17831 blog posts, only 4586 of them had incoming links. Table 3.2 depicts the time distribution of both the blog posts and the incoming links.

It is interesting to note, that 80% of the total posts which have received at least one in-

[5]http://technorati.com
[6]http://www.tuaw.com
[7]December 7th, 2008.
[8]http://technorati.com/developers/api/cosmos.html

Age	Inlinks	Percentage
0 days	26346	49,2%
1 day	13470	25,1%
between 1 and 7 days	6653	12,4%
between 7 and 30 days	2406	4,5%
between 30 and 60 days	928	1,7%
between 60 and 365 days	2523	4,7%
over 365 days	1249	2,3%
Total	53575	99,9%

Table 3.3: The age of the incoming links with respect to the publication date of the post they cite.

coming link (3653 posts out of the total 4586), were submitted within the year 2008. Consequently, either TUAW was not so popular before 2008 and the bloggers were unaware of the information published there, or the posts submitted before 2008 were of medium or low quality, so that only a few other bloggers referred to them. Hence, time-aware influence metrics which measure time difference in days, are indeed necessary to differentiate between influential bloggers.

We investigate also the temporal distribution of the incoming links for a blog post measuring the intermediate time between the date a post was submitted and the date it received each of the incoming links. The results are depicted in Table 3.3. Almost half of the total inlinks were received (published) the same day that the post was submitted. Only a percentage of 2.3% of all inlinks are dated one or more years after the publication of the post. These results prove the necessity of time-aware metrics for the identification of the influentials; since the posts are influential for a few days, *it is not particularly useful to identify influentials for the whole lifetime of the blog site, but it is more substantial to identify the now-influential bloggers of the blog site*.

3.3.3.2 Identifying the influential bloggers

In this subsection we apply the proposed methods on the acquired dataset. Apart from the proposed methods, we also examine a naive method which ranks the bloggers by using only their activity, i.e. number of published posts – the activity index, one ranking method which is a straightforward adaptation of a method coming from the bibliometric literature – the h-index [146] (we call these two methods as the plain methods), and a more sophisticated method, proposed in [5].

We divide the experimentation into three parts: in the first part, we compare the influential bloggers indicated by the proposed methods, to the bloggers found by the plain

methods. We use the entire dataset as a baseline experiment, examine whether temporal considerations are worthy examining; in the second part, we compare the influential bloggers indicated by the proposed methods, with those found by the influence-flow method using the posts published in November 2008, to prove that even for small time intervals the rankings are different; finally, we examine the temporal evolution of the influential bloggers identified by the proposed methods during the year 2008, to examine whether the most influential bloggers lose their lead in influence and strengthen even more the necessity for temporal considerations.

3.3.3.3 The new methods vs. the plain ones

Table 3.4 includes the ten most influential bloggers based solely on their activity (i.e., productivity) measured by the number of posts they have published in TUAW. We also provide the dates that the first post (fourth column) and the last post (fifth column) of each blogger was published. Although *S. McNulty* is ranked first, he has not submitted any posts during the last 4.5 months. A similar observation of inactivity holds also for other top-10 influential bloggers, like *D. Chartier* who is inactive about 1.4 year and *C.K. Sample, III*, who has no post in the last 2.7 year. Recall, that both *S. McNulty* and *D. Chartier*, were ranked among the top-5 influential bloggers with the information-flow method ([5, Table 1]).

Table 3.5 presents a ranking of the ten most influential bloggers when the h-index [146] metric is used; recall that this metric examines the number of posts of each blogger and the number of incoming links to each posts, awarding both productivity and influence. The third column of Table 3.14 displays the value of the h-index metric for each blogger and the next two columns show the total number of posts he/she has submitted in TUAW and how many of them have been cited by other posts respectively. Finally, the last column illustrates the total number of incoming links that all the posts of a blogger have received.

Comparing Table 3.5 to Table 3.4, some significant differences derive. These differences justify that productivity and influence do not coincide. The most active blogger, *S. McNulty* is ranked 8^{th} when the ranking is done in decreasing h-index order. According to the h-index metric, the most influential blogger is *E. Sadun* who has 31 articles that has at least 31 incoming links each. *E. Sadun* is the fourth most active blogger in TUAW, though she has posted nothing in the last 2.5 months. Although she has been inactive recently, she is still the most influential according to the h-index metric. This proves that the h-index can indicate the most influential blogger, but cannot identify bloggers who are *both* influential and active.

In the sequel, we apply the two proposed metrics MEIBI and MEIBIX in our dataset.

	Bloggers	D_a	**First**	**Last**
1	S. McNulty	3037	06/01/2005	31/07/2008
2	D. Caolo	2242	07/06/2005	04/12/2008
3	D. Chartier	1835	26/08/2005	30/08/2007
4	E. Sadun	1560	09/11/2006	26/09/2008
5	C.K. Sample III	1057	01/03/2005	05/06/2006
6	M. Lu	1043	13/12/2006	04/12/2008
7	L. Duncan	954	19/09/2004	23/01/2007
8	C. Bohon	793	24/02/2004	04/12/2008
9	M. Rose	793	29/11/2006	05/12/2008
10	M. Schramm	648	07/06/2007	04/12/2008
11	Barb Dybwad	529	05/11/2004	05/03/2005
12	Sean Bonner	449	30/01/2004	25/08/2004
13	Robert Palmer	354	06/05/2008	04/12/2008
14	Victor Agreda, Jr.	314	01/02/2005	05/12/2008
15	Steven Sande	304	06/05/2008	04/12/2008
16	Damien Barrett	252	22/10/2005	01/08/2006
17	Brett Terpstra	226	28/12/2007	24/11/2008
18	Jan Kabili	186	30/08/2005	02/09/2006
19	Fabienne Serriere	148	22/10/2005	29/04/2006
20	Dan Lurie	143	10/06/2006	10/07/2007

Table 3.4: Bloggers ranking based on the number of posts submitted (active bloggers).

The ranking of the bloggers according to the MEIBI metric is displayed in Table 3.6.

The data displayed in Table 3.6 indicate that the blogger whose posts were the most influential recently, is *C. Bohon*. This is partially explained by the fact that 676 out of the total 793 posts, have received 9439 references; it is the highest number of incoming links among the other bloggers. Furthermore, all posts have been commented 14745 times.

On the other hand, *E. Sadun*, the most influential blogger according to the h-index metric, falls in the fourth position; considering the fact that she has remained relatively inactive in the past 2.5 months, this is a satisfactory result. *R. Palmer* and *S. Sande* occupy the second and third position respectively. All top-three bloggers have submitted posts within December 2008. This is an indication that the MEIBI index not only identifies the most influential bloggers, but also the most active. It is a metric that suits very well to our case, as Blogosphere changes rapidly and our metric manages to keep track of these changes by handling the ages of the posts and the comments that they receive.

Table 3.7 presents the most influential bloggers according to the MEIBIX index. One may detect several similarities between Table 3.6 and Table 3.7. The most active blogger of TUAW, *S. McNulty*, is not among the top-10 influential bloggers when the ranking is performed according to either MEIBI or MEIBIX. Consequently, although *S. McNulty* is

	Bloggers	$h_{h,a}$	$\|D_a\|$	**Cited**	**Inlinks**
1	E. Sadun	31	1560	489	5759
2	C. Bohon	29	793	676	9439
3	M. Schramm	25	648	339	4322
4	R. Palmer	25	354	354	4809
5	M. Rose	24	793	364	4222
6	D. Caolo	23	2242	459	4907
7	M. Lu	23	1043	397	4282
8	S. McNulty	23	3037	334	3212
9	B. Terpstra	22	226	223	3013
10	C. Warren	22	133	112	1605
11	Steven Sande	22	304	304	3775
12	Nik Fletcher	18	128	72	1048
13	Chris Ullrich	14	54	31	475
14	Victor Agreda, Jr.	14	314	68	700
15	Joshua Ellis	13	28	28	431
16	Giles Turnbull	12	41	40	440
17	Jason Clarke	9	14	12	144
18	Lisa Hoover	9	30	16	196
19	TUAW Blogger	8	11	11	113
20	David Chartier	7	1835	113	279

Table 3.5: Bloggers ranking based on the h-index.

an active blogger, he has not submitted influential posts recently. Table 3.5 though, reveals that the blogger in question, is the 8^{th} most influential when the ranking is determined by the plain h-index metric.

Finally, we computed the correlation of the rankings produced by h-index, MEIBI and MEIBIX by using the *Spearman's rho* metric. The results (Table 3.8) indicate that MEIBI and MEIBIX produce similar rankings, but both of them diverge from the h-index ordering significantly.

3.3.3.4 The new methods vs. the influence-flow method

For the comparison of the proposed metrics against the basic competitor, i.e., influence-flow method [5], we select a subset of the real data in order to be fairer. It was obvious by the experimentation of the previous paragraphs, that the inactivity has a dramatic effect upon the final ranking. The real question concerning the usefulness of the proposed methods is whether in a small period of time, say a month, these methods would provide different rankings than those of the influence-flow method. Thus, we selected to work upon the blog posts of November 2008 only. For comparison purposes, we present in Table 3.9

	Bloggers	$h_{M,a}$	**Comments**
1	C. Bohon	49	14745
2	R. Palmer	46	9916
3	S. Sande	36	7246
4	E. Sadun	34	32432
5	M. Rose	30	13499
6	M. Schramm	30	12838
7	C. Warren	28	4857
8	D. Caolo	27	27985
9	M. Lu	25	17966
10	B. Terpstra	17	3770
11	Scott McNulty	17	41813
12	Victor Agreda, Jr.	14	23359
13	Chris Ullrich	11	1824
14	Nik Fletcher	10	3033
15	Giles Turnbull	9	1382
16	TUAW Blogger	8	2640
17	Joshua Ellis	7	627
18	Jason Clarke	6	347
19	Lisa Hoover	6	822
20	Alberto Escarlate	3	437

Table 3.6: Bloggers ranking based on the MEIBI index.

the top-10 of active (most productive) bloggers during November 2008 as this ranking is provided by the TUAW site itself.

In Table 3.9 we present the most influential bloggers for November 2008 as they are provided by the influence-flow method and the MEIBI and MEIBIX metrics. Neither MEIBI nor MEIBIX generate rankings that agree with the TUAW ranking of bloggers. TUAW concerns *R. Palmer* as more influential than *S. Sande*. On the other hand, MEIBI concerns *R. Palmer* and *S. Sande* to be equally influential. The former has authored more posts which received more comments, whereas the latter's posts although fewer, have been referenced more times by other posts. The ranking produced by MEIBIX positions *S. Sande* into the second place, higher than *R. Palmer*. We could state that MEIBIX is more sensitive to the number of incoming references than MEIBI.

Comparing the rankings produced by the proposed methods with the ranking according to the influence-flow model, we can state that this model assigns to *C. Bohon* the first position of the list. The model concerns *R. Palmer* as the second most influential blogger for the period of November of 2008 and agrees with TUAW. Despite *S. Sande* has published more articles that received more incoming links, *M. Lu*'s posts have attracted more comments. Hence, we conclude that *M. Lu* is primarily influential inside the TUAW com-

	Bloggers	$h_{MX,a}$
1	C. Bohon	48
2	R. Palmer	47
3	S. Sande	37
4	E. Sadun	33
5	C. Warren	30
6	M. Rose	29
7	M. Schramm	27
8	M. Lu	26
9	D. Caolo	25
10	B. Terpstra	15
11	Scott McNulty	15
12	Victor Agreda, Jr.	13
13	Nik Fletcher	10
14	Chris Ullrich	9
15	Giles Turnbull	9
16	TUAW Blogger	8
17	Joshua Ellis	7
18	Jason Clarke	6
19	Lisa Hoover	6
20	Alberto Escarlate	3

Table 3.7: Bloggers ranking based on the MEIBIX index.

Methods	ρ
h-index – MEIBI	0.478788
h-index – MEIBIX	0.321212
MEIBI – MEIBIX	0.951515

Table 3.8: Correlation of rankings

munity, whereas *S. Sande* has published influential posts that stimulated other bloggers to refer to them.

D. Caolo has authored less posts than *S. Sande*. Although his articles attracted both less comments and inlinks, the influence-flow model assigns him a higher rank than *S. Sande*. Obviously, the model's determination of influential bloggers, by taking into consideration only the best post and discarding all others, leads to erroneous rankings.

The *Spearman's rho* metric was used to compute the correlation of the rankings of Table 3.9. The results illustrated in Table 3.10, reveal that MEIBI and MEIBIX produce rankings that diverge significantly from the one generated by the influence-flow model.

	Bloggers	D_a
1	C. Bohon	47
2	R. Palmer	42
3	S. Sande	34
4	M. Schramm	29
5	D. Caolo	20
6	M. Rose	19
7	B. Terpstra	15
8	C. Warren	8
9	M. Lu	8
10	V. Agreda	5

	Blogger
1	C. Bohon
2	R. Palmer
3	M. Lu
4	C. Warren
5	D. Caolo
6	C. Ullrich
7	S. Sande
8	M. Rose
9	V. Agreda
10	Jason Clarke

	Blogger	M
1	C. Bohon	26
2	R. Palmer	20
3	S. Sande	20
4	D. Caolo	17
5	M. Schramm	16
6	M. Rose	13
7	M. Lu	8
8	B. Terpstra	7
9	C. Warren	7
10	V. Agreda	4

	Blogger	X
1	C. Bohon	27
2	S. Sande	20
3	R. Palmer	19
4	D. Caolo	18
5	M. Schramm	16
6	M. Rose	13
7	M. Lu	8
8	B. Terpstra	7
9	C. Warren	7
10	V. Agreda	4

Table 3.9: Bloggers ranking according to: TUAW (left). Influence-flow model (center). MEIBI and MEIBIX (right).

Methods	ρ
TUAW – influence-flow model	0.284848
TUAW – MEIBI	0.948485
TUAW – MEIBIX	0.939394
influence-flow model – MEIBI	0.418182
influence-flow model – MEIBIX	0.357576
MEIBI – MEIBIX	0.987879

Table 3.10: Corellation of rankings

3.3.3.5 Temporal evolution of the rankings produced by MEIBI and MEIBIX

Finally, it is interesting to examine how the rankings generated by the proposed metrics vary over time. Figures 3.1 and 3.2 depict the top-10 influence rankings of the bloggers in the past 11 months (from January 2008 to November 2008), when MEIBI and MEIBIX are applied respectively. The columns in Figures 3.1 and 3.2 represent the progression of time, whereas the rows contain the bloggers, ordered according to the time they were recognized as influential. Therefore, the (i, j)-th cell stores the rank of the i^{th} blogger in the j^{th} time window. The dash symbol signifies that the particular blogger was not among the top-10 of that period.

MEIBI and MEIBIX produce similar rankings; MEIBIX is more affected by the number of incoming links, whereas MEIBI assigns better scores to the posts that attracted more comments.

Studying the blogger rankings fluctuation over time, composes a valuable tool for distinguishing bloggers that have been influential for a very long or very short time. The former can be considered as more influential, as compared to the latter which are proved

	Jan 2008	Feb 2008	Mar 2008	Apr 2008	May 2008	Jun 2008	Jul 2008	Aug 2008	Sep 2008	Oct 2008	Nov 2008
Erica Sadun	1	2	1	2	1	4	3	4	2	-	-
Scott McNulty	2	10	8	6	6	3	4	-	-	-	-
Cory Bohon	3	1	2	1	2	1	2	1	3	2	1
Dave Caolo	4	8	5	3	5	5	6	5	6	7	4
Mike Schramm	5	4	4	9	9	8	7	6	5	5	5
Brett Terpstra	6	5	7	7	8	-	-	7	8	9	8
Christina Warren	7	6	-	8	-	7	9	-	7	8	9
Mat Lu	8	3	6	4	3	6	8	8	9	6	7
Michael Rose	9	7	3	5	-	-	-	9	10	3	6
Nik Fletcher	10	9	9	10	-	-	-	-	-	-	-
Chris Ulrich	-	-	10	-	-	-	-	-	-	-	-
Robert Palmer	-	-	-	-	4	2	1	2	1	1	2
Steven Sande	-	-	-	-	7	9	5	3	4	4	3
Joshua Eliis	-	-	-	-	10	10	-	-	-	-	-
Gilles Turnbull	-	-	-	-	-	-	10	10	-	-	-
Victor Agreda, Jr.	-	-	-	-	-	-	-	-	-	10	10

Figure 3.1: Influential bloggers' blogging behavior over 2008, according to MEIBI.

more trustworthy. Certainly, many other categories of bloggers can be derived from the retrospection of their activity through time and many potential applications can be developed using these categories.

	Jan 2008	Feb 2008	Mar 2008	Apr 2008	May 2008	Jun 2008	Jul 2008	Aug 2008	Sep 2008	Oct 2008	Nov 2008
Erica Sadun	1	2	1	3	1	4	3	4	2	-	-
Scott McNulty	2	-	8	6	5	3	4	-	-	-	-
Cory Bohon	3	1	2	1	2	1	2	1	3	2	1
Dave Caolo	4	7	4	2	7	5	6	5	7	7	4
Brett Terpstra	5	5	7	7	6	-	-	7	9	9	7
Christina	6	6	-	8	-	6	9	-	6	8	9
Mat Lu	7	4	5	5	3	7	7	8	10	6	8
Michael Rose	8	8	3	4	-	-	-	9	8	3	6
Mike Schramm	9	3	6	9	8	8	8	6	4	5	5
Nik Fletcher	10	9	9	10	-	-	-	-	-	-	-
Chris Urlich	-	10	10	-	-	-	-	-	-	-	-
Robert Palmer	-	-	-	-	4	2	1	2	1	1	2
Steven Sande	-	-	-	-	9	9	5	3	5	4	3
Joshua Ellis	-	-	-	-	10	10	-	-	-	-	-
Giles Turnbull	-	-	-	-	-	-	10	10	-	-	-
Victor Agreda	-	-	-	-	-	-	-	-	-	10	10

Figure 3.2: Influential bloggers' blogging behavior over 2008, according to MEIBIX.

3.4 The problem of bloggers classification

We have already mentioned that detecting the influence in the Blogosphere is important, since it may be exploited by companies, organizations, and advertisers for various purposes. However, there is a set of questions that somehow blurs the term "influence". For example, is a formerly influential blogger as significant as a blogger who is currently influential? How can we classify a blogger who has stopped publishing his thoughts, but his posts are still being read, commented and referenced? Should two influential bloggers with different publishing activity be treated equally?

These issues were some of our strongest motivations and to address them, we introduce the following four classes of bloggers:

- *Class $\mathcal{A}$: Currently Active - Currently Influential:* In this class we integrate the bloggers who are currently publishing many posts (i.e. they are highly productive) and the majority of these posts have strong impact inside and outside the community (i.e. they are highly influential). Here we encounter the most significant bloggers.

- *Class $\mathcal{B}$: Not Active Currently - Currently Influential:* The bloggers who were once active but their posts still have significant impact inside and outside the community, belong to this category. These writers are also important, since although they have stopped their activity, the subjects described and analyzed in their posts some time ago are still interesting.

- *Class $\mathcal{C}$: Currently Active - Not Influential Currently:* Here we categorize the bloggers who publish many posts currently, but these posts have no or small impact to other bloggers or readers. Note that some of the bloggers of this class, could have been influential in the past, but for some reason they have lost their ability to attract others.

- *Class $\mathcal{D}$: Not Active Currently - Not Influential Currently:* This category includes all the other bloggers, even those who were once active or influential, or both.

After we have formed the bloggers classes, we set a new problem: Given a set A of bloggers in a community and the four aforementioned classes $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$, how can we categorize the bloggers of the set within one of these classes at a specific time moment t?

The novelty introduced by these classes is that now we do not simply try to identify which bloggers are (or were) influential, but we are interested in discovering the authors who are *both* influential and productive *currently*. We firmly believe that the usage of this set of categories can lead to much more informative and useful conclusions.

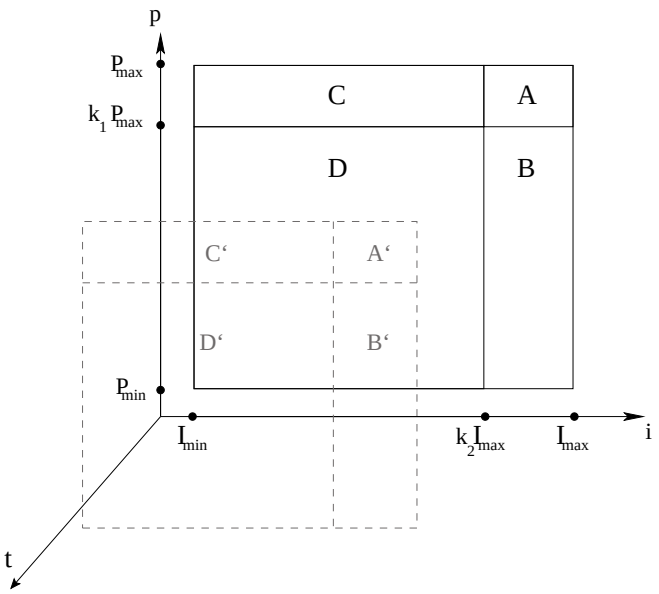


Figure 3.3: The 3-D space where the bloggers classes are located.

Apparently, the influence-flow model which considers only the best post of a blogger and does not take into account the temporal aspects related to the post, its incoming links and its comments, is not suitable to confront the problem we pose. Moreover, the usage of simple metrics such as MEIBI and MEIBIX can turn problematic when we try to express a multi-dimensional problem with a single numeric value.

3.4.1 Identifying and classifying influential bloggers

In this subsection we provide our solution to the problem we have set. Suppose that there is a way to evaluate the productivity and the influence of a blogger a in an arbitrary time instance, P_t^a and I_t^a respectively (in the next section we provide detailed description of how we can estimate them both).

To confront the problem, we adopt a geometric solution by considering a three dimensional space determined by three vertical axes: the productivity axis $\hat{p}$, the influence flowing from this productivity $\hat{i}$ and time $\hat{t}$. Figure 3.3 illustrates this three dimensional space (pit) where our four introduced classes are located in this space. On the productivity axis we distinguish two points: P_{max}^t and P_{min}^t which denote the productivity values of the most and less productive blogger respectively, during time instance t. Similarly, on the influence axis we place the points I_{max}^t and I_{min}^t which symbolize the influence values of the most and less influential blogger.

The areas $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ of the figure represent our introduced family of classes presently (i.e. $t = 0$), whereas the areas $\mathcal{A}'$, $\mathcal{B}'$, $\mathcal{C}'$ and $\mathcal{D}'$ represent the same classes in

a previous moment $t = t_1$ sometime in the past. The co-ordinates of the points that determine the regions occupied by each class at an arbitrary instance t, are given below:

$$\begin{aligned}
\mathcal{A} &\quad \Big[(P_{max}, k_2 I_{max}, t)(P_{max}, I_{max}, t)(k_1 P_{max}, I_{max}, t)(k_1 P_{max}, k_2 I_{max}, t)\Big] \\
\mathcal{B} &\quad \Big[(k_1 P_{max}, k_2 I_{max}, t)(k_1 P_{max}, I_{max}, t)(P_{min}, I_{max}, t)(P_{min}, k_2 I_{max}, t)\Big] \\
\mathcal{C} &\quad \Big[(P_{max}, I_{min}, t)(P_{max}, k_2 I_{max}, t)(k_1 P_{max}, k_2 I_{max}, t)(k_1 P_{max}, I_{min}, t)\Big] \\
\mathcal{D} &\quad \Big[(k_1 P_{max}, I_{min}, t)(k_1 P_{max}, k_2 I_{max}, t)(P_{min}, k_2 I_{max}, t)(P_{min}, I_{min}, t)\Big] \qquad (3.4)
\end{aligned}$$

where k_1, k_2 are two predefined parameters receiving values lower than 1. These parameters determine the strictness of the bloggers classification and our experimentation with several different blog communities, has proved that typical values providing satisfactory results are $k_1 = 0.7$ and $k_2 = 0.65$.

Within this space, the blogging activity of an individual at a given time instance is represented by a single point lying on the pi plane. The co-ordinates P_t^j, I_t^j of this point, namely BEP_t^j (Blogger Evaluation Point), reflect both the temporal productivity and temporal influence of a blogger a, at a specific time instance t.

3.4.2 Blogger productivity and influence

In this subsection we present a set of metrics that will assist us in categorizing the bloggers. More specifically, we examine how we can evaluate the productivity and the influence of a blogger with respect to the unique characteristics of the Blogosphere. The methods proposed here satisfy the requirements described in the previous subsections and the formulae used to estimate the productivity and the influence of a blogger are developed in a way that keeps computational requirements at low levels, without concerning the stability and convergence issues encountered in the implementation of the influence-flow model.

As already mentioned, the map in the Blogosphere changes rapidly, in a manner that a blogger who would currently be considered as productive or influential, is not guaranteed to remain productive or influential in the future. New bloggers enter the community whereas others leave it and thousands of posts are submitted every day. In this dynamic environment, the time that a blogger's post was submitted is crucial, since a blog post becomes "old" very quickly. An issue being discussed in a blog post at the present time and is now of major importance, may be totally outdated after two months. Similarly the submission date of the incoming links and the comments is also significant, since it reveals in general how

influential a blogger is presently.

To account for this, we assign a time-varying score $S_{BP,d}$ to a post d of a blogger a as follows:

$$S_{BP,d} = \gamma \frac{L_d}{\overline{L}} \Big(\frac{\theta}{t - t_d + \theta} \Big)^{\delta} \tag{3.5}$$

where the parameters γ, δ and θ are defined as previously. The definition of the scores $S_{BP,d}$ embodies the correlation between the value of a post and its age; the scores decay over time. Based on this definition, we introduce a new metric, ***BP-Index (Blogger's Productivity Index)*** for evaluating the productivity of an individual blogger. The definition of BP-Index follows:

Definition 3. A blogger a is has BP-index $h_{BP,a}$, if $h_{BP,a}$ of his D_a posts get a score $S_{BP,d} \geq h_{BP,a}$, and the rest $D_a - h_{BP,a}$ posts get a score $S_{BP,d} < h_{BP,a}$.

This definition awards the productivity of a blogger and according to it, a blogger will be currently productive if s/he has posted several long posts recently.

In the sequel, we examine how the influence of a blogger can be evaluated. As we have already mentioned the influence of a blog post has a dual nature and its is comprised of the impact it has inside the community (revealed by the comments it receives) and its impact outside the community, which is expressed by the number and the age of the incoming links. If a post is not cited or commented anymore, it is an indication that it negotiates outdated topics or proposes outdated solutions. On the other hand, if an old post continues to be linked until presently, then this is an indication that it contains influential material. Equation 3.6 reflects this dual nature by assigning to each blog post a score determined by both the comments and the inlinks.

$$S_{BI,d} = w_l \sum_{\forall r \in D_{c,d}} \Big(\frac{\theta}{t - t_r + \theta} \Big)^{\delta} + w_c \sum_{\forall c \in C_d} \Big(\frac{\theta}{t - t_c + \theta} \Big)^{\delta} \tag{3.6}$$

The parameters w_l and w_c function similarly to γ; that is, they are used to grant the score $S_{BI,d}$ a reasonably large value. However, one may argue that a comment is not as valuable as an incoming link, and these parameters can be used to regulate the desired balance. In our experiments we have used the combination $w_l = 100$ and $w_c = 10$, which means that each incoming link is considered as important as ten user comments.

Based on equation 3.6 the definition of the ***BI-Index (Blogger's Influence Index)*** metric is formulated as follows:

Definition 4. A blogger a is has BI-index $h_{BI,a}$, if $h_{BI,a}$ of his D_a posts get a score $S_{BI,d} \geq h_{BI,a}$, and the rest $D_a - h_{BI,a}$ posts get a score $S_{BI,d} < h_{BI,a}$.

Age	Engadget		Techcrunch	
	Inlinks	**Comments**	**Inlinks**	**Comments**
0 days	132,204	2,693,745	43,080	483,451
1 day	40,360	476,896	43,614	164,889
between 2 and 7 days	39,470	208,834	40,371	49,028
between 8 and 30 days	26,866	46,169	19,894	14,336
between 31 and 60 days	14,535	26,192	10,182	5,023
between 61 and 365 days	43,320	171,291	27,730	18,485
over 365 days	23,125	49,692	8,906	11,349
Total	319,880	3,672,819	193,777	746,561

Table 3.11: Incoming links and comment age with respect to the publication date of the original post.

This definition will reward the bloggers whose posts are receiving many comments and incoming links presently. Therefore, it covers the matter of identifying currently influential bloggers and in combination with BP-Index it will assist us in categorizing them into the proposed classes.

Based on these two definitions, we can now categorize the bloggers by setting a point on the pi plane of Figure 3.3. This point describes the recent blogging behavior of an individual and its co-ordinates are determined by the BP-Index and BI-Index. The region where the point is located, specifies the class the blogger belongs.

The introduction of this family of metrics generates a straightforward policy for evaluating the temporal productivity and influence of the bloggers. No user-defined weights need to be set before these metrics provide results, whereas the most sound features of the Blogosphere are considered. Moreover, the calculation of the metrics can be performed in an online fashion, since they do not involve complex computation and they do not present stability problems like those encountered when using eigenvector-based influence scores. Note that the developed metrics are similar in spirit with the h-index and its variations (see [146]) that recently became popular in the scientometrics literature, but the challenges in the Blogosphere are completely different: there are comments associated with each blog post, the time granularity is finer, the author of a post is a single person, the resulting graph might contain cycles, and much more.

3.4.3 Experimental evaluation

In this section we evaluate our metrics against the influence flow model and several scientometrics. Since millions of blog sites exist, it is impossible to crawl the entire Blogosphere to obtain a complete dataset. Hence, in order to attest our proposed methods,

	Engadget	**Techcrunch**
Bloggers	93	107
Posts	63,358	19,464
Inlinks	319,880	193,777
Comments	3,672,819	746,561
Comments per Post	57.97	38.36
Inlinks per Post	5.05	9.96
Posts per Author	681.27	181.91
Average Post Length (words)	180.52	380.76

Table 3.12: Dataset characteristics.

it is essential to detect active blog communities that provide all the required data characteristics, such as the publication date of a post, its corresponding author, and the number of comments that each post received along with their submission date and the user voting data. Engadget[9] and the Techcrunch[10] technology blog are two communities that meet all these requirements. We collected all the blog posts and the desired data from two communities, to confirm that the proposed methods can be applied to every blog community having similar characteristics.

So, we performed a full crawl of the entire Engadget Web site and from the document collection that we obtained, we extracted 63,358 blog posts published by 93 distinct bloggers. This accounts for approximately 681 posts per blogger on average). Some posts were spanning across multiple pages, due to the large number of the submitted comments; in total, we gathered about 3.67 million comments which means that each post of Engadget receives on average roughly 58 comments. Similarly, we also crawled[11] Techcrunch and mined 19,464 posts, authored by 107 writers (average 182 posts per blogger). The comments that were made to these posts were 746,561 and the average number of comments per post is approximately 38.

After these procedures, the only information that was missing from our dataset, was the total number and publication date of the incoming links. To obtain it, we employed the Google's blog search service by requesting the posts including references to the posts of our dataset. Apart from the number of the incoming links, we also retrieved the date that the referring post was submitted and its author's name. This information is necessary for the calculation of the MEIBI and MEIBIX metrics. Finally, we collected roughly 320 thousands inlinks for the posts of Engadget and about 193 thousands for the posts of Techcrunch. The precise sizes of our dataset are recorded in Table 3.12. The final row

[9]http://www.engadget.com

[10]http://www.techcrunch.com

[11]Both crawls were performed on March 28, 2010

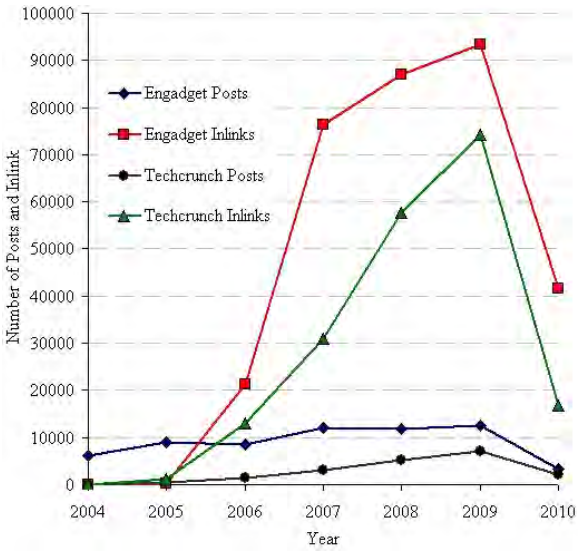


Figure 3.4: Time distribution of posts and inlinks.

indicates the average post length in words, including the title.

The writers of Engadget are more productive, since they publish on average more posts than those of Techcrunch and their posts are more commented by the readers. However, the posts of Techcrunch attract on average about double references compared to those of Engadget and are much longer.

3.4.3.1 Temporal characteristics of inlinks and comments

Before we start evaluating our proposed methods, we will examine the importance of our motivations. In other words, we shall try to figure out whether temporal aspects are worth studying and if they are really important when we try to identify influential bloggers in online communities.

Initially, we present the publication dates of the posts and the incoming links for both communities. Figure 3.4 illustrates the time distribution of the posts and their inlinks. If we exclude 2010 which is not yet completed, the most productive year was 2009.

Similarly, Figure 3.5 depicts the time distribution of all the comments made to the original posts for both communities. As before, 2009 was the most productive period, since in that year the readers submitted the most comments.

We also investigate the temporal distribution of the incoming links for a blog post, by measuring the intermediate time between the date a post was submitted and the date it received each of the incoming links. The results for both Engadget and Techcrunch are recorded in Table 3.11 (the numbers within parentheses denote the corresponding percentage with respect to the total number of inlinks, or comments). Regarding Engadget, about

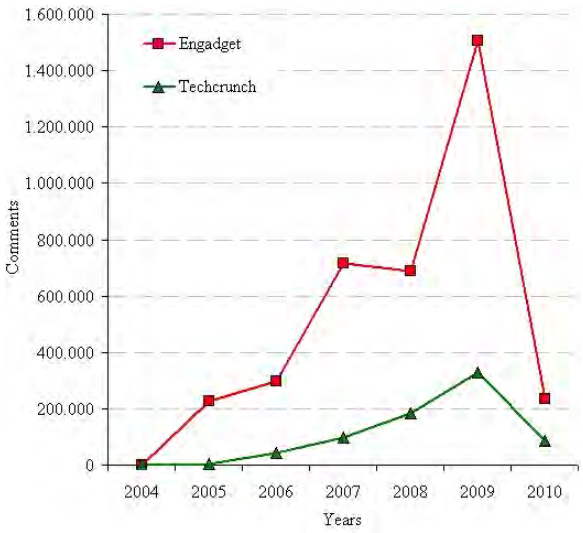


Figure 3.5: Time distribution of comments.

41% of the total inlinks were received (published) the same day that the post was submitted. Furthermore, we observe that the posts of Engadget receive the 66% of their total inlinks within the first week after their publication. Only a percentage of 7.2% of all inlinks are dated one or more years after the publication of the post. The results are similar for the Techcrunch blog: 65% of the total citations are made within the first week after the submission of the post, and only 1.4% of them has age greater than one year.

Moreover, the posts of both communities are mainly commented for only two days after their publication: approximately 86% of all the comments are submitted within this small period. On the other hand, a small percentage of roughly 8% of the comments have age greater than one month.

These results prove the necessity of time-aware metrics for the identification of the influentials; since the posts are influential for a few days, *it is not particularly useful to identify influentials for the whole lifetime of the blog site, but it is more substantial to identify the now-influential bloggers of the blog site*. Moreover, our time-aware influence metrics, which measure time difference in days, are indeed necessary to differentiate between influential bloggers.

3.4.3.2 Identifying the influential bloggers

In this subsection, we apply the proposed methods on the acquired dataset, in order to categorize the bloggers of the two blog communities to the introduced classes. Apart from the proposed metrics we also examine a naive method which ranks the bloggers by considering only their activity, i.e., number of published posts—the activity index—, one

	Bloggers	N	First	Last
1	D. Murph	11555	10/07/2006	28/03/2010
2	P. Rojas	5897	02/03/2004	24/10/2007
3	R. Block	5643	02/03/2004	30/07/2009
4	P. Miller	5000	21/09/2005	25/03/2010
5	D. Melanson	4856	15/12/2004	28/03/2010
6	T. Ricker	4798	04/05/2005	26/03/2010
7	N. Patel	3091	19/04/2007	24/03/2010
8	E. Blass	2269	05/09/2005	13/02/2008
9	M. Perton	2122	13/12/2004	16/05/2006
10	J. Topolsky	2057	18/06/2007	25/03/2010

	Bloggers	N	First	Last
1	M. Arrington	4903	11/06/2005	28/03/2010
2	E. Schonfeld	2685	24/09/2007	28/03/2010
3	J. Kincaid	1821	09/04/2008	28/03/2010
4	M. Siegler	1435	12/04/2009	28/03/2010
5	R. Wauters	1379	20/10/2008	28/03/2010
6	D. Riley	1275	02/05/2007	28/05/2008
7	L. Rao	1260	28/01/2009	28/03/2010
8	J. Biggs	698	15/08/2006	28/03/2010
9	M. Hendrickson	579	24/07/2007	21/10/2009
10	N. Gonzalez	458	05/10/2006	18/05/2009

Table 3.13: Bloggers ranking based on the number of posts submitted (active bloggers) for Engadget (left) and Techcrunch (right).

ranking method which is a straightforward adaptation of a method coming from the bibliometric literature—the h-index [146]— (we call these two methods as the plain methods), the solution proposed in [5] and the two time-sensitive metrics appeared in [12].

We divide the experimentation into three parts: in the first part, we compare the influential bloggers indicated by the proposed methods, to the bloggers found by the plain methods, MEIBI and MEIBIX. We use the entire dataset as a baseline experiment to examine whether temporal considerations are worth examining; in the second part, we compare the influential bloggers indicated by the proposed methods, with those found by the influence-flow method using the posts published in February 2010, to prove that even for small time intervals the rankings are different; finally, we examine the temporal evolution of the influential bloggers identified by the proposed methods during the year 2009, to examine whether the most influential bloggers lose their lead in influence and strengthen the necessity for temporal considerations even more.

Table 3.13 includes the ten most influential bloggers based solely on their activity (i.e., productivity) measured by the number of posts they have published in the Engadget and Techrunch blogs. We also provide the dates that the first post (fourth column) and the last post (fifth column) of each blogger was published.

Although *P. Rojas* is the second most active of Engadget, he has not submitted any posts for the last 2.5 years. A similar notification of inactivity can also be made for other top-10 influential bloggers of both communities, like *R. Block* of Engadget who is inactive for about 8 months and *D. Riley* of Techcrunch, who has not posted in the last 22 months. Apparently, this ranking is not suitable in our case, since it does not take into consideration the matter of recency in blogging activity and also, it is impossible to identify which bloggers are influential or not.

Table 3.14 presents a ranking of the ten most influential bloggers of each of the examined communities, when the h-index [146] metric is used; recall that this metric examines the number of posts of each blogger and the number of incoming links to each posts, award-

	Bloggers	$h_{h,a}$	$\|D_a\|$ (Cited)	Inlinks	Cmnts
1	R. Block	53	5643 (3197)	25251	577288
2	J. Topolsky	52	2057 (1980)	20858	264644
3	T. Ricker	42	4798 (4200)	36175	178198
4	N. Patel	42	3091 (2970)	26381	181447
5	P. Miller	40	5000 (4302)	30946	209522
6	D. Murph	34	11555 (10895)	63028	631108
7	C. Ziegler	33	1997 (1891)	14560	130673
8	R. Miller	27	1385 (1245)	9722	68795
9	V. Savov	26	944 (926)	8277	46436
10	D. Melanson	24	4856 (4196)	21878	147491

	Bloggers	$h_{h,a}$	$\|D_a\|$ (Cited)	Inlinks	Cmnts
1	M. Arrington	91	4896 (4458)	71618	248552
2	E. Schonfeld	53	2667 (2455)	28440	94843
3	M. Siegler	43	1414 (1285)	15194	71363
4	J. Kincaid	39	1805 (1651)	14383	61788
5	D. Riley	36	1275 (1189)	11631	32721
6	R. Wauters	34	1356 (1219)	9432	44990
7	M. Hendrickson	31	579 (532)	5371	18641
8	L. Rao	27	1240 (1062)	6498	27011
9	M. Kirkpatrick	26	447 (422)	3935	11387
10	G. Author	25	158 (139)	2307	12824

Table 3.14: Bloggers ranking based on h-index for Engadget (left) and Techcrunch (right).

ing both productivity and influence. The third column of Tables 3.14 displays the value of the h-index metric for each blogger of Engadget (left) and Techcrunch (right). The next column shows the total number of posts he/she has submitted and how many of them have been cited by other posts respectively. Finally, the two last columns illustrate the total number of incoming links and comments that all the posts of a blogger have attracted.

The comparison of Table 3.13 with the Table 3.14 reveals some significant differences. These differences justify that productivity and influence do not coincide. According to the h-index metric, the most influential blogger of Engadget is *R. Block* who has written 53 articles having at least 53 incoming links each. *R. Block* is the third most active blogger in Engadget, though he has posted nothing in the last 8 months. Although he has been inactive recently, he is still the most influential according to the h-index metric. Similar notifications can be made for the Techcrunch blog (i.e. note that *M. Kirkpatrick* is the ninth most influential according to h-index, but he is not among the top-10 productive).

All these indications prove that h-index can indicate the most influential blogger, but cannot identify bloggers who are *both* influential and active. In addition, it ignores the temporal aspects characterizing each blogger.

To compare the proposed metrics against the influence-flow method [5], we select a subset of the real data in order to conduct fairer experiments. It was obvious by the experimentation of the previous paragraphs, that the inactivity has a dramatic effect upon the final ranking. The real question concerning the usefulness of the proposed methods is whether in a small period of time, say a month, these methods would provide different rankings than those of the influence-flow method. Thus, we selected to work upon the blog posts of February 2010 only. For comparison purposes, we also present in Tables 3.15 and 3.16 the top-10 of active (most productive) bloggers during February 2010 as this ranking is provided by the sites themselves.

In Tables 3.15 and 3.16 we present the most influential and productive bloggers of Engadget and Techcrunch respectively, for February 2010 as they are provided by the influence-flow method and the MEIBI and MEIBIX metrics. Neither MEIBI nor MEIBIX

	Bloggers	$\|D_a\|$	Inlinks	Com
1	D. Murph	172	1179	8203
2	V. Savov	146	1194	7518
3	D. Melanson	102	643	4716
4	C. Ziegler	101	1197	5839
5	T. Stevens	73	404	2699
6	R. Miller	70	454	2710
7	T. Ricker	58	641	3820
8	N. Patel	55	550	4350
9	P. Miller	52	445	2684
10	J. Flatley	41	196	1483

	Blogger
1	J. Topolsky
2	R. Lai
3	D. Murph
4	N. Patel
5	V. Savov
6	L. June
7	T. Stevens
8	D. Melanson
9	J. Stern
10	T. Ricker

	Blogger	$h_{M,a}$
1	C. Ziegler	45
2	D. Murph	42
3	V. Savov	41
4	D. Melanson	32
5	N. Patel	32
6	T. Ricker	32
7	P. Miller	26
8	R. Miller	24
9	T. Stevens	23
10	J. Stern	23

	Blogger	$h_{MX,a}$
1	C. Ziegler	46
2	D. Murph	43
3	V. Savov	41
4	D. Melanson	33
5	N. Patel	32
6	T. Ricker	32
7	P. Miller	26
8	R. Miller	25
9	T. Stevens	23
10	J. Stern	21

Table 3.15: Bloggers ranking (February 2010) according to: Engadget (left). Influence-flow model (center). MEIBI and MEIBIX (right).

	Bloggers	$\|D_a\|$	Inlinks	Com
1	L. Rao	116	271	1603
2	M. Siegler	114	808	5380
3	R. Wauters	93	441	2949
4	E. Schonfeld	74	397	3038
5	J. Kincaid	69	431	2981
6	M. Arrington	53	513	4400
7	M. Butcher	23	59	562
8	G. Author	13	89	776
9	D. Coldewey	9	68	685
10	S. Lacy	8	35	439

	Blogger
1	M. Arrington
2	J. Kincaid
3	D. Coldewey
4	R. Wauters
5	E. Schonfeld
6	M. Siegler
7	V. Wadhwa
8	G. Author
9	J. McKenna
10	M. Burns

	Blogger	$h_{M,a}$
1	M. Siegler	32
2	E. Schonfeld	22
3	M. Arrington	21
4	J. Kincaid	19
5	R. Wauters	19
6	G. Author	9
7	L. Rao	9
8	D. Coldewey	8
9	G. Kumparak	8
10	M. Butcher	7

	Blogger	$h_{MX,a}$
1	M. Siegler	33
2	E. Schonfeld	22
3	M. Arrington	22
4	R. Wauters	20
5	J. Kincaid	19
6	G. Author	9
7	L. Rao	9
8	D. Coldewey	8
9	G. Kumparak	8
10	M. Butcher	7

Table 3.16: Bloggers ranking (February 2010) according to: Techcrunch (left). Influence-flow model (center). MEIBI and MEIBIX (right).

generate rankings that agree with the ranking provided by the blog sites themselves. For instance, Engadget concerns *D. Murph* as more influential than *C. Ziegler* in contrast to the rankings generated by MEIBI and MEIBIX. Indeed, although *C. Ziegler* has authored fewer posts than *D. Murph*, his posts received more references from other posts. A similar notification can be made for *N. Patel* and *T.Stevens*; MEIBI and MEIBIX consider the former as more influential than the latter, since his posts, although fewer, have attracted more incoming links and comments. The simple productivity-based ranking provided by Techcrunch is also ineffective, since although *L. Rao* is slightly more productive than *M. Siegler*, his posts gathered much fewer inlinks and comments.

The comparison of MEIBI and MEIBIX against the influence-flow model reveals many remarkable differences. Based on the score of the best post of a blogger (thus only only one post is considered), the ranking that the model generates appears ineffective. For instance, the model considers *J. Topolsky* as the most influential blogger of Engadget. Indeed, *J. Topolsky* has authored the best post of the entire community for February 2010, where he describes his impressions regarding Windows Phone 7 Series. The post has attracted 148 inlinks and 697 reader comments, whereas one outgoing link was included. However, *J. Topolsky* was not very productive during that period since he posted 17 times compared to the rich activity of *D. Murph* who has published 172 posts. Regarding influence, we firmly believe that a single post is not a safe indication of a blogger's influence; the posts

of other bloggers were cited and commented more frequently than those of *J. Topolsky*. Similar notifications can be made for other bloggers of Engadget (e.g. *R. Lai*, *L. June*) and Techcrunch (e.g *V. Wadhwa*, *J. McKenna* and *M. Burns*).

Now, let us examine how our proposed methods perform in this limited portion of the dataset. The left and right diagrams of Figure 3.7 illustrate the classification of the bloggers of Engadget and Techcrunch, respectively. Regarding Engadget, six bloggers are characterized as both productive and influential, thus they are categorized in class $\mathcal{A}$. The most influential blogger of Engadget according to the BI-Index is *C. Ziegler* in agreement to the rankings of MEIBI and MEIBIX. Although *D. Melanson* is not as productive as the other six, his posts gathered many incoming links and comments, hence this blogger belongs to class $\mathcal{B}$. In addition, there are five more bloggers that are highly productive, but their posts are not cited or commented sufficiently. On the other hand, Techcrunch has five bloggers who are both productive and influential (class $\mathcal{A}$) and two more whose high productivity is not accompanied by equally high influence. Once more, there is no blogger of Techcrunch who can be categorized in class $\mathcal{B}$. The reliability of our methods is confirmed by the data recorded on the left parts of Tables 3.15 and 3.16. BP-Index awards the bloggers who are really highly productive, whereas BP-Index assigns high scores to the influential ones.

Finally, to evaluate the correlation between the different rankings produced by each method, we employed the *Spearman's rho* metric. The results illustrated in Table 3.17 and Table 3.18 record the correlation between the different rankings produced by the examined models for Engadget and Techcrunch, respectively, during February 2010. The experiment reveals that MEIBI and MEIBIX produce rankings that diverge significantly from the one generated by the influence-flow model.

Methods	ρ
Engadget – influence-flow model	0.56
Engadget – MEIBI	0.73
Engadget – MEIBIX	0.73
influence-flow model – MEIBI	0.20
influence-flow model – MEIBIX	0.20
MEIBI – MEIBIX	1.00

Table 3.17: Correlation of rankings for Engadget (February 2010) according to Spearman's rho.

Methods	ρ
Techcrunch – influence-flow model	0.24
Techcrunch – MEIBI	0.68
Techcrunch – MEIBIX	0.68
influence-flow model – MEIBI	0.36
influence-flow model – MEIBIX	0.36
MEIBI – MEIBIX	1.00

Table 3.18: Correlation of rankings for Techrunch ((February 2010)) according to Spearman's rho.

3.4.3.3 Bloggers classification

In the sequel, we apply our proposed metrics to the posts of our dataset and we examine whether these metrics are suitable for categorizing bloggers. Table 3.19 contains the 15 top-ranked bloggers of Engadget and Techcrunch along with the corresponding values of BP-Index, BI-Index, MEIBI and MEIBIX. The second column of each table denotes the class each blogger belongs according to the values of BP-Index and BI-Index. Moreover, the left and right diagrams of Figure 3.6 illustrate the classification of the bloggers of Engadget and Techcrunch respectively.

The first conclusion that derives from both diagrams, is that Class $\mathcal{D}$ is the most multitudinous category. In other words, the vast majority of the bloggers are neither productive nor influential compared to the top-ranked ones. On the other hand, only a limited number (3 out of 93 for Engadget and 4 out of 107 for Techcrunch) are categorized in Class $\mathcal{A}$ and only these are characterized as both productive and influential recently. Our initial claim that productivity does not coincide with influence is verified by the bloggers belonging to Classes $\mathcal{B}$ and $\mathcal{C}$. Engadget has six such bloggers (three are influential but not productive and the other three are productive but not influential), whereas the Techcrunch community includes two writers having high productivity but medium recent influence.

The classification based on the BP-Index and BI-Index does not agree with the rankings based on MEIBI and MEIBIX metrics (Table 3.19) and this is something that we partially anticipated: MEIBI does not account for *recent* influence, since it does not consider the age of the incoming links of each post. On the other hand, MEIBIX does not account for recent productivity and, furthermore, both metrics ignore the age of the comments made to each post, hence they do not award the recency of influence inside the communities. *J. Topolsky* of Engadget is one of the three top-ranked influential bloggers according to MEIBI and MEIBIX (values 113 and 125 respectively), however our new methods place him in Class $\mathcal{D}$. A similar disagreement between the rankings also holds for *N. Patel*. Regarding the bloggers of the Techcrunch community, the rankings of MEIBI and MEIBIX coincide with

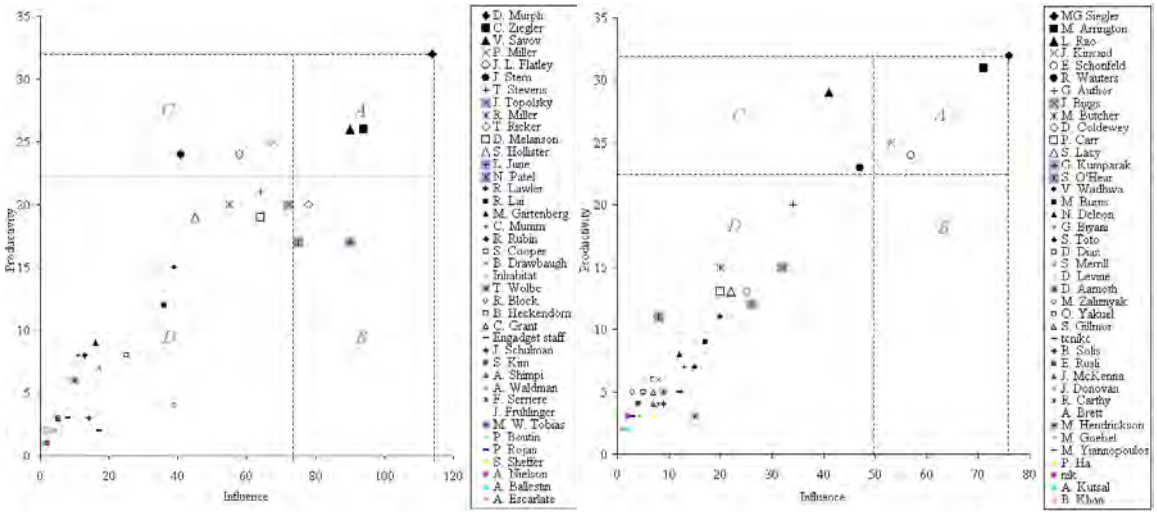


Figure 3.6: Classification of the bloggers of Engadget (left) and Techcrunch (right).

the classification of the new methods.

To discover the truth, we examined the statistics of our bloggers in the last three months (since January 1st, 2010). The results displayed in Table 3.20 indicate that the combination of BP-Index and BI-Index achieves a more accurate classification of the Engadget bloggers. Although MEIBI considers *D. Murph*, *C.Ziegler* and *J. Topolsky* equally influential, we notice that the last one has published much fewer posts which attracted fewer incoming links and comments than the posts of the other two. Therefore, we conclude that *J. Topolsky* is not productive recently and definitely not as influential as *D. Murph* and *C.Ziegler*. Regarding the rest of the bloggers, the rankings generated by MEIBI and MEIBIX somehow agree with the proposed blogger classification.

	Bloggers	**Class**	$h_{BP,a}$	$h_{BI,a}$	$h_{M,a}$	$h_{MX,a}$
1	D. Murph	$\mathcal{A}$	32	114	113	134
2	C. Ziegler	$\mathcal{A}$	26	94	113	129
3	V. Savov	$\mathcal{A}$	26	90	101	122
4	P. Miller	$\mathcal{C}$	25	68	89	105
5	J. L. Flatley	$\mathcal{C}$	24	58	55	67
6	T. Stevens	$\mathcal{D}$	21	64	60	68
7	J. Stern	$\mathcal{D}$	21	41	45	50
8	T. Ricker	$\mathcal{B}$	20	79	92	107
9	J. Topolsky	$\mathcal{D}$	20	72	113	125
10	R. Miller	$\mathcal{D}$	20	55	77	85
11	D. Melanson	$\mathcal{D}$	19	64	67	80
12	S. Hollister	$\mathcal{D}$	19	45	41	41
13	L. June	$\mathcal{B}$	17	90	79	81
14	N. Patel	$\mathcal{B}$	17	75	105	123
15	R. Lawler	$\mathcal{D}$	15	39	40	43

	Bloggers	**Class**	$h_{BP,a}$	$h_{BI,a}$	$h_{M,a}$	$h_{MX,a}$
1	MG Siegler	$\mathcal{A}$	32	76	97	114
2	M. Arrington	$\mathcal{A}$	31	71	114	137
3	L. Rao	$\mathcal{C}$	29	41	32	42
4	J. Kincaid	$\mathcal{A}$	25	53	55	64
5	E. Schonfeld	$\mathcal{A}$	24	57	72	86
6	R. Wauters	$\mathcal{C}$	23	47	51	60
7	G. Author	$\mathcal{D}$	20	34	42	45
8	J. Biggs	$\mathcal{D}$	15	32	31	36
9	M. Butcher	$\mathcal{D}$	15	20	16	18
10	D. Coldewey	$\mathcal{D}$	13	25	27	28
11	P. Carr	$\mathcal{D}$	13	20	23	23
12	S. Lacy	$\mathcal{D}$	13	22	19	19
13	G. Kumparak	$\mathcal{D}$	12	26	29	31
14	S. O'Hear	$\mathcal{D}$	11	8	4	5
15	V. Wadhwa	$\mathcal{D}$	11	20	19	19

Table 3.19: Bloggers categorization for Engadget (left) and Techcrunch (right).

Another remarkable issue is that Engadget includes three bloggers who are not productive recently, but their posts continue to attract incoming links and comments from their

readers. For instance, *T. Ricker* has authored 171 posts in the last three months; 2152 external posts included references to these posts, whereas 12940 comments were submitted by the readers to express their agreement or disagreement. On the other hand, notice that Techcrunch does not include bloggers belonging to Class $\mathcal{B}$ and, in general, the most productive bloggers are also the most influential. Two exceptions to this notification are *L. Rao* and *R. Wauters* who are categorized in Class $\mathcal{C}$ (productive, but not influential recently).

	Bloggers	**Posts**	**Inlinks**	**Comments**
1	D. Murph	464	3901	24123
2	V. Savov	406	3940	21810
3	C. Ziegler	312	3889	21117
4	D. Melanson	296	1982	13246
5	T. Stevens	287	1931	12901
6	P. Miller	217	2161	24346
7	N. Patel	204	3287	22950
8	R. Miller	193	1616	9698
9	J. L. Flatley	180	1210	7415
10	T. Ricker	171	2152	12940
11	J. Stern	120	1222	5312
12	L. June	110	937	21737
13	R. Lawler	85	820	2907
14	J. Topolsky	64	1927	16737
15	S. Hollister	53	429	2243

Table 3.20: Statistics of the Engadget bloggers between 01/01/2010 and 28/03/2010.

3.5 Blog Site Quality Scores

Although the generic problem of evaluating a Web site has been extensively studied in the past and many effective solutions exist now (such as PageRank and its variants, or HITS), the issue of the evaluation of a blog site is still an open research problem. This is due to the inappropriateness of the aforementioned eigenvector-based methods for two reasons [78]: At first, these methods treat blogs as typical interlinked Web pages and do not consider blog-specific information. And second, the blog entries graph is very sparse, hence, the performance of the traditional Web ranking methods is decreased.

In [78] the authors presented BlogRank, a method for ranking weblogs based on the link graph and on several similarity characteristics between weblogs. In this method, the blog graph is initially enhanced with implicit links, that is, virtual links which reveal the participation of the bloggers and the commentators in multiple blog services. In the sequel,

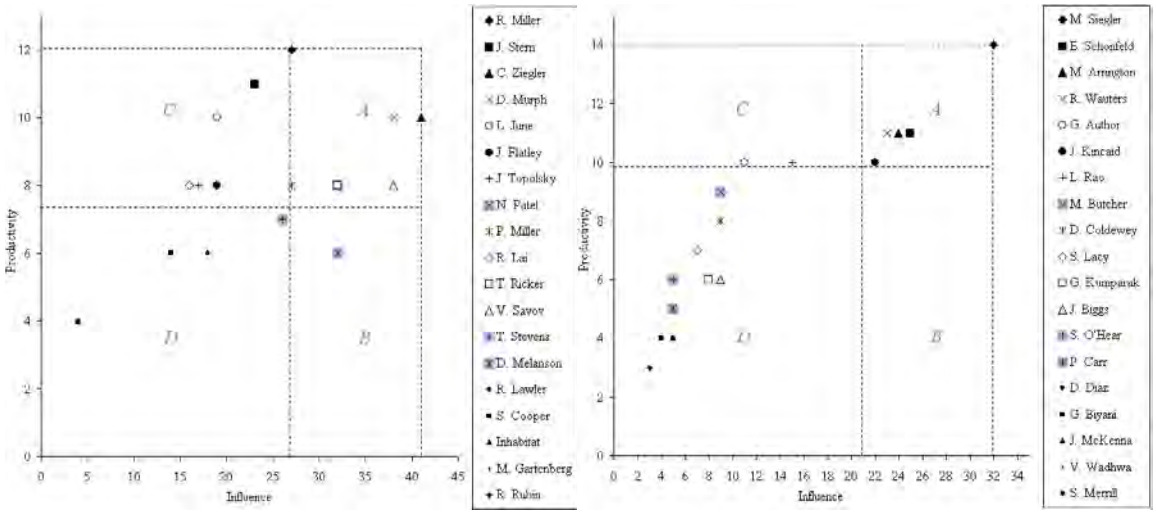


Figure 3.7: Classification of the bloggers of Engadget (left) and Techcrunch (right) for February 2010.

they define the BlogRank $S_{BR,b}$ of a blog site which has N incoming links in a PageRank recursive manner:

$$S_{BR,b} = (1 - E) + E \sum_{n=1}^{N} P(n \to b) S_{BR,n} \tag{3.7}$$

where E is a damping factor with a typical value equal to 0.85. $P(n \to b)$ represents the probability that a user who is currently in the blog site n will visit b. In the original PageRank method, we set $P(n \to b) = 1/M$ where M is the total number of outlinks of n; that is, the user may follow each outlink with equal probability. However, in BlogRank it is assumed that a user has a higher probability to visit a post which shares the same topic with n. For this reason, the blog graph is further enhanced by adding bidirectional links between the blogs which share the same thematic categories.

Now let us introduce our methods for evaluating a blog site. The first method is based on the idea that *a blog site is as important as its member bloggers are*. In other words, the impact of a blog site is determined by the influence of the blogger/s who publish posts in this specific site. For instance, a blog community which includes numerous influential bloggers is apparently of high importance. Based on this idea, we introduce *SBI-Rank (Summed Bloggers Influence)*, a metric which accumulates the influences of the member bloggers of a blog site into a single quantity $S_{SBI,b}$:

$$S_{SBI,b} = \sum_{\forall a \in A_b} h_a \tag{3.8}$$

Within Blogosphere we encounter two types of blogs [4]: a) the community blogs, or multi-authored blogs, where several bloggers may start discussions, and b) the individual blogs, maintained and updated by one blogger. In the latter case, the SBI-Rank of a blog site is identical to the influence score of its unique author. Furthermore, if the influence metric h_a is time-aware (i.e. MEIBI, MEIBIX, etc), then SBI-Rank is also time-aware since it rewards blog sites which include presently influential authors.

Our second blog evaluation method embodies the spirit of the *journal impact factor (IF)* [57], a metric which ranks scientific journals. The *IF* is based on the average number of citations received by each article of a journal within a given time period. More specifically, if the *IF* of a journal in a year Y is k, then the articles published in the years $Y-1$ and $Y-2$ received on average k citations in the year Y.

Nevertheless, this metric is impractical for evaluating blog sites for two reasons: The first one is that in contrast to the research papers, the vast majority of the blog posts become old very quickly [12]. Consequently, the posts published one or two years ago are probably never read or referenced in the present. The second reason is that *IF* is only based on citations and does not account for the user comments which are also an indication of a post's impact. We can easily overcome the first problem by replacing the years with a smaller time window (i.e. month or week). Regarding the second issue, we remind that the impact of a blog post has a dual nature, reflected by the number of incoming links and the comments. To integrate this dual nature into our analysis, we introduce the *impact units*, a linear combination between the inlinks and the comments.

$$I_d = w_r D_{c,d} + w_c C_d \tag{3.9}$$

where w_r and w_p are two constants similar to the ones used in the BI-scores which regulate the balance between the inlinks and the comments. Now, we are ready to introduce our second blog quality metric, *Blog Impact Factor (BIF)*, which is formally phrased as follows:

Definition 5: A blog site b has BIF equal to $S_{BIF,b}$ in a time window w, if the posts published in the two previous windows $w-1$ and $w-2$ received on average $S_{BIF,b}$ impact units within w.

For instance, in case a blog site b has $S_{BIF,b} = 5$ in March, then the posts published in the two previous months (January and February) received on average 5 impact units during March.

Blog	$S_{BIF,b}$
blogs.adobe.com	23
www.thehindu.com	22
www.planetmysql.org	20
www.mvblogs.org	15
sportsblogs.org	15
planetsun.org	14
planet.haskell.org	14
www.finextra.com	11
www.planetnetbeans.org	11
www.businessweek.com	11

Blog	$S_{SBI,b}$
sportsblogs.org	1546
planetsun.org	1499
www.autosport.com	1335
fashionplanet.worldofSL.com	1009
www.order-order.com	1007
www.libertaddigital.com	878
www.mvblogs.org	780
minagi.akari-house.net	725
www.golem.de	659
www.thehindubusinessline.com	656

Table 3.21: Blogs impact rankings according to: BIF (left) and SBI-Rank based on MEIBI (left).

3.5.1 Experimental Evaluation

The dataset we used is the TREC blogs08, a repository comprised of approximately 28.5 million blog posts (documents or permalinks) and 1.3 million blog feeds. The permalinks and the feeds occupy roughly 1,445 GB and 808 GB in uncompressed forms respectively.

The experiments described here is part of a wider study which is described in details in subsection 4.3.5 in Chapter IV. For further information regarding the dataset processing and the methodology followed in these experiments, the interest reader can also refer there.

In Table 3.21 we present two rankings with the most qualitative blog sites of the dataset according to the proposed BIF (left table) and SBI-Rank (right table). The former rewards the blog sites which contain recently referenced post posts and according to it, the site having the greatest impact is *http://www.newyorker.com*; the posts published on November and December of 2008 attracted on average 23 incoming links on January 2009. The second and third most influential blog sites on January 2009 were *www.thehindu.com* and *www.planetmysql.org* with 22 and 20 incoming links per post respectively. Now regarding SBI-Rank which is based on the influence of the member of bloggers of a community, we used the MEIBI index for our calculations. The corresponding ranking shows that according to this metric, the most influential blog site is *sportsblogs.org* with $S_{SBI,b} = 1,546$, followed by *planetsun.org* and *www.autosport.com* ($S_{SBI,b} = 1,499$ and $S_{SBI,b} = 1,335$ respectively).

3.6 Conclusions

In this chapter we discussed the issue of identifying the influential bloggers in blog communities. We studied an early model (influence-flow model) which assigned influence

scores to bloggers according to a number of his/her posts attributes (incoming and outgoing links, length and number of submitted comments) and we exposed its drawbacks. Considering the unstable nature of Blogosphere and the rapid changes which happen in shorts periods of time, our discussion focused on introducing temporal aspects in the identification of the influentials. Furthermore, this early model estimated the influence of a blogger by taking into account only the highest scoring blog post and ignored the rest of them; we stated that such a policy overlooks the blogger's productivity; a blogger may have authored only one highly influential post, whereas others, may have published hundreds of less influential posts.

Initially, we introduced MEIBI and MEIBIX, two time-sensitive metrics which addressed the issue of productivity by taking into consideration all the posts a blogger has authored. The former assigned scores to all the posts of a blogger by computing their age (i.e. the time elapsed since their publication), whereas the latter is based on the age the incoming links; if an old post continues to be cited in the present, then it is considered influential now.

In the sequel, we expanded these two metrics by introducing two interrelated metrics, BP-index and BI-index. This new approach attempts to graphically solve the problem of identifying the bloggers who are presently productive and influential. Therefore, the computation of these two metrics leads to a classification model which categorizes a blogger into one of four classes: $\mathcal{A}$ (both productive and influential), $\mathcal{B}$ (influential but not productive), $\mathcal{C}$ (productive but not influential), and $\mathcal{D}$ (nor productive, nor influential).

All our proposed approaches were attested against the influence-flow model and some of the most popular scientometrics by using data crawled from real-world blog communities. The experiments demonstrated the usefulness of our methods by providing more sensible and effective blogger rankings.

CHAPTER IV

Ranking

4.1 Introduction

The issue of returning results of high quality is of primary importance for search engines. It is broadly acceptable that the success of these systems is mainly determined by their capability on generating results which are relevant to an incoming query, thus satisfying the information needs of the users. The operation which is responsible for identifying relevant documents within a document storage is called ranking. Ranking is one of the main principles which distinguish search engines from databases; whereas the latter are usually required to return *all* the results which are relevant to a query, the former must return the top-k *best* results.

Nowadays, ranking in Web search engines is a particularly complex task, as it depends on multiple features; the relevancy of a document to a query, its overall objective value, its previous usefulness to the users as a result, etc. The common strategy followed by the modern Web search engines dictates that the system initially identifies the documents which are most suitable to the query by taking into consideration only a limited set of parameters. This initial step must be highly optimized since the retrieval requires traversal of huge inverted lists and furthermore, the complete evaluation of all candidate results would lead to prohibitive processing durations. Having identified the top-K most relevant results on the previous stage, the system performs a full evaluation of these K results by exploiting additional information such as positional and field information (stored in the index), click through data and relevance feedback. Finally, it returns the top-k list comprised of the most promising results.

In this chapter we examine some state-of-the-art probabilistic retrieval functions, such as BM25 (firstly introduced [115]), a variant which takes into consideration the zone of the document where a term appears (BM25F, [89]), and another variant which takes into consideration term proximity, namely BM25TP ([31]). In the sequel, we introduce

BM25TOPF, a method which assigns scores to the candidate results by taking into consideration a) both positional and zone data, and b) the ordering of the words in the query. BM25TOPF builds upon the TZP compression algorithm we presented in Chapter II. Recall that TZP replaces the position of a word in a document by its *occurrence* which encodes both positional and zone data. BM25TOPF captures the spirit of both BM25F and BM25TP and combines them within a single ranking formula.

One of our main motives which led us in the introduction of BM25TOPF was to examine whether the usage of the additional information can really lead to search results of higher quality. Another motive was that the discovery of a ranking function which combines many different parameters (i.e. frequencies, term proximity, zone weighting, document lengths, etc.) is a quite challenging task.

Furthermore, in this chapter we examine the issue of improving the retrieval effectiveness of the blog vertical search engines. Here we are not only interested in providing ranking schemes which elevate the most relevant documents, but we are primarily interested in retrieving qualitative opinionated documents, that is, blog entries which contain opinions about the query subject. Due to the aforementioned increase in the size of Blogosphere, the opinions expressed through it are now of crucial importance since they affect a large number of users and their impact is large. For instance, a positive opinion about a product can significantly increase its commercial success whereas in contrast, multiple negative statements about a politician can decrease his/her publicity and affect the success of his/her political career. Similar examples include artists, events, travel locations, service providers, and generally every judgeable aspect of life.

For these reasons, the problem of opinionated retrieval of blog entries is considered both interesting and challenging and has gained the attention of the research community. In addition, the introduction of the polarity and opinion search task by the Text Retrieval Conference (TREC) in 2006 and 2008 [103, 104, 105] has attracted even more researchers to propose solutions for this problem. In general the suggested opinion retrieval models primarily consist of three basic components: the first one implements a traditional information retrieval (IR) system which identifies topic-relevant documents (i.e. blog posts) from a document set, with respect to a given query. In the sequel, a classification or lexicon-based algorithm is employed to determine whether these posts contain opinions. Finally, a third component assigns opinion scores and combines them with the relevance scores of the IR system to produce a final ranked list of documents.

Although numerous opinionated retrieval models have been proposed in the relevant literature, none of them takes into consideration objective quality scores. The research performed towards the enhancement of the Web retrieval effectiveness indicates that the

inclusion of quality evaluation metrics (such as PageRank) in the ranking methods of Web search engines leads to improved results. In this study we convey this notion to the field of the opinionated blog post retrieval and we introduce *QUIQS (Query Independent Quality Scores)*, with the aim of improving the identification of relevant results. QUIQS consist of three families of evaluation metrics which measure the objective, query-independent quality of the i) blog posts, ii) blog sites, and iii) bloggers. Such metrics have been presented in chapter III, therefore, here we demonstrate how a vertical blog search engine can apply them to generate qualitative opinionated blog entries in response to the incoming queries.

The rest of the chapter is organized as follows: in section 4.2 we examine the problem of Web ranking; we initially present the existing state-of-the-art probabilistic functions (subsections 4.2.1, 4.2.2, and 4.2.3) and in the sequel, we describe and evaluate our proposed BM25TOPF function (subsections 4.2.4 and 4.2.5 respectively). The problem of the opinionated blog post retrieval is discussed in section 4.3, and the chapter closes with our conclusions in section 4.4.

4.2 Probabilistic Web retrieval

In this section we provide a brief overview of the current state-of-the-art ranking functions and in the sequel, we present our own ranking method that exploits the additional information stored in our proposed index setup. In Table 4.1 we summarize the symbols used by the presented scoring functions and, also, we explain their meaning.

4.2.1 The BM25 function

The BM25 weighting scheme was developed by [115] as a way of building a probabilistic model sensitive to term frequency and document length. It is not a single function, but actually a whole family of scoring functions, with slightly different components and parameters. One of the most prominent instantiations of the function is as follows.

Given a query Q containing the terms $t_1, t_2, ..., t_n$, each document d is assigned a relevance score that is given by the following formula:

$$S_{BM25}(d) = \sum_{i=1}^{n} w_{t_i} \frac{f_{d,t_i}(k_1 + 1)}{f_{d,t_i} + K}, \tag{4.1}$$

$$K = k_2 \left(1 - b_1 + \frac{b_1 l_d}{\bar{l}} \right) \tag{4.2}$$

where k_1, k_2 and b_1 are three predefined constants. Although there are some other

Symbol	Meaning
C	The entire document collection (corpus)
d	An arbitrary document in the collection
$z_{j,d}$	The j^{th} zone of d
N	Number of documents in C
Q	A user query
t_i	The i^{th} term in Q
N_{t_i}	Number of documents containing t_i
$w_{t_i} = \log(N/N_{t_i})$	Inverse Document Frequency (IDF) of t_i
$p_{t_i,Q}$	The position of t_i in query Q
$p_{t_i,d}$	The position of t_i in document d
p_{t_i,z_j}	The position of t_i in zone z_j
f_{d,t_i}	Number of occurrences of t_i within d
$f_{z_{j,d},t_i}$	Number of occurrences of t_i within $z_{j,d}$
l_d	The length of d (number of terms)
$\bar{l} = (\sum_{i=1}^{N} l_i)/N$	The average document length in C
$l_{z_{j,d}}$	The length of $z_{j,d}$ (number of terms)
$\overline{l_{z_j}} = (\sum_{j=1}^{N} l_{z_{j,d}})/N$	The average length of z_j in C

Table 4.1: Notation.

variants of the BM25 weighting scheme, the one we provide here is the most popular among them.

4.2.2 Zone weighting

Zone weighting is related to the appropriate result scoring when ranking structured or semi-structured documents. BM25F is a model proposed by [89] and takes into consideration the physical position of a term within a document (i.e. in which field of an XML document a term appears). The introduction of zones in chapter II and the usage of the enriched index structure, allows us to use BM25F in standard Web documents.

For a user query $Q = \{t_1, t_2, ..., t_n\}$, the document scoring according to BM25F is performed in two phases: Initially, we obtain the accumulated weight of a query term t_i over all fields as follows:

$$W_z(d, t_i) = \sum_{z_{j,d} \in d} \frac{S_{z_j} f_{z_{j,d},t_i}}{1 - b_2 + b_2 \frac{l_{z_{j,d}}}{\overline{l_{z_j}}}} \tag{4.3}$$

where b_2 is a predefined constant usually set equal to $b_2 = 0.75$, and S_{z_j} is a static score assigned to each document zone. The existence of the S_{z_j} score allows us to modify the importance of a term appearing in special locations within a document. For example, we could assume that a document having the query terms in the title is more relevant to this

specific query, than a document which contains these terms in its normal text.

On the second phase, the accumulated weights are being used to compute the BM25F scores according to the following formula:

$$S_{BM25F} = \sum_{i=1}^{n} w_{t_i} \frac{W_z(d, t_i)}{W_z(d, t_i) + k_3} \tag{4.4}$$

where, similarly, w_{t_i} represents the IDF of t_i, and k_3 is a predefined constant.

4.2.3 Term proximity scoring

Document retrieval functions based on the vector space model (see for instance [117, 111, 81]) and the bag-of-words representation of documents, such as Okapi BM25 have been proved to be effective in ad-hoc information retrieval tasks. One of their drawbacks is that they do not take the proximity of query terms within a document into account. However, there are many queries where the best results contain the query terms in a single phrase, or at least in close proximity as indicated by [118, 119, 95].

The BM25TP scheme is an expansion to the BM25 weighting, attempting to integrate term proximity into the original scoring function. It was firstly presented by [31] and it is very similar to the one proposed by [113]. We examine this model because in experiments with the TREC collection it was the only one exhibiting significantly better performance than the others, according to [119].

Suppose a user submits the query $Q = \{t_1, t_2, ..., t_n\}$. With every query term t_i we associate an accumulator $acc_d(t_i)$ that stores the term's proximity score within the current document d. Whenever the query processor encounters a posting belonging to a query term t_x, it computes the distance (number of postings) between this posting and the previous posting belonging to the term t_y. If $t_x \neq t_y$, then we increment the accumulators for both terms according to the following formulas:

$$acc_d(t_x) = acc(t_x) + w_{t_x} \frac{1}{(p_{t_x,d} - p_{t_y,d})^2}, \tag{4.5}$$

$$acc_d(t_y) = acc(t_y) + w_{t_y} \frac{1}{(p_{t_y,d} - p_{t_x,d})^2}, \tag{4.6}$$

If $t_x = t_y$ we leave the accumulators unchanged. By using these accumulators the BM25TP score is now defined as follows:

$$S_{BM25TP} = S_{BM25} + \sum_{i=1}^{n} min\{1, w_{t_i}\} \frac{acc_d(t_i)(k_1 + 1)}{acc_d(t_i) + K} \tag{4.7}$$

The BM25TP provides significant improvements in result quality over BM25 scoring. These gains are becoming even greater as the size of the document collection increases. This is due to the fact that for larger collections, the probability of finding non-relevant documents that contain the query terms by chance is greater than for smaller collections.

One drawback of this method is its insensitivity to the query terms ordering, since it holds that $(p_{t_x,d} - p_{t_y,d})^2 = (p_{t_y,d} - p_{t_x,d})^2$. Consequently, BM25TP assigns scores to the documents regardless of the query formulation and fails to distinguish the difference between queries such as *John is faster than Mary* and *Mary is faster than John* as mentioned in [92]. To address this issue, we initially introduce a quantity $\xi(t_x, t_y)$ which receives the following values:

$$\xi(t_x, t_y) = \begin{cases} 1, \ p_{t_x,Q} - p_{t_y,Q} > 0 \\ -1, \ p_{t_x,Q} - p_{t_y,Q} < 0 \end{cases} \tag{4.8}$$

That is, in case we examine a document containing two terms which appear in reverse order than in the query, $\xi(t_x, t_y)$ receives a constant negative value (-1), whereas a constant positive value is assigned in the opposite case. The following fraction:

$$a_d(t_x, t_y) = (p_{t_x,d} - p_{t_y,d}) \Big/ \xi(t_x, t_y) \tag{4.9}$$

is always positive in case the terms in the document are positioned in the same order as in the query, and negative in the opposite case. Now, to reward both term proximity and correct term ordering, we need to replace the square difference $(p_{t_x,d} - p_{t_y,d})^2$ in the denominators of Equations 4.5 and 4.6 by a function which is:

- always positive, regardless of $p_{t_x,d} - p_{t_y,d} > 0$ or $p_{t_x,d} - p_{t_y,d} < 0$; i.e., we do not desire to assign negative scores.
- becoming higher as $p_{t_x,d} - p_{t_y,d}$ increases and vice versa; i.e., it rewards term proximity (since the score is inversely proportional to this quantity).
- becoming higher in case of $p_{t_x,d} - p_{t_y,d} < 0$ and vice versa; i.e., it rewards correct term ordering.

A function which satisfies these conditions is the polynomial

$$\varphi_d(t_x, t_y) = \Big(a_d(t_x, t_y)\Big)^2 - a_d(t_x, t_y) + 1 \tag{4.10}$$

which is always positive for all $a_d(t_x) \geq 1$ and $a_d(t_x) \leq 0$, which in our case is always true, since $p_{t_x,d} - p_{t_y,d}$ receives values within $(-\infty, -1] \cup [1, \infty)$. Notice that there

is an infinite number of polynomials satisfying the requirements that we have set, however, here we have chosen the simplest one which requires the minimum processing. By replacing the square difference $(p_{t_x,d} - p_{t_y,d})^2$ by the polynomial $\varphi_d(t_x)$, the accumulators of Equations 4.5 and 4.6 are transformed according to the following relationships:

$$acc'_d(t_x) = acc'_d(t_x) + w_{t_x} \frac{1}{\varphi_d(t_x, t_y)}, \tag{4.11}$$

$$acc'_d(t_y) = acc'_d(t_y) + w_{t_y} \frac{1}{\varphi_d(t_x, t_y)}, \tag{4.12}$$

This new form satisfies all the requirements that he have set: In case the query terms appear close in a document, the value of $\varphi_d(t_x)$ is small, hence the accumulator value increases. Furthermore, if these terms appear in the opposite order with respect to the given query, $\varphi_d(t_x)$ increases thus reducing the overall document's score.

Based on the accumulators of Equation 4.11 we introduce a modified scoring formula, BM25TOP, which apart from term proximity, it also takes into account the query formulation and the query terms ordering. The scores of BM25TOP are provided by the following equation:

$$S_{BM25TOP} = S_{BM25} + \sum_{i=1}^{n} min\{1, w_{t_i}\} \frac{acc'_d(t_i)(k_1 + 1)}{acc'_d(t_i) + K} \tag{4.13}$$

where K is calculated as previously by using Equation 4.2. Note that the proposed enhancement for term ordering applies not only to BM25TP, but also, to all term proximity scoring schemes which utilize position-dependent accumulators, such as the effective variant proposed by [119].

4.2.4 Combining term proximity with zone weighting

Until now, we have discussed several scoring models that take different parameters into consideration. BM25TP is specially designed towards evaluating term proximity in a query, whereas our BM25TOP enhancement also takes into consideration the way a query is formulated. On the other hand, BM25F emphasizes on the physical location of a term in a document. Nevertheless, to the best of our knowledge there is no publicly known model which combines term proximity, query term ordering and zone weighting into a single scoring formula.

The main idea is that term proximity is a feature which should be further rewarded when the query terms are positioned within the same zone. For instance, when two or more terms of a given query are encountered in close proximity within the title of a document,

then the score of this document should increase.

To incorporate term proximity and zone scoring, we initially replace the document accumulators of Equation 4.11 by the *zone accumulators*, determined by the following relationship:

$$acc'_{z_j}(t_x) = \begin{cases} acc'_{z_j}(t_x) + w_{t_x} \frac{1}{\varphi_d(t_x, t_y)}, \ t_x, t_y \ \in \ z_j \\ acc'_{z_j}(t_x), \ otherwise \end{cases} \tag{4.14}$$

Therefore, instead of assigning one accumulator per query term per document, we assign one accumulator per query term per *zone* and we compute each of them according to equation 4.14.

Now let us return to Equation 4.3 which represents the accumulated weights of the query terms over all the document zones. As we have already mentioned, we desire to reward term proximity when the terms of a query appear into the same zone. For this reason, we integrate into these weights our modified accumulator value; the new weights are now evaluated by applying:

$$\mathcal{W}_z(d, t_i) = \sum_{z_{j,d} \in d} S_{z_j} \left(1 + \frac{1}{k_2} \frac{acc'_{z_j}(t_i)}{acc'_{z_j}(t_i) + k_1} \right) \frac{f_{z_{j,d}, t_i}}{1 - b_2 + b_2 \frac{l_{z_{j,d}}}{\overline{l_{z_j}}}} \tag{4.15}$$

In this equation, the left quantity rewards the occurrence of a query term in a particular document zone, whereas the right one rewards both term proximity and correct ordering. Notice that the special weight S_{z_j} applies to both terms, since term proximity is not equally important for all zones. Consequently, the occurrence of two adjacent query terms in the title of a document is more significant than a similar occurrence in its plain text. On the second phase, the accumulated weights are being used to compute the BM25TOPF scores according to the following formula:

$$S_{BM25TOPF} = \sum_{i=1}^{n} w_{t_i} \frac{\mathcal{W}_z(d, t_i)}{\mathcal{W}_z(d, t_i) + k_2} \tag{4.16}$$

where, similarly, w_{t_i} represents the IDF k_2 is a predefined constant. BM25TOPF awards documents that contain all, or some of the query terms multiple times in significant locations (i.e. title, headings etc) and moreover, the documents having the query terms close to each other. It also takes into consideration the query term ordering, the document length, the zones length and the inverse document frequencies of the query terms.

As we demonstrate in our experimental section, BM25TOPF improves retrieval effectiveness by a significant margin compared to the existing approaches.

4.2.5 Experimental Evaluation

To evaluate the performance of our proposed method in a fair and unbiased manner, it is required that we obtain a predefined set of queries. Furthermore, for each of these queries, we need to possess a list of humanly judged relevant documents. For this reason, we have employed the results of the Web Adhoc (WA) Task of TREC-2009 Web Track [34]. This task consists of 50 topics (test queries) all accompanied by a corresponding list of relevant documents from the Clueweb09-T09B data set.

For the needs of this experiment we have developed a query serving system consisting of ten query servers and a broker. Each server was assigned a different index shard, whereas the broker was responsible for merging the results generated by each server. For the evaluation, we used the 'trec_eval' standard program utilized by the TREC community in order to calculate several measures indicating the retrieval effectiveness of a system. These measures are Mean Average Precision (MAP), R-Precision and Precision@n ($P@n$) for $n = 10, 20$ and 30.

BM25TOPF is compared against BM25, BM25TP, BM25TOP and BM25F. The values of the various parameters of section 4.2 that we used in our experiments are recorded on the left part of Table 4.2. In addition, on the right part of Table 4.2 we provide the weighting scenario that we have set in order to evaluate BM25F and BM25TOPF. According to this scheme, a term occurring in a document's title is considered six times more important than one appearing within the main text, whereas the ones appearing in the headings of a document are four times stronger than the normal words.

Parameter	**Value**
k_1	1.2
k_2	2.0
k_3	2.0
b_1	0.9
b_2	0.75

Zone	S_{z_j}
Body (Normal Text)	1
Anchor Text	1
Title Text	6
Document's URL	2
Headings	4
Page Description	3
Image Description	1
Label Text	1

Table 4.2: Parameter setting for the various ranking methods (Left) and zone weighting scenario for the BM25F and BM25TOPF functions (Right).

Table 4.3 shows the performance of the five examined retrieval methods in the 50 queries of the WA task. The first point that is highlighted by the presented results is that all methods performed better than the plain BM25 model. The improvements are somehow limited when term proximity scoring is considered (BM25TP and BM25TOP), and become

significant for our zone weighting scheme (BM25F).

Retrieval Method	MAP	P@10	P@20	P@30	R-Precision
BM25	0.0599	0.2820	0.2460	0.2047	0.1127
BM25TP	0.0634	0.2820	0.2530	0.2220	0.1216
BM25TOP	0.0658	0.2940	0.2780	0.2387	0.1238
BM25F	0.0730	0.3140	0.2690	0.2407	0.1318
BM25TOPF	0.0784	0.3360	0.2820	0.2627	0.1390

Table 4.3: Performance of different retrieval methods for the 50 queries of the Adhoc Task of TREC-2009 Web Track.

BM25TOPF outperformed all of its adversary approaches and the results indicate that term proximity in combination with zone weighting indeed leads to improved retrieval effectiveness. The Mean Average Precision we achieved by using BM25TOPF was 0.0784 in comparison to the MAP value of 0.073 performed by the second best method, BM25F. This is translated into a result quality improvement of about 6.8%. Furthermore, BM25TOPF performed much better than the proximity-only approaches; the MAP values for BM25TP and BM25TOP were 0.0634 and 0.0658, respectively.

The combination of term proximity with correct term ordering in BM25TP also leads to better performance; BM25TOP produced results which were more qualitative by approximately 3.8% (in terms of MAP) than those generated by BM25TP. However, both term proximity functions were outperformed by BM25F and BM25TOPF. This indicates that zone weighting is a more important feature than term proximity when ranking documents in Web search engines.

Regarding the average Precision values at cut off points 10, 20, and 30, BM25TOPF exhibited better performance that its adversary approaches. The P@10 value was 0.336, 6.5% higher than the corresponding P@10 value achieved by BM25F. Regarding the precision value at cut-off point 20, BM25TOP was slightly outperformed by BM25TOPF, but defeated both BM25TP and BM25F.

4.3 Opinionated blog post retrieval

One of the most challenging issues in opinionated retrieval is to develop an effective method for assigning query-related opinion scores to the documents [58]. The early models did not consider the issue of the opinion relevancy to the query topic; they arbitrarily assumed that each expressed opinion refers to the subject of the query. The most recent approaches addressed this issue by applying either proximity-based strategies or data mining algorithms. However, none of the opinion scores introduced so far embody information

which indicates the generic value and impact of the retrieved documents. In this dissertation we introduce an opinion scoring approach that takes into consideration both query and opinion independent data which indicates the value of the post. Our main motivation is that the opinionated retrieval of blog posts must exploit objective and query independent criteria. Such an improvement would allow an opinion retrieval system to provide rankings which are both relevant and contain *high quality* opinions.

More precisely, the query independent model that we consider in this chapter is composed by elements which reflect the influence of the blogger who authored each post. The key idea is that an opinion expressed by an influential blogger is apparently more important than the opinion of another blogger who is of lower impact. In this work we also examine the value of the entire blog site which hosts the retrieved opinion. Following a spirit similar to PageRank, we consider that the documents appearing in reputable blog sites are more useful than other posts published in unpopular sites. All these query independent parameters which indicate the objective quality of the bloggers, blog sites, and blog posts, are collectively named *QUIQS (QUery Independent Quality Scores).*

Moreover, recent works have demonstrated that the inclusion of proximity information within the ranking component leads to enhanced opinion retrieval effectiveness [58]. Such methods consider that the distance of an opinion term to the query term is a measure of their relatedness; consequently, the opinion terms have stronger connections with the terms which are closer to their position. In this work we extend this idea by taking into consideration the physical location of the document (namely zone, or field) where the query and opinion terms occur. Therefore, our model rewards close term proximities occurring in "important" document fields, such as in the title. Our experiments conducted with the TREC Blogs08 dataset demonstrated that our method outperformed the baseline approach which employs plain IR functions by 39%, and the term proximity-model of [58] by roughly 7.2%.

4.3.1 Related work

The problem of opinion retrieval and sentiment analysis has attracted the attention of the researchers since 2006, when TREC introduced the polarity and opinion search task [103]. In contrast to the traditional document retrieval, the key problem here is to identify documents which are both relevant to a given query and contain opinion expressions. In [156] and [155] the authors adopt a machine learning approach which employs support vector machines to build opinion classifiers. Their proposed system ranks the retrieved documents by computing linearly combined opinion and relevance scores. Support vector machines for sentiment analysis were also used in [98], where the authors attempt to

combine diverse sources of potentially pertinent information.

In [99] the authors construct an opinion lexicon with respect to the given query. Their algorithm initially employs a general opinion lexicon which is refined by computing the opinion weights of its words. Moreover, [139] and [138] study the issue of building lists of subjective words (i.e. *good* against *bad*, or *excellent* against *poor*) with the aim of capturing expressed opinions within a document.

Similarly to some traditional Web ranking models, a number of relevant works takes into account the proximity of the query terms within the retrieved posts to achieve effective ranking [31]. For instance, [156] computed the probability of query terms and opinion terms co-occurrence by employing a word window. Similarly, [143] considered a word window around each query term and calculated the distance between the query terms and each word in the window. In [39] the authors computed the proximity by employing n-grams and experimented with several machine learning classification methods. The authors in [58] proposed a proximity-based opinion propagation model to calculate the opinion density at each point in a document. In addition, [107] employed supervised machine learning techniques to identify positive and negative reviews of movie films, whereas [139] used special words (such as *poor* and *nice*) and a machine-learning algorithm to achieve sentiment analysis.

Nevertheless, none of the aforementioned approaches take into consideration query-independent information during ranking. In this work we examine how the influence of a post's author and the importance of a blog site can be combined with the aforementioned strategies to enhance ranking. The first work which attempted to identify the influential bloggers in a community is [5], where the authors introduced a post scoring function based on the number of comments and the scores of the incoming and outgoing links. In the sequel, they identified the influential bloggers by the post which received the highest score. Moreover, [12] and [14] presented numerous methods which take into consideration the temporal aspects of the Blogosphere and the productivity of the bloggers.

In addition, [78] introduced BlogRank, a PageRank generalization for ranking weblogs. In this study the authors highlighted the sparseness of the blog graph and detected the inappropriateness of the traditional Web ranking models in Blogosphere. They propose a strategy for enhancing the blog graph with implicit links which are created by considering the participation of some bloggers in multiple social networks. Finally, [135] introduced B2Rank, a method for ranking blogs with respect to the behavioral features of the users.

Symbol	Meaning
A	the set which contains all bloggers
B	the set which contains all blog sites
D	the set which contains all blog posts
a	a blogger $a \in A$
b	a blog site $b \in B$
d	a blog post $d \in D$
D_a	the set which contains all the blog posts of the blogger a
D_b	the set which contains all the blog posts of the site b
C_d	the set of comments to the post d
$D_{c,d}$	the set of posts referring (have a link) to the post d
$D_{r,d}$	the set of posts referenced by the post d
L_d	the length (in words) of the post d
t_d	the time stamp of d
S_d	a score value of the post d
S_b	a score value of the blog b
h_a	a metric for the evaluation of the blogger a

Table 4.4: Summary of the used symbols

4.3.2 Preliminaries

The basic theoretical elements behind the problem are similar to the ones presented in subsection 3.3.2 of chapter III. Nevertheless, for easiness and comfort reasons we briefly repeat the cornerstones of the concepts of this chapter.

Our analysis is based on three sets: A which contains all bloggers, B which includes all blog sites, and D which is composed of all the blog entries. From these main sets we identify two important subsets, $D_a \subset D$ which accommodates the blog posts authored by a blogger a, and $D_b \subset D$ which includes the set of posts published by a blog site b.

For each blog entry $d \in D$ we formulate a set of properties which includes: i) the number of comments C_d submitted by the post readers, ii) the number of other posts $D_{c,d} \subset D$ and $D_{r,d} \subset D$ referring to and referenced by d, and iii) t_d which represents the date and time when the post was published expressed as a time stamp[1]. The elapsed time since the creation of a post d (i.e. the age of a post) is symbolized as Δt_d and is expressed in seconds.

Furthermore, we define three score values h_a, S_d, and S_b: The former reflects the impact of a blogger a, whereas the other two represent the value of a post d and a blog site b respectively. These scores are the foundation upon which we shall build our query independent blog post evaluation mechanisms. All the aforementioned notifications are summarized in Table 4.4.

[1]The time stamp is a 32-bit integer which represents the number of the elapsed seconds since January 1st, 1970

4.3.3 Query Independent Quality Scores (QUIQS)

In this work we introduce a scoring model which apart from the established relevance and opinion scores, also takes into account query-independent blog quality information. The key idea is that a blog post not only must be relevant to a given query and contain an opinion, but it also has to be qualitative and highly influential. In other words, a robust opinion retrieval system must consider the issue of the authority of a blog post and rank the important opinions higher.

The authority of a blog entry is reflected by numerous properties: A first important notification is that a blog post inherits the reputation of the author who published it. Therefore, an opinion expressed by an influential blogger is considered more valuable than one published by an individual who is of lower reputation. However, the influence of a blogger is non-static and changes over time [12], [14]. Since the user submits a query at the present time instance, we are mainly interested in measuring the *current* bloggers' influence.

Furthermore, similarly to the original PageRank concept, we consider that the opinions which are published in reputable blog sites are of higher importance than others which are hosted in sites of lower value. Of course, there is a huge amount of research in the traditional Web IR field which proposed eigenvector-based methods for identifying authoritative Web pages, such as PageRank and HITS [79]. However, these methods are not useful to our problem since blog sites in Blogosphere are very sparsely linked [78]. Similarly to the previous occasion, due to the highly dynamic character of Blogosphere, the value of a blog site is not constant. The approaches we present in section 3.5 adopt the time-aware spirit of the aforementioned blogger influence metrics.

Based on these notifications, we introduce the query-independent quality score (QUIQS) which consists of the following three basic components:

- *The post value:* The importance of a blog post is partially reflected by the impact it has on other bloggers and readers. There are two primary parameters indicating this impact: the number of the Web pages which contain references to the post, and the number of comments submitted by the readers to express their thoughts on the original content. Consequently, since an opinion published in an influential post is accessed by a large number of individuals, we consider it more important than another which was never referenced or commented.

- *The influence of the blogger:* The wide impact of the author who expresses an opinion is a significant factor which affects its importance. Hence, the more readers the writings of a blogger attract, the higher rankings should his/her opinions receive.

- *The impact of the blog site:* The opinions published in a reputable blog site attract a large number of readers and gain more attention. With only a few exceptions, the value of the blog site which hosts a post is a partial indication of the post's value.

In the following subsection we quantify QUIQS and we demonstrate how these approaches can be combined with the existing opinion retrieval strategies to form a new improved ranking model.

4.3.4 Combining opinion and relevance scores with QUIQS

Now let us examine how the proposed query-independent metrics can be combined with the relevance and opinion scores into a single ranking model. Given a query q and a document set D, our objective is to retrieve a subset of documents $D' \subset D$ which are both relevant to q and contain opinions (*Relevant Opinionative Documents, ROD*, [156]).

Initially, the query is processed by a Web IR system which identifies the relevant documents and assigns scores by treating them as typical Web pages. The score $\mathcal{S}(d, q)$ of a blog post d with respect to the query q can be computed by using any of the well-established Web ranking functions such as the inverse document frequency idf, BM25, BM25F, BM25TP [31], etc. The retrieval effectiveness can be enhanced by applying query pre-processing algorithms which a) attempt to identify concepts within the query, and b) expand the query with the aim of extending the pool of relevant documents [156]. Query expansion algorithms may include dictionary-based methods which utilize external sites such as Wikipedia, or local context analysis. In this work we do not study in depth such query pre-processing approaches; we primarily focus on demonstrating the significance of QUIQS in opinionated retrieval.

More specifically, we introduce a new type of scores, $\mathcal{S}_{QI}(d, a, b)$, which indicate the query-independent quality of a blog entry authored by a blogger a and published in a blog site b. Based on our previous discussion, the scores $\mathcal{S}_{QI}$ are expressed as a linear combination of the entire blog site quality S_b, the blogger's influence score h_a, and the overall value of the blog entry S_d. The following equation captures these features:

$$\mathcal{S}_{QI}(d, a, b) = \mathcal{W}_b S_b + \mathcal{W}_a h_a + \mathcal{W}_d S_d \tag{4.17}$$

where $\mathcal{W}_b, \mathcal{W}_a$, and $\mathcal{W}_d$ are three constants used to adjust the importance of the blog site score, the influence score of the blogger, and post quality score respectively. Equation 4.17 dictates that the overall quality of a blog entry d depends on the influence of its author and the importance of the blog site which published it. Furthermore, the query-independent

features of the post in question (i.e. length, number of incoming links and comments) are also considered.

In the sequel, the $\mathcal{S}(d, q)$ and $\mathcal{S}_{QI}(d, a, b)$ are combined to form the final score a candidate post d receives with respect to the query q:

$$\mathcal{S}_{IR}(d, a, b, q) = \mathcal{W}\mathcal{S}(d, q) + (1 - \mathcal{W})\mathcal{S}_{QI}(d, a, b) \tag{4.18}$$

where $\mathcal{W}$ is a constant parameter which tunes the contribution of the query-dependent and query-independent scores in the overall score of the post. A typical setting which works well in most cases is $\mathcal{W} = 0.8$.

Equation 4.18 determines the relevance score of a post d with respect to q, however, it is still required that we choose a strategy to compute the opinion score of d. In [58] the authors have shown that the proximity of the query and opinion terms leads to significant gains in retrieval effectiveness. Their model is built upon the idea that an opinion term refers with higher probability to the terms which are located near its position. This method initially considers each document as a vector $d = (t_1, ..., t_i, ..., t_j, ..., t_{|p|})$ and introduces an opinion probability score at each position i of the document, given by the equation:

$$P(i, d) = \sum_{j=1}^{|d|} p(t_j, d)p(j, i, d) \tag{4.19}$$

where $p(t_j, d)$ is the opinion score of term t_j at the position j of the document. In addition, $p(j, i, d)$ denotes the probability that the term at the position j refers to the the term in the position i, and is calculated as follows:

$$p(j, i, d) = \frac{k(j, i)}{\sum_{j'=1}^{|d|} k(j', i)} \tag{4.20}$$

where $k(i, j)$ is a non-increasing distance kernel function such as Gaussian or Laplacian which implements the concept that the closer to an opinionated term a query term is, the greater the probability that the opinion refers to this term is. For instance, consider the opinion term *excellent*. In case a term *device* appears right after *excellent*, then the expressed opinion refers to *device* with great probability. In the opposite case where *excellent* and *device* appear in distant locations, then *excellent* may refer to another topic within the post.

Based on the positional opinion scores of equation 4.19, the overall probability that the document d expresses and opinion about the query q is calculated as follows:

$$\mathcal{S}_O(d,q) = \frac{1}{|Q|}\sum_{i \in Q} P(i,d) \tag{4.21}$$

where Q symbolizes a set which contains all the positions of the query terms of q within d. Equation 4.21 indicates that to estimate the probability that d contains an opinion about q, it is required that we compute the opinion probabilities in the positions where the query terms occur within d. Nevertheless, this model ignores the physical locations of the document where the opinion and query terms may occur. For instance, a blog post which contains an opinion term and the query terms in proximal positions in its title is more important than another which contains them in distant locations within its main body. For this reason, we provide an enhancement of the opinion probabilities of equation 4.21 by introducing the *field opinion probabilities (FOP)*:

$$\mathcal{S'}_O(d,q) = \sum_{\forall z \in d} \frac{\mathcal{K}_z}{|Q_z|} \sum_{i \in Q_t} P(i,d) \tag{4.22}$$

where $\mathcal{K}_z$ is a constant weight parameter which denotes the value of a particular document zone z. Moreover, Q_z symbolizes a set which contains all the positions of the query terms of q within the zone z of d.

Finally, the opinion and IR scores of equations 4.18 and 4.22 are factorized in the final scoring function which is given by the following formula:

$$\mathcal{S}(d,a,b,q) = \mathcal{S}_{IR}(d,a,b,q)\mathcal{S'}_O(d,q) \tag{4.23}$$

4.3.5 Experiments

In this section we provide the experimental analysis of our methods. Initially, we provide a brief description of the employed dataset, and we present important implementation details which allow us to apply QUIQS efficiently during query processing. In the sequel, we present measurements which indicate that the inclusion of quality-based query independent scores in opinion retrieval leads to significant performance benefits.

4.3.5.1 Dataset characteristics and processing

The dataset we used is the TREC blogs08, a repository comprised of approximately 28.5 million blog posts (documents or permalinks) and 1.3 million blog feeds. The permalinks and the feeds occupy roughly 1,445 GB and 808 GB in uncompressed forms respectively.

	Field
1	DocumentID
2	TREC-ID
3	Author Score
4	Blog Site Score
5	Post Score
6	Pointer

Table 4.5: Required metadata for computing QUIQS: For each post we store its integer identifier, the internal TREC identifier, the three QUIQS for the author, the blog site, and the post itself, and a pointer value which stores the location of the full text of the post.

Now let us describe the methodology of processing the dataset in order to compute the scores of sections 4.3.4. Ideally, the most efficient approach dictates that we pre-compute for each blog post the author, blog site, and post QUIQS. In the sequel, it is only required to maintain these scores into an in-memory data structure which will allow us to quickly retrieve these scores during query processing and compute the desired opinion scores. In Table 4.5 we show a sample record of the aforementioned data structure. In particular, for each blog entry we store:

- An integer document identifier (DocID), which is identical to the one we use to represent the document during inverted index construction.
- An internal identifier assigned by the dataset authors (TREC-ID), which will be used for our own evaluation purposes (i.e. to compare our results with the ones provided by TREC). Of course, TREC-ID can be omitted in real-world implementations.
- The three QUIQS, and
- A pointer which stores the location of the document's full text in the repository. The full text of the post will be used in the second phase of the retrieval by the opinion classifier, to identify whether there are any opinions expressed within the post, or not.

An simple yet very efficient solution is to implement this data structure by employing a standard table indexed and sorted by ascending Document ID; hence, in case we need to compute the score for a document d, we merely need to access the data stored in the record $d - 1$.

The construction of the data records of Table 4.5 requires the preprocessing of the dataset and the extraction of numerous statistics such as its length, the numbers of inlinks, outlinks, comments, etc. From now on, we collectively refer to these statistics as the post's

metadata. In Table 4.6 we record the metadata required to compute all QUIQS, whereas in the second column we show the source from where we retrieve the required data. The term *"directly"* indicates that the metadata in question can be directly extracted by accessing and processing the text of the post; *"after processing"* denotes that we obtain the desired information by a special merging procedure which must take place after text processing. Finally, the indication *"feed"* states that the respective metadata can only be retrieved by accessing the corresponding feed file.

The dataset is organized in individual files which contain one thousand of posts each. Initially, our text processor extracts the posts out of each file, and assigns one unique successive integer identifier to each post. In addition, it retrieves the TREC-ID value, the URL, the time stamp, the length (in words) and the number of outlinks of each document. Then, for each of these files, it outputs three structures: a small inverted index for these 1000 posts, an array which stores the metadata of each post (similar to the one illustrated in Table 4.6), and a Web graph in the form of *(URL, DocID, number of inlinks, list of [inlinkID, timestamp] pairs)*. Notice that the inlinks list stores not only the identifier of the document referring to a specific URL, but also, its time stamp. This is necessary for the calculation of the MEIBIX and BI-scores because these methods involve computations of the ages of the incoming links.

After the text processing is completed, a merging procedure firstly merges all the small metadata arrays into a single metadata table which also has the form of Table 4.6. In the sequel, a second procedure merges all the partial Web graphs into the complete Web graph of the dataset. In total, the Web graph of the blogs08 dataset consists of about 571 million vertices (URLs) and 1.05 billion edges (links), leading to an average of 1.84 incoming links for each encountered URL. During the Web graph merging we also store within the metadata structure the number of the incoming links of each document and a pointer value which points to the respective inlinks list.

Finally, a third application merges the partial indexes into the final inverted index structure. For the needs of our experimental evaluation we adopted the block-based index setup introduced in chapter II which apart from the positional data, it also stores zone information within each posting. This scheme allows us i) to apply our proposed field opinion probabilities (FOP) which expand the proximity-based retrieval model of [58], and ii) to use more sophisticated ranking functions which combine term proximity with zone scoring, such as BM25FTOP. The final merged inverted index structure that we constructed occupied in total roughly 71.8 GB. It consisted of a lexicon with 17,329,126 unique terms, accompanied by an inverted file comprised of 11,693,508,871 postings.

Nevertheless, we are still missing two pieces of information: the author of a blog post,

	Field	Source
1	DocumentID	directly
2	TREC-ID	directly
3	Feed-TREC-ID	directly
4	Author	feed
5	Blog Site	directly
6	Time stamp	directly
7	Number of comments	feed
8	Length (in words)	directly
9	Number of outlinks	directly
10	Number of inlinks	after processing
11	Pointer to the text	directly
12	Pointer to the inlinks	after processing

Table 4.6: Intermediate metadata required to construct the structure of Table 4.5.

and the number of comments submitted by its readers. Although in most cases this information is present within the text of the document, it is quite impossible to retrieve it without any errors due to the differences in pages formatting, encoding, and languages. For this reason, we choose to access the corresponding XML feed which accompanies the dataset and locate the desired data for each blog post. This strategy ensures both maximum effectiveness and comfort, since we do not have to develop complex and costly data mining algorithms to process the text of each Web page. However, for a percentage of the examined posts the author information was not available. In these occasions, we considered that the blog entries were published by individual sites (i.e. not community blogs) and we set the author name equal to the name of the site.

After the creation of the metadata Table 4.6 we can easily generate the required metadata of Table 4.5. Therefore, we successively scan each row by applying the equations of chapter III and we compute each of desired QUIQS. Notice that some QUIQS may involve a second processing step; for instance, SBI-Rank for blog sites requires that we compute in advance the influence metrics of all bloggers, and then sum up the metric values of all member authors to obtain $S_{SBI,b}$.

4.3.5.2 QUIQS-based Rankings

In this subsection we describe the experimental measurements of QUIQS for posts, bloggers, and blog sites and we present some representative rankings. These rankings demonstrate the differences among our proposed QUIQS and they verify the theoretical elements of section 4.3.3. In addition, they confirm that the computation of QUIQS is applicable to large scale data sets.

Title	Author	Date	$\|D_{c,d}\|$	$\|C_d\|$	$S_{M,d}$	$S_{MX,d}$
Mozilla Labs Archive: Introducing Ubiquity	Mar10	27/08/2008	84	504	44,309	53,911
* Jenni Bowlin *	Jenni Bowlin	11/11/2008	47	608	44,116	32,962
Lordz of vengeance: The Originator	JFreak	13/11/2008	45	623	43,895	26,287
Aby Garvey from Simplify 101	Ali Edwards	05/11/2008	17	1729	43,769	24,261
Happiness is: where women create!	Sandra Evertson	10/11/2008	82	347	43,676	23,892
Abyquilt: a challenge	May Britt	02/01/2009	162	115	43,550	29,311
Fox weatherblog: the snow keeps coming	Fox 12 Weather	22/12/2008	13	1597	43,469	28,930
Improving culture through kvetching	JanaRiess	10/09/2008	22	1791	43,432	35,849
Lordz of vengeance: don't cause trouble	Keith S	03/11/2008	45	650	42,954	27,362
Introducing men of life	ndheady	10/11/2008	52	535	42,682	32,681

Table 4.7: The ten most influential blog posts of the TREC blogs08 dataset accompanied by the numbers of incoming links, comments, and their corresponding MEIBI and MEIBIX values.

In all the experiments that we conduct here, we consider that the current date is February 15th, 2009. This date is two weeks beyond the last crawl date of the dataset and it was selected instead of the real current date, since in the opposite case our time decaying metrics would assign near-zero values to all the involved scores.

Table 4.7 contains the ten most influential blog posts according to the MEIBI and MEIBIX metrics. The first three columns denote the title, the author and the date of publication of the corresponding blog entry, whereas the next two columns contain the number of incoming links and the number of comments the post attracted respectively. Finally, the sixth and seventh column contain the values of MEIBI and MEIBIX respectively, rounded up to the closest integer. Notice that the posts presented in this Table are not the most referenced, nor the most commented. The post which attracted the most incoming links, totally 721, was published on *statcounter.com* and it was not commented at all. On the other hand, the most commented post gathered in total 7,919 comments, but only 2 incoming references. It was published on *celebfashtrends*, a blog site which discusses fashion topics.

However, recall that MEIBI and MEIBIX take into consideration both references and comments and are also sensitive to the freshness of the post and the age of its incoming links. Due to these properties, the two aforementioned posts were not included among the

Blogger	$h_{M,a}$
Mike Smithson@politicalbetting	331
Briian@briian.com	273
fuglyhorseoftheday@fuglyhorse	238
hilzoy@obsidianwings	205
Robert McEvily@sixsentences	201
Ernesto@torrentfreak	197
lilyng@lilyng2000	189
WeezerMonkey@weezermonkey	184
Quatremer@coulissesbruxelles	184
Geoff Manaugh@bldgblog	183

Blogger	$h_{MX,a}$
Mike Smithson@politicalbetting	330
Briian@briian.com	311
fuglyhorseoftheday@fuglyhorse	240
Robert McEvily@sixsentences	236
hilzoy@obsidianwings	204
Ernesto@torrentfreak	193
BlogRolling@blogrolling.com	192
lilyng@lilyng2000	190
Geoff Manaugh@bldgblog	190
Angel@thevoiceofadventure	189

Table 4.8: Bloggers influence rankings according to: MEIBI (left), and MEIBIX (right)

ten most influential articles. From the results shown in this Table we observe that MEIBI and MEIBIX behave differently and highlight different characteristics. A representative example are posts 2 and 8: Although the former has been referenced more times, the latter received a higher MEIBIX score; this reveals that the references of the eighth post are more recent than the ones of the second post.

Now let us examine the metrics which evaluate the influence of a blogger. In Table 4.8 we present three rankings of the ten most influential bloggers of the dataset according to MEIBI (left table), and MEIBIX (right table). To avoid the synonymies among different bloggers which would distort our results, we format the author names by using two parts separated by an "at" sign (@): the prefix reveals the blogger's true name, whereas the suffix holds the name of the blog site on which the author submits his/her writings. This technique is more preferable for distinguishing bloggers than the one adopted in [78] which simply discards common blogger names such as *admin*, *John*, etc.

Recall that MEIBI identifies the bloggers who authored qualitative blog posts recently, and MEIBIX rewards those whose posts received recent references. This difference is reflected on the rankings of Table 4.8. For instance, notice that MEIBI ranks *hilzoy@obsidianwings* higher than *Ernesto@torrentfreak*, whereas the opposite holds on the MEIBIX ranking.

Furthermore, in Figure 4.1, we present the bloggers ranking according to the BP and BI indices; the horizontal axis represents the productivity of a blogger (indicated by the BP-Index) and the vertical axis his/her influence (determined by the BI-Index). Consequently, an author who is both productive and influential recently must be placed near the top right corner of the figure. Based on this graph, the most productive and influential blogger is *sportblogs@sportblogs* for whom it holds that $h_{BP,a} = 41$ and $h_{BI,a} = 809$.

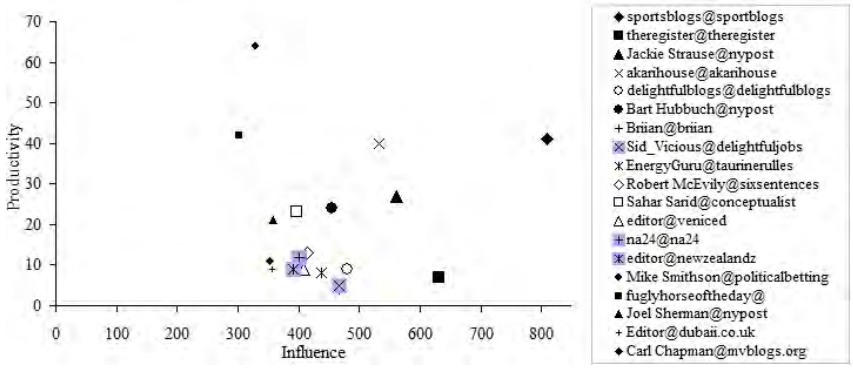


Figure 4.1: Authors Ranking according to the BP and BI indices.

4.3.5.3 Retrieval Effectiveness

In this subsection we present measurements of the retrieval effectiveness of our proposed methods against a set of adversary approaches. For the needs of this experiment we employed a set of 20 opinionated queries used in the blog retrieval task of TREC 2009[2]. Each query of our test set is accompanied by the corresponding "qrels" file which contains the documents which are 1) relevant, 2) both relevant and opinionated and 3) both relevant and factual.

Initially, we notice that the QUIQS can be combined in multiple ways and the number of the possible combinations is very large. We have experimented with a wide variety of such combinations and here we choose the five highest performing scenarios which cover all introduced QUIQS and also, offer the best retrieval effectiveness.

In the first scenario, namely $Q1$, we use the MEIBI-scores $S_{M,d}$ for the query-independent evaluation of a blog post, MEIBI-index $h_{M,a}$ for the bloggers' influence, and BIF for the blog site impact. In the second scheme, we use MEIBIX scores, MEIBIX-index and BIF respectively. The third and fourth settings dictate that we employ the MEIBI-scores and MEIBI-index for the estimation of the posts' value and the blogger's influence; For the computation of the impact of the communities we use SBI-Rank ($Q3$) and BlogRank ($Q4$) respectively. Finally, in the last scenario we measure the retrieval effectiveness by using the ι-score for the posts, the ι-index for the bloggers' influence and SBI-Rank for the blog community importance. All these combinations are summarized in Table 4.9.

Regarding the weights of equation 4.17, we also experimented with multiple setups; A representative example which performs reasonably well in all five scenarios is to set $\mathcal{W}_d = 0.3$, $\mathcal{W}_a = 0.2$, and $\mathcal{W}_b = 0.5$.

[2]The query set of TREC 2009 is comprised of 50 queries, however, only 21 of them are about opinionated retrieval. For one query out of these 21 queries, TREC does not supply the relevant opinionated documents.

	Symbol	Meaning
1	$Q1$	Posts: $S_{M,d}$ - Author: $h_{M,a}$ - Blog Site: $S_{BIF,b}$
2	$Q2$	Posts: $S_{MX,d}$ - Author: $h_{MX,a}$ - Blog Site: $S_{BIF,b}$
3	$Q3$	Posts: $S_{M,d}$ - Author: $h_{M,a}$ - Blog Site: $S_{SBI,b}$
4	$Q4$	Posts: $S_{M,d}$ - Author: $h_{M,a}$ - Blog Site: $S_{BR,b}$
5	$Q5$	Posts: $S_{\iota,d}$ - Author: $h_{\iota,a}$ - Blog Site: $S_{SBI,b}$

Table 4.9: Three example QUIQS combinations applied for opinionated retrieval evaluation.

Our experimentation was divided in four phases: During the first phase, we applied only traditional IR functions including BM25, a popular expansion which integrates document zones (BM25F), and our proposed BM25TOPF which combines term-proximity weighting, query term ordering, and zone scoring into a single ranking formula (subsection 4.2.4). On the second and third phases, we combined the relevance scores of the first phase with the term-proximity model of [58] (marked as TPM) and our proposed field opinion scores of equation 4.22 (FOS) respectively. Finally, on the last stage we attested the usefulness of our query-independent metrics by applying the three QUIQS scenarios of Table 4.9.

The results of all these experimental phases are presented in Table 4.10. Each output ranking was evaluated by calculating three different measures: mean average precision (MAP), R-precision (R-Prec), and precision at the first 10 documents of the ranked list only (p@10). The numbers within the parentheses indicate the precision improvement of each scenario over the baseline method. Furthermore, the experimental phases we conducted are separated from the others by a horizontal line.

The first three rows of Table 4.10 contain measurements of the retrieval effectiveness of the standard IR functions. These results are the baseline of our experimentation and our goal is to improve their performance. As expected, BM25F outperformed the traditional BM25 method by a margin of about 4.8% in terms of MAP. The best-performing scheme was BM25TOPF; its MAP was approximately 6.5% and 11% higher than the ones of BM25F and BM25 respectively.

In the sequel, we implemented the term-proximity opinion retrieval model of [58] which is the second adversary approach for our methods. Notice that in order to apply any of the opinion retrieval strategies, it is required that we possess a special opinion lexicon which contains specific sentiment expressing words. In our experiments we utilized a lexicon structure which is based on multiple product reviews and data sheets submitted in the Amazon.com Web service. It was constructed by using sentiWordNet [50], a publicly available resource for opinion mining which assigns to each three sentiment scores: positivity, negativity, and objectivity. The lexicon in question has been proved particularly effective for the requirements of the TREC 2008 blog track [82].

	Method	MAP	R-Prec	p@10
IR	BM25	0.0531	0.0828	0.0650
	BM25F	0.0558	0.0844	0.0677
	BM25FTOP	**0.0597**	**0.0903**	**0.0742**
IR + TPM	BM25+TPM	0.0698	0.1066	0.0857
	BM25F+TPM	0.0724	0.1099	0.0881
	BM25FTOP+TPM	**0.0808**	**0.1187**	**0.0974**
IR + FOS	BM25+FOS	0.0711	0.1078	0.0868
	BM25F+FOS	0.0733	0.1113	0.0895
	BM25FTOP+FOS	**0.0823**	**0.1207**	**0.0991**
IR + FOS + QUIQS	BM25+FOS+$Q1$	0.0751	0.1139	0.0918
	BM25F+FOS+$Q1$	0.0774	0.1177	0.0947
	BM25FTOP+FOS+$Q1$	0.0868	0.1275	**0.1048**
	BM25+FOS+$Q2$	0.0748	0.1139	0.0918
	BM25F+FOS+$Q2$	0.0763	0.1165	0.0935
	BM25FTOP+FOS+$Q2$	0.0855	0.1233	0.1022
	BM25+FOS+$Q3$	0.0755	0.1140	0.0925
	BM25F+FOS+$Q3$	0.0780	0.1197	0.1002
	BM25FTOP+FOS+Q3	**0.0870**	**0.1277**	**0.1048**
	BM25+FOS+$Q4$	0.0702	0.1046	0.0844
	BM25F+FOS+$Q4$	0.0717	0.1055	0.0859
	BM25FTOP+FOS+$Q4$	0.0785	0.1094	0.0916
	BM25+FOS+$Q5$	0.0544	0.0603	0.0574
	BM25F+FOS+$Q5$	0.0568	0.0692	0.0602
	BM25FTOP+FOS+$Q5$	0.0660	0.0781	0.0720

Table 4.10: Evaluation of the retrieval effectiveness using different ranking methods.

Moreover, since the creators of the term-proximity model report that their approach achieved optimal performance by using the Laplacian kernel function, we replace $k(i, j)$ in equation 4.20 by the following:

$$k(i,j) = \frac{1}{2b} exp\left[\frac{-|i-j|}{b}\right] \tag{4.24}$$

where b is a parameter which we set equal to $6\sqrt{2}$, a value which maximizes the retrieval effectiveness of the Laplacian kernel. The lines 4–6 of Table 4.10 indicate that the term-proximity model achieved significant improvements in all measurements over the simple IR functions. More specifically, when combined with BM25, it achieves a MAP equal to 0.0698 and performs approximately 14.5% higher than the best IR function alone (BM25FTOP). However, if we compare the same IR functions with and without the term proximity model we observe a total performance increase which ranges between 24% (for BM25F) and 26% (for BM25).

Now let us examine the effectiveness of our proposed Field Opinion Scores (FOS)

Zone	K_z
Body (Normal Text)	1
Anchor Text	1
Title	6
URL	2
Headings	4

Table 4.11: The zone weighting scheme for the field opinion scores.

which expand the aforementioned model. The equation 4.22 dictates that each blog post must be divided into a number of distinct zones; then, the occurrence of opinion and query terms in each document zone is weighted accordingly. In this work we fragmented the blog entries of our dataset into five zones: title, URL, body, anchor text, and text encountered within headings. Each zone was assigned a weight value according to Table 4.11, a scheme which provides effective IR retrieval as mentioned in [14].

The third part of table 4.10 (rows 7–9) records the performance of FOS. In total, the field expansion of the term-proximity model leads to an enhancement of about 1.5-2%. More specifically, the mean average precision in case FOS is combined with BM25, increases by about 1.8%, whereas the combination with BM25FTOP leads to an enhancement of 1.9%. These gains are also noticeable for the other evaluation metrics, i.e., R-Precision and Precision@10.

Finally, in our last experiment we included the five QUIQS scenarios of Table 4.9 to the post retrieval process. The values of the three evaluation metrics for our attested combinations are reported in rows 10–24 of Table 4.10. Our first notification is that all QUIQS enhance retrieval effectiveness by a margin of 5 to 6%. The most effective combination is $Q3$, which evaluates the blogs posts by using the MEIBI scores, the authors by employing MEIBI, and the blog sites by using the SBI-Rank metric. In particular, a ranking strategy which utilizes BM25FTOP, FOS and $Q3$ (BM25FTOP + FOS + $Q3$) outperforms the baseline approach with the plain BM25 by enhancing MAP by a percentage touching 39%. In comparison with the strategy which combines term-proximity model to BM25FTOP, the aforementioned scheme generates rankings with higher MAP by a margin of 7.2%.

Regarding the other four scenarios, we observe a slightly decreased performance in comparison with $Q3$. However, all of them provide significant improvements over the adversary approaches. More specifically, the MAP for the strategy (BM25FTOP + FOS + $Q2$) was 0.0855, 37.8% and 5.5% greater than the MAP of the baseline and term-proximity approaches respectively. In addition, the approach (BM25FTOP + FOS + $Q1$) outperformed the baseline method by 38.8% and the term-proximity model by 6.9%.

The last two settings also provide improvements over the baseline method; however, the

performance gains are limited in comparison to the first three approaches. Therefore, the MAP for strategy (BM25FTOP + FOS + $Q4$) which uses BlogRank for the evaluation of the blog sites, was about 10-11% lower than the MAP achieved by the first three scenarios. On the other hand, the MAP for the last scenario which utilizes the ι-score and ι-index for blog post and bloggers' influence evaluation, was 0.0660, 24% lower than the MAP of the first three settings.

4.4 Conclusions

In this chapter we dealt with the issue of improving the retrieval effectiveness of Web and blog search engines. Initially we presented the current state-of-the-art probabilistic ranking functions of the Okapi family, i.e. BM25, BM25F, and BM25TP and we exposed their main weaknesses. We stated that none of them takes into consideration the ordering of the words in the query, hence they treat equally queries such as *John is faster than Mary* and *Mary is faster than John*. Moreover, they fail to combine the concepts of zone weighting (supported by BM25F) and term proximity scoring (embodied in BM25TP).

We introduced a new probabilistic scoring function, namely BM25TOPF, which addresses both of these problems. BM25TOPF incorporates zone weighting, term proximity scoring and correct term ordering and provides an effective mechanism for evaluating and ranking not only Web documents, but also documents which exhibit some kind of fielded structure. BM25TOPF is evaluated against the adversary schemes by using data from the Adhoc (WA) Task of TREC-2009 Web Track [34]. The Mean Average Precision of our method for the 50 queries of the task was at least 6% higher than that of BM25F, the most effective competitor function. A second line of experiments which indicated the superiority of BM25TOPF included MAP measurements by using the topics of the blog retrieval task of TREC 2009. These measurements demonstrate that BM25TOPF is equally effective when applied to retrieve blog entries.

The effective retrieval of opinionated blog entries was the second research field discussed in this chapter. We present the primary features of a standard opinion retrieval system and we focus on an important disadvantage of the current approaches: the lack of support to query independent quality measures. Such metrics have been extensively studied and applied to the traditional Web retrieval task (i.e. PageRank) with great success.

In this chapter we adopt a similar spirit in the retrieval of opinionated blog posts. Based on the metrics of chapter III we define QUIQS, the query-independent quality scores. QUIQS include metrics which evaluate the objective quality of i) a blog post, ii) a blog site, and iii) blogger. These metrics incorporate the essence of Blogosphere such as the

great volatility, the temporal instability, the submission of comments, etc. In the sequel, we form the score of a blog post entry as a linear combination of i) the QUIQS, ii) the query dependent scores (computed by using a standard probabilistic ranking function), and iii) the opinion scores (determined by a dictionary-based method, or a machine-learning algorithm such as SVM).

We performed an exhaustive experimental evaluation of our approach by using the TREC blogs08 dataset, a large repository comprised of approximately 28.5 million blog posts and 1.3 million blog feeds which occupy roughly 1.45 TB and 808 GB in uncompressed forms respectively. Our measurements – which were achieved by employing a set of 20 queries coming from blog retrieval task of TREC 2009 – demonstrated that our method enhances MAP by over 7% compared to the precision of the standard lexicon-based model.

CHAPTER V

Scientometrics and Knowledge Extraction

5.1 Introduction

The effective retrieval of scientific articles and of researchers is of primary importance for the work of a scientist. A robust and accurate search process allows users to quickly locate relevant documents and assists them in improving the quality of their research. For this reason, the enhancement of the vertical IR systems which enable access to scientific resources (i.e. academic search engines, digital libraries, scientific databases) is a particularly important area of research.

Apart from the traditional article retrieval mechanism, the modern academic search engines now provide enriched informational content and capabilities including author profiles, evaluation of scientometric indicators, searches for relevant documents, lists of citing articles, time filtered results and numerous others. Although the scientific libraries maintain repositories of considerably smaller sizes compared to the entire Web, the core issues of effectiveness and efficiency are present here too. As more and more articles are being published and the number of the researchers increases over time, problems such as automatic and unified article classification and parallel algorithm execution obtain a crucial role for these systems.

In this chapter we describe the results of a multi-level research regarding the scientific vertical search engines. In the first level we study the most important methods which have been proposed for the evaluation of the research work of a scientist. These methods, called scientometrics, assign to each researcher a single score value which indicates his/her productivity and influence in the academic community. We introduce new concepts in the area, such as coterminal citations, an effect which is closely related to co-citation and self-citation. Based on these concepts, we propose a new metric, the f-index, which quantifies the impact of coterminal citations in scientific networks.

The second problem addressed in this dissertation is the classification of the research articles into one or more fields of science. Although the generic document classification is a well-studied topic and an enormous amount of work is dedicated to it, it does not take into consideration the parameters of the specialized version of the problem examined here. An example of such a parameter is the co-authorship information recorded in the articles, e.g., when an author A co-operates with B the produced articles are about X, whereas when there is a co-operation between A and C the papers discuss topics related to Y. Such type of information is included in a supervised machine-learning (ML) classification algorithm that we developed for this purpose. This algorithm initially employs a set of labeled articles and trains a model by taking into account the papers' keywords, authors, journals, and co-authorship information. In the sequel, it utilizes this model to assign labels to the unlabeled articles. Our approach is experimentally compared against two popular ML methods, Support Vector Machines and AdaBoost.MH; the experiments produced promising results, since our method outperformed its adversaries by a significant margin.

This ML classification algorithm was the base for two more contributions. The first one concerns the creation of topic-specific scientometrics, that is, indicators which evaluate the work of a scientist not generally, but *only in a particular scientific area*. In this chapter we introduce three topic-sensitive extensions which render h-index, contemporary h-index, and trend h-index sensitive to a specific scientific area. These extensions are considered particularly useful, since they could assist administrators and managers to evaluate their personnel according to their area of expertise. For instance, in case someone is interested in hiring a database engineer, then the correct question to answer is *"between A and B who is the best database engineer"* and not *"between A and B who is the best scientist"*, because A or B could be excellent scientists, but not related to this particular job.

The second application of the aforementioned algorithm regards the identification of attractive research fields for new scientists. This problem concerns the majority of the starting researchers because an erroneous selection could lead to failed research and wasted time, work, and resources. A first approach is to consider the popularity of each research area (i.e. enumerate the relevant published articles), however, such a strategy is considered naive because there are popular topics of science which are unsuitable for new scientists due to their native difficulty. The solution requires that we first identify the characteristics of the new scientists and the features which render a field "hot". In the sequel, we combine these characteristics and we develop a model which provides useful answers and knowledge.

The final problem discussed and addressed in this chapter is caused by the constantly increasing repositories of the digital libraries. Although the computation of scientometrics is rather easy for small datasets, the situation becomes more complex in large-scales, since

the involved data cannot fit in the memory of a single workstation. Hence we have to either use a secondary and much slower type of storage (i.e. disks), or to distribute the computations to a number of interconnected machines. In this chapter we introduce four methods for computing the scientometrics in parallel by using Hadoop/MapReduce, a fault-tolerant framework designed to distribute algorithms in an effortless and robust manner. We attest all four methods by using two types of clusters, one lab local network and a Web cloud commercial infrastructure. We show that the solutions which exploit Combiners, offer efficient execution in terms of both running times and used network bandwidth.

The rest of the chapter is organized as follows: in section 5.2 we discuss the essence of coterminal citations and we introduce f-index. The experimental evaluation of the proposed metric is reported in subsection 5.2.3. In section 5.3 we address the problem of computing the scientometric indicators in parallel by using the MapReduce framework. Initially we present the basic elements of the framework (subsection 5.3.2) and in the sequel we describe the four strategies for solving the problem, along with our experimental evaluation (subsections 5.3.3, 5.3.4, and 5.3.5 respectively). The third subject of this chapter, the identification of attractive research areas for new scientists is presented in section 5.4, whereas in section 5.5 we introduce the supervised ML algorithm for classifying research articles. In section 5.6 we conclude this chapter and we summarize its primary contributions.

5.2 The f-index

The evaluation of the scientific work through scientometric indicators has long attracted significant scientific interest, but recently has become of ground practical and scientific importance. An increasing number of academic institutions are using such indicators to decide faculty promotions, and automated methodologies have been developed to calculate such indicators. Also, funding agencies use them to allocate funds, and recently some governments are considering the consistent use of such metrics for funding distribution. For instance, the Australian government has established the Research Quality Framework (RQF) as an important feature in the fabric of research in Australia; the UK government has established the Research Assessment Exercise (RAE) to produce quality profiles for each submission of research activity made by department/institution.

The use of such indicators to characterize a scientist's merit is controversial, since this assessment is a complex social and scientific process that is difficult to narrow it into a single scientometric indicator. In his article, David Parnas [108] described some possible negative consequences to the scientific progress that could be caused by the "publish or perish" marathon run by all scientists and he proposed not taking into account the scientometric

metric indicators. Also, [3] depicted several shortcoming of the metrics currently in use, i.e., of the Impact Factor and of the h-index. The following phrase, attributed to A. Einstein, could be a representative of the opponents of the scientometric indicators: "Not everything that can be counted counts, and not everything that counts can be counted".

Indeed, neither arguments nor applied methodology currently exist to decide which indicators are correct or incorrect, though, the expressive and descriptive power of numbers (i.e., scientometric indicators) can not unthinkingly be ignored [2]. As Lord Kelvin said "When you can measure what you are speaking about, and express it in numbers, you know something about it. But when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind". In the present section we argue that instead of devaluing the scientometric indicators, we should strive to develop a "correct/complete set" of them and, most importantly, to use them in the right way. Furthermore, we study an aspect of the scientometric indicators that has not been investigated in the past and we propose a new robust scientometric indicator.

5.2.1 The notion of coterminal citations

Traditionally, the impact of a scholar is measured by the number of authored papers and/or the number of citations. The early metrics are based on some form of (arithmetics upon) the total number of authored papers, the total number of citations, the average number of citations per paper, and so on. Due to the power-law distribution followed by these metrics, they present one or more of the following drawbacks (see also [70]): a) they do not measure the impact of papers, b) they are affected by a small number of "big hits" articles, and c) they have difficulty to set administrative parameters.

J.E. Hirsch in [70] attempted to collectively overcome all these disadvantages and proposed the h-index. The h-index was a really path-breaking idea, and inspired several research efforts to cure various deficiencies of it, e.g., its aging-ignorant behavior [125].

Nevertheless, there is a latent weakness in all scientometric indicators developed so far, either those for ranking individuals or those for ranking publication fora, and the h-index is yet another victim of this complication. The inadequacy of the indicators stems from the existence of what we term here - for the first time in the literature - the coterminal citations.

With a retrospective look, we see that one of the main technical motivations for the introduction of the h-index, was that the metrics used until then (i.e., total, average, max, min, median citation count) were very vulnerable to self-citations, which in general are conceived as a form of "manipulation". In his original article, Hirsch made specific mention about the robustness of the h-index with respect to self-citations and indirectly argued that the h-index can hardly be manipulated. Indeed, the h-index is more robust than traditional

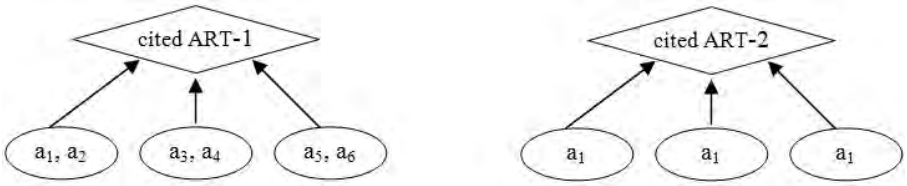


Figure 5.1: Citing extremes: (Left) No overlap at all. (Right) Full overlap.

metrics, but it is not immune to them [120]. Actually, none of the existing indicators is robust to self-citations. In general, the issue of self-citations is examined in many studies, e.g., [68], [142], and the usual practice is to ignore them when performing scientometric evaluations, since in many cases they may account for a significant part of a scientist's reputation [53] and sometimes are used to support promotional strategies [71].

At this point, we argue that there is nothing wrong with self-citations; in many cases they can effectively describe the "authoritativeness" of an article [80], e.g., in the cases that the self-cited author is a pioneer in his/her field and s/he keeps steadily advancing his/her field in an step-by-step publishing fashion, until gradually other scientists discover and follow his/her ideas. In any case, regardless of the reason that they are being made, self-citations work as a driving force in strengthening the impact of the research [142].

In the sequel we will exhibit that the problem is much more complex and goes beyond self-citations; it involves the essential meaning of a citation. Consider for instance the citing patterns appearing in Figure 5.1.

ART-1 is cited by three other papers (the ovals) and these citing articles have been authored by (strictly) discrete sets of authors, i.e., $\{a_1, a_2\}$, $\{a_3, a_4\}$ and $\{a_5, a_6\}$, respectively. On the other hand, ART-2 is cited by three other papers which all have been authored by the same author a_1. Notice that we make no specific mention about the identity of the authors of ART-1 or ART-2 with respect to the identity of the authors a_i; some of the authors of the citing papers may coincide with those of the cited articles. Our problem treatment is more generic than self-citations.

While we have no problem to accept that ART-1 has received three citations, we feel that ART-2 has received no more than one citation. Reasons to have this feeling include for instance the heavy influence of ART-2 to author a_1 combined with the large productivity of this author. Nevertheless, considering that all authors a_1 to a_6 have read (have they?) ART-1 and only one author has read ART-2, it seems that the former article has a larger impact upon the scientific thinking. On the one hand, we could argue that the contents of ART-2 are so sophisticated and advanced that only a few scholars, if any, could even grasp some of the article's ideas. On the other hand, for how long could such situation persist? If ART-2 is a significant contribution, then it would get, after some time, its right

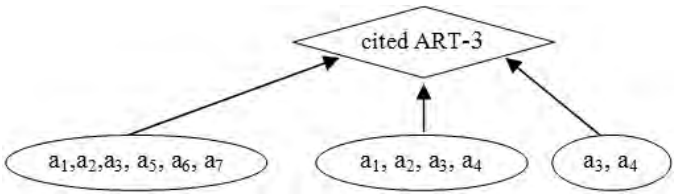


Figure 5.2: Citing articles with author overlap.

position in the citation network, even if the scientific subcommunity to which it belongs is substantially smaller than the subcommunity of ART-1.

The situation is even more complicated if we consider the citation pattern appearing in Figure 5.2, where there are overlapping sets of authors in the citing papers. For instance, author a_3 is a coauthor in all three citing papers.

This pattern of citation, where some author has (co-)authored multiple papers citing another paper is in the spirit of what is termed in this article the coterminal citations. Coterminal citations can be considered as a generalization of what is widely known as co-citation, and their introduction attempts to capture the "inflationary" trends in scholarly communication which are reflected by co-authorship and "exaggerate" citing [36, 37, 38, 109].

Apparently, there exists no prior work on dealing with coterminal citations; the closest relevant works include techniques to filter self-citations or weigh multi-author self-citations [120, 121] and an early scheme to count cardinalities of citing authors. Our target is to develop a metric of scientific excellence for individuals that will not be affected by the existence of coterminal citations, i.e., that it will "appropriately" weigh them. We firmly believe that the exclusion of self-citations is not a fair action; neither is any form of ad hoc normalization. Each and every citation has its value, the problem is to quantify this value. The notion of coterminal citations leads naturally to the process of the discovery of their patterns of existence and of their "controlled discount".

5.2.2 The f-index

We consider the citing example shown in Figure 2 where an article, say A, is cited by three other articles and let us define the quantity nca^A to be equal to the number of articles citing article A. We define the series of sets $F_i^A = \{a_j : \ author\ a_j\ appears\ in\ exactly\ i\ articles\ citing\ A\}$. For the case of article ART-3, we have that $F_1^A = \{a_5, a_6, a_7\}$, $F_2^A = \{a_1, a_2, a_4\}$, $F_3^A = \{a_3\}$.

Then, we define f_i^A to be equal to the ratio of the cardinality of F_i^A to the total number of distinct authors citing article A, i.e., $f_i^A = F_i^A/authors$. These quantities constitute the coordinates of a nca^A-dimensional vector f^A, which is equal to $f^A = \{f_1^A, f_2^A, f_3^A, ..., f_{nca^A}^A\}$.

The coordinates of this vector define a probability mass, since $\sum_{i=1}^{nca^A} f_i^A = 1$. For the above example of the cited article ART-3, we have that $f^{ART-3} = \left\{\frac{3}{7}, \frac{3}{7}, \frac{1}{7}\right\}$. Similarly, for the cited article ART-1, we have that $f^{ART-1} = \left\{\frac{6}{6}, \frac{0}{6}, \frac{0}{6}\right\}$ and for ART-2, we have that $f^{ART-2} = \left\{\frac{0}{1}, \frac{0}{1}, \frac{1}{1}\right\}$.

Thus, we have converted a scalar quantity, i.e., the number of citations that an article has received, into a vector quantity, i.e., f^A, which represents the penetration of A's ideas – and consequently of its author(s) – to the scientific community; the more people use a scholar's work, the greater the impact is. In general, these vectors are sparse with a lot of zeroes after the first coordinates. The sparsity of the vector reduces for the cited articles which have only a few citations. Naturally, for successful scholars we would prefer the probability mass to be concentrated to the first coordinates, which would mean that consistently new scientists become aware of and use the article's ideas.

As the probability mass gets concentrated on the coordinates near the end of f^A, the "audience" gets narrower and in some cases it may imply bad practices (see [108]) like *publishing pacts*, i.e., citation exchange, *clique building*, i.e. researchers form small groups that use jargon to discuss a narrow topic, though broad enough to support the existence of a conference/journal, and then they publish papers "from the clique for the clique", and finally practices which lead to papers with *minimum publishable* increment, i.e., after completion of a substantial study, many researchers divide the results to produce as many publishable papers as possible, that share a large fraction of citations to the same papers.

Though, working with vectors is complicated; we can exploit a "weighting" vector, say s, to convert vector into a scalar value through a dot-product operation, i.e., $\hat{f} = f \cdot s$. For the moment, we will use the plainest vector defined as $s_1 = nca, nca - 1, ..., 1$; other choices will be presented in the sequel. Thus, for the example article ART-3 which we are working with, we compute a new decimal number characterizing its significance, and this number is equal to $N_f^A = f^A \cdot s_1 = 3\frac{3}{7} + 2\frac{3}{7} + 1\frac{1}{7} = \frac{16}{7} \Rightarrow N_f^A \approx 2.28$.

5.2.2.1 The weighting vector

Now, we can define the proposed f-index in a spirit completely analogous to that of h-index. To compute the f-index of an author, we calculate the quantities N_f^A for each one of his/her authored articles A_i and rank them in non-increasing order. The point where the rank becomes larger than the respective N_f^A in the sorted sequence, defines the value of f-index for that author. The name for that new index comes from the fact that it is fractional citation counting scheme.

Earlier, we used the most simple weighting vector; different such vectors can disclose

r	Scientist - h	r	scientist - h	r	scientist - h
1	Hector Garcia-Molina - 77	17	Oded Goldreich - 48	23	Carl Kesselman - 42
2	Jiawei Han - 66	17	Philip S. Yu - 48	24	Olivier Faugeras - 41
3	Ian Foster - 65	17	Prabhakar Raghavan - 48	25	Teuvo Kohonen - 40
4	Robert Tarjan - 64	17	Leslie Lamport - 48	25	Amit Sheth - 40
5	Rakesh Agrawal - 62	17	Douglas C. Schmidt - 48	25	Craig Chambers - 40
6	Jennifer Widom - 60	18	Michael I. Jordan - 47	25	Demetri Terzopoulos - 40
6	Scott Shenker - 60	18	Donald E. Knuth - 47	25	David A. Patterson - 40
7	Jeffrey D. Ullman - 59	18	Ronald Fagin - 47	25	Philip Wadler - 40
8	Deborah Estrin - 58	18	Micha Sharir - 47	25	Jose Meseguer - 40
9	David Culler - 56	19	H. V. Jagadish - 46	25	George Karypis - 40
9	Amir Pnueli - 56	19	Mihir Bellare - 46	26	Geoffrey E. Hinton - 39
10	Richard Karp - 55	19	Pat Hanrahan - 46	26	Stefano Ceri - 39
10	Serge Abiteboul - 55	19	Garcia Luna Aceves - 46	26	Leonard Kleinrock - 39
11	David J. DeWitt - 54	20	Michael Franklin - 45	26	Saul Greenberg - 39
11	David E. Goldberg - 54	20	Alex Pentland - 45	26	Judea Pearl - 39
12	Anil K. Jain - 53	20	Martin Abadi - 45	26	David Dill - 39
13	Hari Balakrishnan - 53	20	Andrew Zisserman - 45	27	Vern Paxson - 38
13	Randy H. Katz - 52	20	Thomas A. Henzinger - 45	27	John A. Stankovic - 38
14	Takeo Kanade - 52	20	Vipin Kumar - 45	27	Krithi Ramamritham - 38
14	Rajeev Motwani - 51	20	Nancy Lynch - 45	27	Ramesh Govindan - 38
15	Don Towsley - 50	21	Christos Faloutsos - 44	27	Jon Kleinberg - 38
15	Christos H. Papadimitriou - 50	21	Thomas S. Huang - 44	28	Al. Sangiovanni-Vincentelli - 37
15	Sebastian Thrun - 50	21	Sally Floyd - 44	28	Edmund M. Clarke - 37
15	Jack Dongarra - 50	21	Robin Milner - 44	29	Herbert Edelsbrunner - 36
15	Ken Kennedy - 50	21	Won Kim - 44	29	Richard Lipton - 36
16	Didier Dubois - 49	22	M. Frans Kaashoek - 43	29	Ronald L. Rivest - 36
16	Lixia Zhang - 49	22	Kai Li - 43	29	Willy Zwaenepoel - 36
16	Michael J. Carey - 49	22	Monica S. Lam - 43	29	Jason Cong - 36
16	Michael Stonebraker - 49	22	Sushil Jajodia - 43	30	Victor Basili - 35
16	Moshe Y. Vardi - 49	22	Rajeev Alur - 43	30	Mario Gerla - 35
16	David S. Johnson - 49	23	Raghu Ramakrishnan - 42	30	Andrew S. Tanenbaum - 35
16	Ben Shneiderman - 49	23	Barbara Liskov - 42	31	Maja Mataric - 33
16	W. Bruce Croft - 49	23	Tomaso Poggio - 42	32	John McCarthy - 32
17	Mihalis Yannakakis - 48	23	Victor Lesser - 42	32	David Haussler - 32
17	Miron Livny - 48	23	Joseph Goguen - 42	33	Stanley Osher - 31
17	Luca Cardelli - 48	23	Henry Levy - 42	33	Tim Finin - 31

Table 5.1: Computer scientists' ranking based on h-index.

different facts about the importance of the cited article. Apart from s_1, we propose also a couple of easy-to-conceive versions of the weighting vector. The vector $s_2 = \{nca, 0, ..., 0\}$ lies at the other extreme of the spectrum with respect to s_1. Finally, if we suppose that the last non-zero coordinate of f^A is f^A_k, then we have a third version of the weighting version defined as $s_3 = \left\{nca, nca - \frac{nca}{k}, nca - 2\frac{nca}{k}, ..., 1\right\}$. For each one of these weighting vectors, we define the respective f-index as f_{s_1}, f_{s_2}, and f_{s_3}. None of these three versions of the weighting vector, and consequently of the respective indexes, can be considered superior to the other two. They present merits and deficiencies in different cases. For instance, the f_{s_1}-index does not make any difference for large h-index values; for scientists with h-index smaller than 15, the obtained f_{s_1}-index can be as much as 50% of the respective h-index, which can partially be explained by the fact that lower-performance (in terms of number of publications) scholars have larger number of self-citations, an explanation which is consistent with the findings of [142].

r	Scientist - f_{s_2} - f_{s_3}	r	scientist - f_{s_2} - f_{s_3}	r	scientist - f_{s_2} - f_{s_3}
1	Hector Garcia-Molina - 68 - 74	17	Donald E. Knuth - 41 - 45	21	Geoffrey E. Hinton - 37 - 37
2	Jiawei Han - 57 - 63	17	Philip S. Yu - 41 - 46	22	Teuvo Kohonen - 36 - 39
2	Ian Foster - 57 - 62	18	Miron Livny - 40 - 45	22	Andrew Zisserman - 36 - 41
3	Robert Tarjan - 56 - 61	18	Luca Cardelli - 40 - 46	22	Sushil Jajodia - 36 - 41
4	Scott Shenker - 54 - 59	18	Ronald Fagin - 40 - 45	23	Joseph Goguen - 35 - 40
5	Jennifer Widom - 53 - 58	18	H. V. Jagadish - 40 - 44	23	Rajeev Alur - 35 - 41
5	Jeffrey D. Ullman - 53 - 55	18	Didier Dubois - 40 - 44	23	Philip Wadler - 35 - 38
6	David Culler - 52 - 53	18	Alex Pentland - 40 - 43	23	Amit Sheth - 35 - 39
7	Deborah Estrin - 51 - 56	18	Thomas S. Huang - 40 - 42	23	Nancy Lynch - 35 - 42
7	Rakesh Agrawal - 51 - 60	18	Sally Floyd - 40 - 43	23	Leonard Kleinrock - 35 - 38
8	David E. Goldberg - 50 - 52	18	Robin Milner - 40 - 42	23	Vern Paxson - 35 - 37
9	Richard Karp - 49 - 55	18	M. Frans Kaashoek - 40 - 41	23	John A. Stankovic - 35 - 37
10	David J. DeWitt - 48 - 51	18	Carl Kesselman - 40 - 42	24	Saul Greenberg - 34 - 37
10	Hari Balakrishnan - 48 - 52	19	Moshe Y. Vardi - 39 - 46	24	Stefano Ceri - 34 - 37
11	Anil K. Jain - 47 - 50	19	Martin Abadi - 39 - 43	24	Raghu Ramakrishnan - 34 - 40
11	Amir Pnueli - 47 - 52	19	Christos Faloutsos - 39 - 43	24	Krithi Ramamritham - 34 - 38
11	Takeo Kanade - 47 - 50	19	Mihalis Yannakakis - 39 - 46	24	Jon Kleinberg - 34 - 36
12	Randy H. Katz - 46 - 51	19	Mihir Bellare - 39 - 45	25	Ramesh Govindan - 33 - 36
12	Lixia Zhang - 46 - 48	19	Oded Goldreich - 39 - 45	25	Edmund M. Clarke - 33 - 34
13	Don Towsley - 45 - 49	19	Garcia Luna Aceves - 39 - 43	26	Judea Pearl - 32 - 36
13	Serge Abiteboul - 45 - 52	19	Kai Li - 39 - 41	26	Richard Lipton - 32 - 35
13	David S. Johnson - 45 - 48	19	Barbara Liskov - 39 - 40	26	Ronald L. Rivest - 32 - 34
14	Ken Kennedy - 44 - 49	19	Tomaso Poggio - 39 - 41	26	Victor Basili - 32 - 35
14	Rajeev Motwani - 44 - 48	19	Henry Levy - 39 - 40	26	Andrew S. Tanenbaum - 32 - 34
14	Sebastian Thrun - 44 - 48	19	Michael Franklin - 39 - 42	26	David Haussler - 32 - 34
14	Ben Shneiderman - 44 - 48	20	Won Kim - 38 - 42	27	Jose Meseguer - 31 - 37
14	Prabhakar Raghavan - 44 - 46	20	Monica S. Lam - 38 - 42	27	David Dill - 31 - 35
15	W. Bruce Croft - 43 - 46	20	Vipin Kumar - 38 - 41	27	Willy Zwaenepoel - 31 - 34
15	Christos Papadimitriou - 43 - 47	21	Victor Lesser - 37 - 41	29	Al. Sang.-Vincentelli - 30 - 34
15	Michael I. Jordan - 43 - 46	21	Thomas A. Henzinger - 37 - 43	28	Mario Gerla - 30 - 33
16	Michael Stonebraker - 42 - 45	21	Micha Sharir - 37 - 43	29	Herbert Edelsbrunner - 29 - 34
16	Jack Dongarra - 42 - 48	21	Olivier Faugeras - 37 - 40	29	Tim Finin - 29 - 30
16	Leslie Lamport - 42 - 45	21	Craig Chambers - 37 - 40	30	Jason Cong - 28 - 33
16	Douglas C. Schmidt - 42 - 46	21	Demetri Terzopoulos - 37 - 38	31	Maja Mataric - 27 - 30
16	Michael J. Carey - 42 - 46	21	David A. Patterson - 37 - 39	31	Stanley Osher - 27 - 31
16	Pat Hanrahan - 42 - 44	21	George Karypis - 37 - 38	32	John McCarthy - 26 - 29

Table 5.2: Computer scientists' ranking based on f_{s_2}. The f_{s_3} value is represented too.

5.2.3 Experiments

The validation of the usefulness of the proposed indexes is not an easy task, given our intention no to harm the reputation of any mentioned scientist. We selected as input data to apply our ideas a number of computer scientists with high h-index [1], who are beyond any question top-quality researchers. Since the data provided by this URL are not up-to-date and contain inconsistencies, we cleansed them first, and kept the scientists with h-index larger than 30.

The ranking in non-increasing h-index is illustrated in Table 5.1; the rankings with the new indicators f_{s_2} and f_{s_3} appear in Table 5.2. Both indicators cause changes in the ranking provided by the h-index. As expected, the values of the f_{s_2}-index are significantly different than the respective h-index values. It is important to note, that these differences (and their size) appear in any position, independently of the value of the h-index. If these differences

[1] http://www.cs.ucla.edu/ palberg/h-number.html

Scientist	h-rank	earned pos. in f_{s_2}
David Haussler	32	+6
Carl Kesselman	23	+5
Geoffrey E. Hinton	26	+5
Lixia Zhang	16	+4
M. Frans Kaashoek	22	+4
Barbara Liskov	23	+4
Tomaso Poggio	23	+4
Henry Levy	23	+4
Craig Chambers	25	+4
Demetri Terzopoulos	25	+4
David A. Patterson	25	+4
George Karypis	25	+4
Vern Paxson	27	+4
John A. Stankovic	27	+4
Victor Basili	30	+4
Andrew S. Tanenbaum	30	+4
Tim Finin	33	+4

Table 5.3: Largest relocations w.r.t. rank position: Most positions up.

concerned only the scientists with the largest h-index, then we could (safely) argue that for someone who has written a lot of papers and each paper has received a large number of citations, then some overlap citations and some self-citations are unavoidable. This is not the case though, and it seems that there is a deeper, latent explanation.

Seeking this explanation, we calculated the differences in ranking positions for each scientist when ranked with h-index versus when they are ranked with the f_{s_2}. The results are illustrated in Table 5.3 and Table 5.4. The general comment is that the scientists who climb up the largest number of positions are those whose work can "penetrate" (and thus benefit) large "audiences". For instance, the research results by *Lixia Zhang* and *John A. Stankovic*, who work on sensors now, are cited in communities like databases, networking, communications. Other scientists whose works is used by large audiences are those working on "computer organization", e.g., *M. Frans Kaashoek, Barbara Liskov, Andrew S. Tanenbaum*, etc. Notice here, that scientists' age has nothing to do with the ranking relocation, since both younger researchers (e.g., *Lixia Zhang*) can climb up positions, just like elder scientists (e.g., *Andrew S. Tanenbaum*).

Another important question concerns whether the particular area of expertise of a researcher could help him/her acquire a larger reputation. Undoubtedly, the research area plays its role, but it is not the definitive factor. Consider for instance, the case of data mining which is a large area and has attracted an even larger number of researchers. We see that *George Karypis* has earned four positions in the ranking provided by f_{s_2}. If the

Scientist	h-rank	lost pos. in f_{s_2}
Rakesh Agrawal	5	-2
Amir Pnueli	9	-2
Didier Dubois	16	-2
Mihalis Yannakakis	17	-2
Oded Goldreich	17	-2
Andrew Zisserman	20	-2
Jose Meseguer	25	-2
Serge Abiteboul	10	-3
Moshe Y. Vardi	16	-3
Micha Sharir	18	-3
Nancy Lynch	20	-3

Table 5.4: Largest relocations w.r.t. rank position: Most positions down.

area of expertise was the only rational explanation for that, then why *Rakesh Agrawal*, who founded the field, is among the scientists that lost the most number of positions in the ranking provided by f_{s_2}? The answers lies in the particularities of the research subfields; *George Karypis* contributed some very important results useful also in the field of bioinformatics. To strengthen this, we can mention the case of *Jiawei Han*. He is a data-mining expert whose work penetrates to communities like mining, databases, information retrieval, artificial intelligence, and his is ranked second, based either on h-index, or on f_{s_2} or on f_{s_3}.

Examining the scholars with the largest loses, we see that scientists who have made ground-breaking contributions and offered some unique results, e.g., *Mihalis Yannakakis*, and *Moshe Y. Vardi*, drop in the ranking provided by the f_{s_2}. This has nothing to do with the theoretical vs. practical sides of the computer science; contrast the cases of *M. Yannakakis* and *M. Vardi*, versus *A. Zisserman* and *R. Agrawal*. It is due to the nature of the scientific results that do not "resound" to other communities.

5.3 Computing scientometrics in large-scale academic search engines with MapReduce

Following the evolution of the Web search engines, the scientific databases and academic search engines have significantly enriched the content of their result pages. Therefore, the results of a query for a research paper are now accompanied by information regarding the articles' authors. Some of the most popular scientific search engines such as Google Scholar[2], Microsoft Academic[3], and Scopus[4] extended this information by con-

[2]http://scholar.google.com
[3]http://academic.research.microsoft.com
[4]http://scopus.com

structing author profiles where they compute and present their associated *scientometrics*.

As already mentioned in section 5.2, the most widespread among them is *h-index*, devised by J.E. Hirsch [69]. This metric assigns a value to each scientist by taking into account not only the number of publications he/she has authored, but also, by considering the number of citations each article received. After the introduction of h-index, an entirely new line of research was drawn and multiple variants were proposed.

The computation of scientometrics is relatively easy when it is performed on small, well-controlled collections of research papers. For instance, in the case of h-index, the evaluation mechanism just needs to determine the articles each researcher has authored and then enumerate all their incoming citations. However, when the size of the collection increases the evaluation becomes more complex since a single workstation cannot accommodate all the involved data (i.e. documents, authors and citations). Therefore, we either have to use a secondary (and slower) type of storage, or solve the problem in parallel by distributing the data to a number of interconnected machines.

MapReduce is a distribution framework designed for solving problems in large scales. It is mainly oriented towards fault-tolerance, distributed storage, and simple implementation without requiring network programming details. This model has been used extensively by the Web search engines to develop a wide range of parallel algorithms. Examples include data mining tasks, information extraction from graphs, data structures construction, text processing, and others.

In this chapter we propose four methods based on MapReduce to compute in parallel the scientometrics in large scientific databases. To the best of our knowledge, this is the first work in the current literature attempting to address this problem in large scales. All previous bibliography does not study in depth the issue in question, since until recently the data collections were small and the problem was not very important. However, the introduction of the large scientific databases and their constantly expanding repositories in combination with the users' increased interest, has rendered the issue much more challenging.

5.3.1 Related work

In this section we present some fundamental articles about MapReduce and its architecture and we discuss some remarkable works which introduce strategies for solving common problems in parallel. Finally, we refer to a number of scientometrics that have been proposed in the related bibliography.

MapReduce was initially introduced by two Google engineers in [60]. In [61] the authors described GFS, the distributed file system on which the framework operates. A more extended presentation of the components of MapReduce is provided in [41]. The most

popular open source implementations of MapReduce and GFS are *Hadoop* and the *HDFS* respectively. A technical overview of their architectural logic and design is provided in [28].

Numerous works have proposed expansions and modifications which allowed the framework to be used in a wider variety of applications. For instance, [1] introduced *HadoopDB*, an architectural hybrid between Hadoop and database management systems. In [150] the authors appended a "Merge" phase to the execution plan of the system with the aim of joining the relational outputs of two separate MapReduce tasks.

Several other research articles have described important problems which were efficiently solved by using MapReduce. For instance, Web search engines have used the framework extensively in data intensive tasks such as inverted index construction [93], and PageRank computation [85]. Text intensive applications include duplicate and near-duplicate document detection [49], language processing algorithms [84] and numerous others. Finally, [91] introduced Pregel, a computational model for processing large graphs. Pregel programs are expressed as a sequence of iterations, in each of which a vertex can receive messages sent in the previous iteration. However, unlike PageRank computation, the evaluation of scientometrics can be performed in a single MapReduce job without requiring multiple iterations.

5.3.2 MapReduce basics

MapReduce builds on the key idea of simplicity; that is, its users should not deal with complex network programming issues [60, 41]. Instead, the system provides an abstraction that requires from the algorithm developers to express their solutions by using only two functions: *map* and *reduce*.

The co-ordination of the parallel execution is performed by a single machine, the *Master*. The Master splits the input data into multiple fragments and assigns the processing of each fragment to a number of m *Workers*. The Workers (called *Mappers* in this phase) apply the map function to every key/value pair of their input and generate an arbitrary number of intermediate key/value pairs. When the input is exhausted, the system employs a number of r Workers (now called *Reducers*) that apply the reduce function to all values associated with the same intermediate key. Their final output is the solution of the assigned task, also formatted in key/value pairs and partitioned in r shards.

There are two more optional components which can be involved in a MapReduce task: The *Partitioner* and the *Combiner*. The former is used to determine how the intermediate files produced by the Mappers should be transferred to the local file systems of the Reducers. The latter, is used to improve the efficiency of the execution by limiting the size

of the data to be transferred from the Mappers to the Reducers by merging the values associated with the same key into associative arrays. The Combiner is deployed by either explicitly declaring a *combine* function, or by properly implementing it within the Mapper itself (*in-Mapper Combiner*). According to [85], the second option is usually preferable.

The MapReduce jobs are executed on top of a distributed file system [60] which transparently addresses all the problems that may occur (e.g., fault tolerance). For instance, in case a worker dies due to a hardware malfunction, Master assigns the job it was processing to another worker without any data loss.

5.3.3 Computing scientometrics with MapReduce

Let us begin by introducing P which is the set containing all papers, and A which includes all authors. Each paper $p_i \in P$ contains a reference section encountered towards the end of the manuscript. From this section we extract $P^{p_i} \subset P$ which contains all papers referenced by p_i; for each reference $p_j^{p_i} \in P^{p_i}$ we retrieve all the contributing authors $A^{p_j^{p_i}}$. In Table 5.5 we summarize all the above notations.

The input of the problem can be considered as a set of (p_i, C^{p_i}) pairs, where p_i represents the integer identifier of an article and C^{p_i} symbolizes its content. Our objective is to construct a list of (a, M_x^a) pairs, where x identifies the metric M employed to evaluate each scientist (see last row of Table 5.5). According to the definitions of all three metrics, it is required that we decompose the required (a, M_x^a) pairs and construct for each author, one pair of the following form:

Symbol	Meaning
P	The set containing all papers
A	The set containing all authors
p_i	An arbitrary paper $p_i \in P$
C^{p_i}	The textual content of p_i
a_j	An arbitrary author $a_j \in A$
A^{p_i}	The authors who created p_i
P^{a_j}	The papers authored by a_j
P^{p_i}	The papers referenced by p_i
$S_x^{p_i}$	The score of a paper p_i with respect to the metric x
$M_x^{a_j}$	A metric evaluating the work of a_j $M_h^{a_j}$: h-index of a_j $M_c^{a_j}$: contemporary h-index of a_j $M_t^{a_j}$: trend h-index of a_j

Table 5.5: List of the most frequent symbols

$$\Big(a, SortedList\Big[\big(p_1, S_x^{p_1}\big), ..., \big(p_N, S_x^{p_N}\big)\Big]\Big) \tag{5.1}$$

where $S_x^{p_i}$ is the score of p_i with respect to the metric x. In case we are interested in computing h-index, this score merely represents the total number of the incoming citations that p_i received.

According to 5.1, to calculate the metric values for an author, we first need to identify all the publications he/she has authored, and then compute their corresponding scores. Notice that the elements of this $(paper, score)$ list must be sorted in descending score order to enable fast metric evaluation with a single iteration. In the following section we present four methods to solve this interesting problem by using MapReduce.

5.3.3.1 Basic algorithm design

We start by feeding the system with the given set of the publications P. According to our previous discussion, we express the input of the Map function in a $(key, value)$ manner by defining (p_i, C^{p_i}) pairs. Within the Mapper, we parse the textual content of each paper $p_i \in P$ and we retrieve all its outgoing references P^{p_i}. For each reference $p_k^{p_i} \in P^{p_i}$ we compute a score $S_x^{p_k}$, according to the metric x we need to evaluate.

For the plain h-index metric, we set the score equal to 1 for all references, thus denoting that the paper $p_k^{p_i}$ has one incoming citation (which of course, is p_i). For the other two metrics, we need to consult equations 5.7 and 5.8. Our goal is to properly set the partial scores in the map phase in order to compute the final scores in the reduce phase. For this reason, during the map phase, we set the partial scores recorded in Table 5.6.

In the sequel, each reference is again parsed and its authors $A^{p_k^{p_i}}$ are identified. For each extracted author, we create one tuple that will be sent to the Reducer and there are two options to format this tuple. The first one (called *method 1*) dictates that we set the author as the key, and create a pair $(paper, score)$ for the value field. Our second option (called *method 2*) is to generate a composite key of the form $(author, paper)$, and place the paper score within the value field. Algorithm 5 illustrates a pseudocode for the map function implementing these two methods.

Although the map phase is almost identical for methods 1 and 2, the reduce phases must

Metric	**Partial Score**
h-index	$S_h^p = 1$
contemporary h-index	$S_c^p = \gamma/(\Delta Y_p)^\delta$
trend h-index	$S_t^p = \gamma/(\Delta Y_{p,c})^\delta$

Table 5.6: Setting the partial paper scores in the map phase for various scientometrics

Algorithm 5 Mapper class(es): In case method 1 is used, the framework executes the step 8a. If we use method 2, we need to execute the step 8b.

```
 1: class Mapper
 2:     method map (integer p_i; string C^{p_i})
 3:         P ← ExtractReferences(C^{p_i})
 4:         for all references p ∈ P
 5:             S^p ← ComputeScore(p)
 6:             A^p ← ExtractAuthors(p)
 7:             for all authors a ∈ A^p
8a:                 emit (a, pair[p, S^p])
8b:                 emit (pair[a, p], S^p)
```

implement a different strategy. Notice that the MapReduce framework guarantees that the data sent from the Mappers arrives at the Reducers in a sorted key order. This gives method 2 an advantage; method 2 implements a *secondary sort*, that is, the data is not only brought to the Reducers in an ascending author order (as holds for method 1), but also, in ascending paper order. This allows a more robust approach of the reduce phase, since we save the cost of *searching* for the incoming papers. To make our state clearer, we provide Algorithms 6 and 7 for the reduce phase of methods 1 and 2 respectively.

Let us discuss Algorithm 6 first. Since the $(paper, score)$ pairs are brought to the Reducers in arbitrary order, we need to store these pairs into a data structure H which will allow us to accumulate the partial paper scores. More specifically, for each value field of the Reducer input, we search in H for the input paper. In case this search fails, we insert the paper along with its corresponding score. In the opposite case, we just accumulate the incoming score to the one which is already stored in H. After all the tuples have been processed, we sort H in a descending score order and we compute the desired metric by iterating through its entries (steps 11–16). The sorting of H can be performed within the main memory of the Reducer since it stores at most a few hundreds entries; the vast majority of the authors has published fewer than 1000 articles.

On the contrary, in Algorithm 7 there is no need for searching; instead, it is only required to allocate an array H to store the paper scores. Since the tuples arrive in sorted order, we just need to compare the paper we are currently processing to the previous one (step 9). In case their identifiers are equal we accumulate their partial scores and update the last record of H. In the opposite case, we store the new paper score in a new position at the end of H. When all the papers of an author have been processed, we repeat the steps 9–17 of Algorithm 6 to compute the desired metric and we proceed with the next author. The final (a, h^a) tuple must be written out in the close method.

Finally, notice that the pair values of Algorithm 6 and the pair keys of Algorithm 7 are

Algorithm 6 Method 1, Reducer class

```
1: class Reducer
2:     method reduce (string a; pairs[integer p, float S^p])
3:         H ← new AssociativeArray
4:         for all pair v ∈ pairs[integer p, float S^p]
5:             if v.p ∈ H
6:                 H^p.S ← H^p.S + v.S^p
7:             else
8:                 H.add(v)
9:         sort H in descending S order
10:        integer papers ← 0, metric ← 0
11:        for all pairs ∈ H
12:            papers ← papers + 1
13:            if H^p.S ≥ papers
14:                metric ← metric + 1
15:            else
16:                stop iteration
17:        emit (a, metric)
```

not included in the basic data types of MapReduce. Consequently, it is required that we implement additional classes which explicitly define how these data types must be read and written by the framework. Nevertheless, the complexity for Algorithm 7 is increased since the custom data type is used in the key; hence, it is required to determine how the system will compare the keys to each other to achieve sorted Mapper output (compareTo method). However, the increased complexity of Algorithm 7 is rewarded with improved execution performance.

5.3.4 Optimizing the performance

Despite their difference in tuples formatting, the Mappers of both methods 1 and 2 still emit data to the Reducers each time an author of a paper reference is extracted. Since we do not check whether the key we are currently processing has been previously sent, it is inevitable that we transmit the same key multiple times. This leads to a performance bottleneck due to the increased network traffic caused among the nodes of the system. Here we attempt to address this problem with the support of the Combiners.

The Algorithm 8 shows how we can extend method 1 with the aim of supporting an in-Mapper Combiner. We call this new approach as method 1-C, where the letter "C" signifies the presence of a Combiner. The cornerstone of method 1-C is to avoid multiple emissions of identical author names and thus, save valuable network bandwidth. To achieve this, we first initialize a container data structure H which shall allow us to emit $(author, list[paper, score])$ tuples instead of the simple $(author, (paper, score))$ tuples of

Algorithm 7 Method 2, Reducer class

```
1: class Reducer
2:     method initialize
3:         string a_prev ← ""
4:         integer p_prev ← 0
5:         integer n ← 0
6:         H ← new Array
7:     method reduce (pair[string a, integer p]; float S^p)
8:         if a = a_prev
9:             if p = p_prev
10:                H(n) ← H(n) + S^p
11:            else
12:                H.add(S^p)
13:                n ← n + 1
14:            else
15:                Perform steps 9–17 of Algorithm 6
16:                H.reset()
17:                a_prev ← a
18:                p_prev ← p
19:                n ← 0
20:    method close
21:        emit (a, metric)
```

method 1. During the references parsing process, each time an author is encountered we perform a look-up in the container (step 10); in case the author is not present in H we insert the record along with the corresponding $(paper, score)$ pair (steps 11–13). In the opposite case, we need to check whether the current reference belongs to the $(paper, score)$ list of the found author. If the search is unsuccessful we store the paper and its score in the list (step 16); otherwise, we update the corresponding list record by summing up the new paper score to the stored one (step 18). After all the input data has been processed, the Mapper emits the tuples stored within H to the Reducer via the Close method.

It is immediately obvious that the method 1 generates an immense number of key-value pairs compared to method 1-C. Method 1-C is much more compact since with method 1, the author is repeated for each reference we send to the Reducer. Nevertheless, we need to mention here that there are two side effects deriving from the usage of a Combiner. The first one is the increased memory footprint of the map function due to the allocation of the container data structure. The second is a possible delay in the execution of the map phase due to the double search we perform (one for the author and one for paper). However, in this specific application that we examine, our experiments reveal that this delay is infinitesimal due to the small length of the $(paper, score)$ lists, and that the usage of a Combiner definitely leads to significant acceleration of the entire task.

Algorithm 8 Method 1-C: Improved version of method 1 with Combiners

```
1: class Mapper
2:     method initialize
3:         H ← new AssociativeArray
4:     method map (integer p_i; string C^{p_i})
5:         P ← ExtractReferences(C^{p_i})
6:         for all references p ∈ P
7:             S^p ← ComputeScore(p)
8:             A^p ← ExtractAuthors(p)
9:             for all authors a ∈ A^p
10:                if a ∉ H
11:                    L^a ← new Array
12:                    L^a.add(p, S^p)
13:                    H.add(a, L^a)
14:                else
15:                    if p ∉ H.L^a
16:                        H.L^a.add(p, S^p)
17:                    else
18:                        H.L^a.update(p, +S^p)
19:    method close
20:        for all authors a ∈ H
21:            emit (a, list(p, S^p))
22: class Reducer
23:    method reduce (string a; list[integer p, float S^p])
24:        H ← new AssociativeArray
25:        for all pair v ∈ list[integer p, float S^p]
26:            if v.p ∈ H
27:                H^p.S ← H^p.S + v.S^p
28:            else
29:                H.add(v)
30:        Perform steps 9–17 of Algorithm 6
```

Finally, we introduce method 2-C where we inject the in-Mapper Combiner approach in method 2. The Algorithm 9 illustrates the basic steps which are similar to those of Algorithm 8. In this case however, the container data structure does not store a list of $(paper, score)$ pairs for each author, but a single cumulative score value per each distinct $(author, paper)$ pair. This minimizes the benefits of using a Combiner because the $(author, paper)$ keys are more numerous than the simple author keys of method 1-C. In addition, notice that the reduce phase in this case is identical to that of method 2.

5.3.5 Experiments

For the experimental evaluation of our theoretic analysis we employed Hadoop 0.20.2, an open-source implementation of the Google's MapReduce framework. We begin this sec-

Algorithm 9 Method 2-C: Improved version of method 2 with the introduction of in-Mapper Combiners. The Reducer is identical to the one of Algorithm 7.

```
1: class Mapper
2:     method initialize
3:         H ← new AssociativeArray
4:     method map (integer p_i; string C^{p_i})
5:         P ← ExtractReferences(C^{p_i})
6:         for all references p ∈ P
7:             S^p ← ComputeScore(p)
8:             A^p ← ExtractAuthors(p)
9:             for all authors a ∈ A^p
10:                if pair(a, p) ∉ H
11:                    H.add(pair(a, p), S^p)
12:                else
13:                    H.update(pair(a, p), +S^p)
14:    method close
15:        for all pairs (a, p) ∈ H
16:            emit (pair(a, p), S^p)
```

tion with a brief description of our test dataset and we proceed with data size measurements and efficiency assessments.

5.3.5.1 Dataset characteristics

Collecting bibliometric data is a challenging task, due to the strict data protection policies applied by the digital libraries. Since crawling is forbidden, we are limited in using only open access document collections. The largest among these collections is the CiteSeerX [33] dataset, an open repository comprised of approximately 1.8 million scientific articles. The dataset is available in three forms: The first one contains the raw text of the publications scattered in 1.8 million plain text files. The other two contain certain meta-data of the documents expressed in SQL and XML formats respectively. The raw text format of the articles requires much and intensive effort towards two directions: a) disambiguation of the authors names and b) references extraction. Although these problems are both interesting and challenging, they are out of the scope of this paper. For this reason, we choose to work with the XML formatted dataset.

For each article of the dataset there are one or more small-sized XML files, each of which represents a different version of the same article. The dataset includes in total 3.9 million XML files, however, in our experiments we use only the latest version; consequently, 1.8 million XML files are used. This large number of small-sized files renders the dataset inappropriate for MapReduce, because the underlying distributed file system

is designed for optimal performance when dealing with considerably larger files. For this reason, we performed a conversion of the dataset by packing thousands of these XML files into larger binary files. After this process, our "new" dataset was comprised of 432 files of 64 MB each.

5.3.5.2 Data sizes

In this subsection we perform measurements of the data sizes exchanged between the Mappers and the Reducers of our proposed methods. Initially we provide method independent numbers indicating the data sizes involved in the examined problem. The first two rows of Table 5.7 concern the Mapper input, whereas the last two are connected to the Reducer output. As mentioned, the input consists of approximately 1.8 million articles which occupy in total roughly 27.6 GB. After the processing of the dataset with MapReduce, the system outputs a set of about 2.8 million $(author, metric)$ pairs the size of which touches 40MB.

Table 5.8 illustrates various statistics; in the first double column we measure the size of the Mapper output of all four examined methods, expressed in number of records and data size in MB. The latter measurement is essential since it reflects the overall size of the data exchanged among the Mapper and Reducer nodes of the cluster. In the next column we record the counts of the Reducers input groups which represent the number of the unique keys which *arrive* at the Reducers. To acquire these measurements, we executed all four methods by employing only one Worker node; Table 5.8 derives from the report generated by the framework at the end of the task.

Statistic	**Value**
Input Records	1,844,272
Input Size	27.6 GB
Output Records	2,865,282
Output Size	39.9 MB

Table 5.7: Problem input-output statistics

Method	**Mapper Output**		**Reducer**
	Records	**Size (MB)**	**Input Groups**
method 1	36,687,999	688.4	2,865,282
method 2	36,687,999	688.4	12,260,311
method 1-C	21,736,395	600.8	2,865,282
method 2-C	34,251,437	643.2	12,260,311

Table 5.8: Record counts and data sizes for the four examined methods

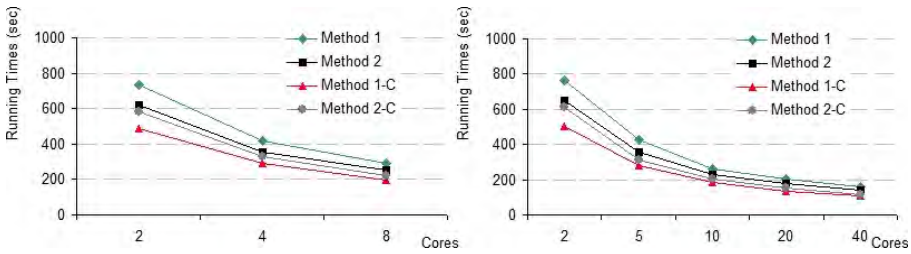


Figure 5.3: Running times of the four methods in a small local cluster (Left), and a Web cluster infrastructure (Right).

Initially we examine the performance of our methods in terms of sizes of the Mapper outputs. The map phase of methods 1 and 2 transmitted in total 36.7 million records occupying roughly 688 MB. On the contrary, the usage of a Combiner in method 1-C decreased these values by 42% (21.7 million records) and 13% (601 MB) respectively. As we anticipated, method 2-C was not equally efficient despite the usage of a Combiner. Compared to methods 1 and 2 we only achieve a reduction in the size of the outputted data by a margin of 6.5%. This is due to the fact that the Combiner of method 1-C lists $(paper, score)$ pairs per each unique author, whereas method 2-C stores one partial score value for each distinct $(author, paper)$ key; the latter key type is much rarer than the former.

The counts of the Reducer input groups reveal that the number of the unique keys which arrive at the Reducers of methods 1 and 1-C is equal to the number of records that depart from it (see third row of Table 5.7). This is due to the fact that the output of the entire task (i.e. (a, h^a) pairs) has the same key as the Mapper output of these two methods. On the other hand, the tuples produced by the Mappers of methods 2 and 2-C are keyed by using $(author, paper)$ pairs, consequently, the unique keys which arrive at the Reducers increase by a factor of approximately 4.2.

5.3.5.3 Efficiency measurements

In this subsection we evaluate the performance of the four methods. To exhaustively attest the scalability of our algorithms, we measured their running times by using two platforms. The first one includes a small-sized lab network, whereas the second one is a larger Web cluster infrastructure. Each experiment was repeatedly performed by employing different numbers of processing cores each time. The results are depicted in Figure 5.3.

Our first observation is that in both platforms, all of our methods scale well for fewer than 20 cores; the doubling of the cluster size almost leads to halved running times. For more cores the gains are slightly limited, due to the increased network latencies. Notice that the running times between the two clusters are not comparable, since these clusters are

equipped with different hardware and they adopt different architectures. In all occasions, method 1-C outperformed the other approaches by a margin ranging between 32% and 35%. Apparently, the existence of the Combiner results in decreased exchange of data among the nodes of the clusters. Although method 2-C also employs a Combiner, it did not perform equally well; compared to method 1-C it was outperformed by about 18%. We have previously explained that the $(author, paper)$ keys of method 2-C are more numerous than the simple author keys of method 1-C, consequently, the benefits of using a Combiner are limited.

Regarding the plain methods 1 and 2, we notice that the latter completed the assigned task slightly faster. Although the amount of data exchanged among the nodes of the system is equal in both methods, method 2 achieves better performance due to the more robust implementation of its Reducer. More specifically, the $(author, paper)$ keys emitted by the Mapper of method 2 are brought to the Reducer in sorted author and paper order (secondary sort), thus saving us the cost of searching for the input papers.

5.4 Identifying attractive research areas for new scientists

One of the most important issues that a new[5] researcher has to address is the correct identification of the primary research field that will determine his/her future career. Our current experience has proved that a significant percentage of starting scientists often choose their area of interest by considering invalid parameters, including the reputation of their future mentors or supervisors, the availability of open PhD theses, or the former success of others who have managed to conduct a productive research in this specific area. Therefore, it is a common phenomenon that capable and diligent scientists are misled and engaged with scientific fields that are considered as obsolete, dead, or prohibitively competent for their current level of experience.

We firmly believe that the primary criterion for the selection of a research area is the new scientist's preferences. A research conducted in a field that is out of the interests or likes of a researcher is undoubtedly condemned. Nevertheless, this criterion is extremely hard to be modeled, since even the scientists themselves are frequently not in the position to determine whether a research area is within their own interests. Along with this notification, a sequence of questions and critical issues are posed.

Certainly the various scientific fields are not equally promising and each of them exhibits its own level of "hostility" for a new scholar. For instance, several scientific domains are considered as obsolete, as the majority of their related problems have found efficient and

[5]In this work we also use the term *starting scientists* or *starters* to refer to new scientists

effective solutions. On the other hand, there are problems that can only be tackled by experienced scientists and publishing a work in such an area is relatively difficult. Apparently, new scientists are not recommended to work in such areas, since it is usually impossible to propose a solution that outperforms the existing schemes and moreover, publishing such solutions has limited probabilities due to the lack of trust by the rest of the members of the scientific community.

The identification of trendy research areas is of great interest for every scientist. Such knowledge is a valuable tool, since it can reveal the correct path for new scholars and assist them in working on modern or newly posed problems. Even the more experienced researchers could benefit from the knowledge of the most fashionable fields, as they could expand their work and develop solutions to novel problems. This is a definite advantage for the science itself.

In this chapter we attempt to formally set and solve this interesting problem. Although there are several previous works which investigate the issue of identifying emerging topics of research, the problem of identifying attractive research areas for new scientists is new; to the best of our knowledge, there is no other work attempting to address it. In our approach we initially examine the main attributes of the problem and we study the space where the solution lies. In the sequel, we consider the most important properties of the new scientists and with that knowledge, we identify the core elements that render a research field attractive to them.

A significant parameter of our problem is the identification of the new scientists and their separation from the more experienced ones. In this work we exploit some of the most sophisticated metrics that have been proposed in the literature. We also introduce a set of Topic-Sensitive extensions which render these metrics aware of the research field that we examine each time. These contributions are tested experimentally by employing a large dataset of scientific articles deriving from the wide areas of Engineering and Computer Science.

5.4.1 Related work

Although the identification of attractive research fields for new scholars has not been previously addressed, the issue of investigating emerging research areas has been studied by several previous works. The approaches proposed in these works are divided into two wider categories, the *co-word* and the *co-citation* analysis methods. The first branch includes policies which focus on directly investigating the contents of a research topic. One of the earliest relevant works is the research of [44], which employed co-word analysis and detected changes in the field of information retrieval during the period between 1987 and

1997. Furthermore, [82] introduced a co-word analysis method for measuring the latest research trends in technical documents.

The most significant problem of the co-word analysis methods is the lack of an objective mechanism which will determine the set of representative keywords from the examined documents [82, 102]. For this reason, the extraction of objective keywords from the examined documents depends highly on each analyst; this certainly introduces some bias. The requirement to eliminate bias forced the researchers to introduce more objective criterions, such as the evaluation of the increment rate of published articles with particular keywords. Several works attempted to identify emerging topics by analyzing the changes in the number of related articles [101, 137]. These studies proved that the increment rate was an effective criterion for determining the value of each keyword. Nevertheless, these works also initially require a set of pre-defined keywords before their proposed algorithms can be applied.

The second category of methods includes the works which attempt to address the problem by applying co-citation approaches. Examples of such works are [130] and [141] which examined the citation properties of several papers in order to identify emerging fields of research. Based on this analysis, they detect sets of highly cited papers; the numbers of these papers and the research area they belong to is then used to obtain the required knowledge. The major problem is that recent works cannot usually receive many with respect to the older works. This difficulty turns co-citation approaches less effective.

In this work [15] we propose a score-based identification of attractive research fields for new scientists. Each research field receives a score according to numerous parameters, such as the reputation of the involved scientists, the prestige of the journals[6] which publish the related papers and the number of incoming citations. Furthermore, these parameters are considered with respect to temporal aspects which reveal the research fields which are attractive *presently*.

Regarding the issue of the evaluation of a researcher's work, there is a significant amount of work attempting to address it. The pioneering article which achieved robust results is [70], where J. Hirsch introduced *h-index*, a metric that rewards both the productivity and influence of a scientist. Motivated by the success of the h-index, several other metrics followed, such as the *SCEAS* system [126], *g-index* [46] and *f-index* [76]. In [22] a normalized version of the metric is presented, whereas in [27], a high-level study of the mathematics and performance is provided. In [47] it is attempted to minimize the gap between the lower bound of the total number of citations calculated by h-index and their real

[6]In this paper we use the word *journal* to refer to a source where an article can be published. Apart from journals, the usage of this word also implies magazines, conference proceedings, digital libraries, etc.

number. Additionally, in [128] two new metrics, the *contemporary h-index* and the *trend h-index* are introduced. The first takes into consideration the time that elapsed since an article was published, whereas the second takes into account the date an article received each of its citations.

Apart from the work that has been conducted towards ranking scientists, there is also a considerable research made for evaluating the prestige of a journal. Although the first relative article was published in early seventies [56], it was not before 2002 that this issue gained a remarkable attention. Bharati in [24], studied the preferences of journals for e-commerce research, whereas [75] employs citation analysis to assess journal quality and ranking. On the other hand, [88] and [127] apply scientometrics to determine the prestige of several information systems journals and scientific conferences respectively. In [112] there is a study which examines the differences across journal rankings, whereas in [29] and [125] several Hirsch-type indices for evaluating journals are proposed.

5.4.2 Problem formulation

Let us begin by introducing $P = \{p_1, p_2, ..., p_{|P|}\}$ which is the set containing all publications (also mentioned as papers, or articles) and $B = \{b_1, b_2, ..., b_{|B|}\}$ that is another set including the journals where the items of P have been published. Note that since each paper is published in exactly one journal, each entry $p_i \in P$ is mapped to a single element $b_l \in B$. Moreover, we define $A = \{a_1, a_2, ..., a_{|A|}\}$ as the set including all the authors (also mentioned as scholars, or scientists) who have contributed to the creation of the items of P and $F = \{f_1, f_2, ..., f_{|F|}\}$ which includes all the research fields involved in our problem.

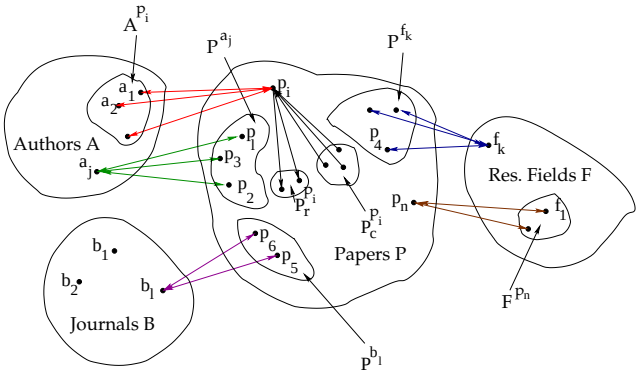


Figure 5.4: Graphical representation of the examined universe

Based on the previous analysis we identify the subset $A^{p_i} \subset A$ which contains the researchers who have authored an article p_i, whereas the topic discussed in p_i is categorized to one or more research fields belonging to the subset $F^{p_i} \subset F$. Equivalently, each author

Symbol	Meaning
P	The set containing all papers
A	The set containing all authors
F	The set containing all research areas
B	The set containing all journals
p_i	An arbitrary paper $p_i \in P$
Y_i	The year of publication of p_i
ΔY_i	The age of publication of p_i
a_j	An arbitrary author $a_j \in A$
f_n	An arbitrary research area $f_n \in F$
b_l	An arbitrary journal $b_l \in B$
A^{p_i}	The authors who created p_i
F^{p_i}	The research areas that p_i belongs to
P^{f_n}	The papers belonging to f_n
P^{a_j}	The papers authored by a_j
P^{a_j,f_n}	The papers authored by a_j and belong to f_n
P^{b_l}	The papers published in b_l
P^{b_l,f_n}	The papers published in b_l and belong to f_n
$P_r^{p_i}$	The papers referenced by p_i
$P_r^{p_i,f_n}$	The papers referenced by p_i and belong to f_n
$P_c^{p_i}$	The papers referring to p_i
$P_c^{p_i,f_n}$	The papers referring to p_i and belong to f_n
$h_\nu^{a_j}$	A metric evaluating the work of an author a_j
$h_\mu^{b_l}$	A metric evaluating the prestige of b_l

Table 5.9: Summary

a_j has published a series of papers $P^{a_j} \subset P$ and each research field f_n contains a subset of papers $P^{f_n} \subset P$.

Apart from these basic sets we also introduce the subset $P_r^{p_i} \subset P$ which contains all papers referenced by p_i, and $P_r^{p_i,f_n} \subset P_r^{p_i}$ which stores the publications referenced by p_i and also, they are classified into the research area f_n. In a similar spirit, $P_c^{p_i} \subset P$ and $P_c^{p_i,f_n} \subset P_c^{p_i}$ include the articles referring to p_i and the articles which both cite p_i and belong to the research field f_n. All introduced sets and subsets along with their connections are illustrated in Figure 5.4.

Finally, we use the symbol Y_i to indicate the year that the paper p_i was published in a journal b_l. Furthermore, $\Delta Y_i = Y_{now} - Y_i + 1$ is used to represent the years elapsed since the journal was published, where Y_{now} is the current year.

The quantity, the quality, the number of incoming references and some other characteristics of the publications of a researcher have been used widely to determine his/her productivity and impact. Several existing works (see section 5.4.1) state that the activity

of a researcher a_j can be evaluated by using a single value $h_\nu^{a_j}$ and they propose effective approaches towards this direction. Moreover, the characteristics of the papers published by a journal and the reputation of the involved authors can be exploited for evaluating this journal by using another metric, $h_\mu^{b_l}$. Note that the symbols ν and μ are identifiers used to differentiate the approaches that exist for evaluating a researcher's work and a journal's prestige respectively.

In Table 5.9 we summarize all the above notifications and in Figure 5.4 we illustrate the examined universe and the connections among the distinct sets of our analysis.

5.4.3 Problem statement

The discussion of the previous subsection determined the boundaries of the space where our problem lies. Our goal now is to identify the research areas F which are attractive for an author a^j, for whom the metric $h_\nu^{a_j}$ receives low values. For this purpose, for each field of research we introduce a special score S^{f_n}, which is calculated by taking into consideration the characteristics of a new scientist and an attractive research field. After that, we only have to sort the research fields by decreasing S^{f_n} order to obtain the desired knowledge.

As we will see later, the main problem includes three component issues which are essential to be addressed before we proceed in the extraction of the desired information. These are the evaluation of a researcher's work, the evaluation of a journal's reputation and the classification of an article within a given taxonomy of research areas. The first two sub-problems are related to finding effective methods for computing the $h_\nu^{a_j}$ and $h_\mu^{b_l}$ metrics and the literature contains numerous satisfactory solutions for this purpose.

Regarding the identification of the research field that an article belongs to, an algorithm for mapping each of the items of the set P to one or more entries of the set F is required. In this work we utilize a machine-learning algorithm which we have developed for this purpose. More details about this algorithm are provided in section 5.5 of this chapter.

5.4.4 Identifying attractive research areas

The problem we discuss here concerns new scientists, that is, scientists with low $h_\nu^{a_j}$ values. To determine an effective solution, it is necessary that we take into consideration an accurate overview of their characteristics. Some of the most important properties of the individuals belonging to this category are the *lack of experience* and the *lack of trust*. The former, lack of experience, is connected to the fact that a new researcher is not always able to discover or even understand the open problems in some challenging research areas. Moreover, even if a problem is formulated, the scholar is not usually in the position to

propose a solution that is more effective than the ones that have already been proposed by other researchers. The latter, lack of trust, means that a new researcher is not reputable and it is expected that his projects will be treated with caution by the rest of the members of the scientific community.

Concerning the research fields, we determine two significant properties: *popularity* and *attractiveness for new scientists*. The former is mainly connected to the number of published articles and the number of scientists dealing with this particular research field. Regarding the latter, our research has shown that not all popular research topics are suitable for them and that additional properties must be considered. We shall discuss these properties shortly, since one of the primary goals of this work is to provide evidence supporting this claim.

To quantify the aforementioned properties and construct a model for evaluating each scientific field, we performed an enquiry among our colleagues. In particular, we have prepared a Web interface and we have asked from other PhD candidates to determine the reasons which render an area of research attractive, and the motivations that led them choose the subjects of their dissertations. The enquiry was answered by 141 new scientists from multiple departments of several universities and its conclusions proved that the most significant attributes that render a research field attractive for a new researcher are:

- *Number of recent articles:* Among all the enquiry answerers, a remarkable percentage of 62% agrees that the number of articles dealing with multiple problems from the same research area is a strong indication about the area's attractiveness and popularity. However, this parameter alone is not sufficient; the articles should also be recent, unless we desire to identify obsolete research fields which were once trendy. Recency is related to the time that has elapsed since a given date. In this work we assume that a paper is recent if it was published up to Y years before the current date and in section 5.4.5 we are conducting experiments by examining different values of recency (i.e. we set $Y = 1, 2$ or 3 years).

- *Impact of articles:* To characterize a research area as attractive for a new scholar the number of recent publications is not adequate; the matter of the impact of these papers is equally important. The impact an article has in the scientific community can be evaluated by applying citation analysis methods which are based on the information provided by the inter linkage of the research papers. Such information includes the number of citations each paper acquired, their age, the publishing journal etc. Furthermore, the number of recent citations received by an entire research field, partially reveals its current popularity. This parameter was verified by the 68% of our

enquiry answerers.

- *Reputation of the publishing journals:* Publication in prestigious journals has significant influence on promotion decisions, tenure and peer recognition. When an article is published in a reputable journal, it is expected that it will gain the attention of a large number of other scientists. Indeed, our enquiry confirmed that a percentage of 64% of new scientists will probably make an effort to propose a more effective methodology to confront the problem that the paper in question studies. In other words, other scientists are being attracted by the content of the papers which are published in high-level journals, since a more efficient approach to the same problem may result in a publication by a journal of equal or higher reputation. Furthermore, it is a common strategy for many new scientists to watch and study the articles published in the most important journals in order to determine the object of their future research. Consequently, the more articles from the same research areas are published in reputable journals, the more attractive this research area is for new authors.

- *Influence of the contributing authors:* In our effort to identify the attractive research areas for new scientists, we also examine the reputation of the authors who have published the most recent and influential works. When a high-level scientist deals with a problem and proposes an effective solution, it is expected that his/her work will be published in a top-quality journal. This is due to his/her high level of expertise and the trust he/she enjoys by the other members of the scientific community. Nonetheless, this does not make the research area the paper belongs to attractive for a starting scientist. Instead, we believe that this matter is detrimental to an author of low reputation who is usually not able to propose a more effective solution. 42% of our enquiry participants stated that they examine the previous experience and a paper author and they are influenced by the qualitative publications of other new scientists.

Based on the aforementioned enquiry and the parameters we discussed above, we conclude that popularity is not the only parameter affecting the new scholars during the selection of their area of research. Other characteristics such as the *impact* of the published articles, the *reputation* of the publishing journals and the *popularity among the other new scholars* must be considered when searching for attractive fields of research for new scientists.

Now we summarize the above notifications by characterizing a field of research as popular for a specific year Y, if its corresponding publications are:

$$[Multitudinous]\ AND\ [Influential]$$
$$AND\ [Authored\ by\ multiple\ distinct\ scientists] \tag{5.2}$$

Consequently, the more publications a research field has, the more popular it is. Additional indications of popularity are the number of incoming citations and the number of the distinct authors dealing with the problems of the research field in question. Based on these properties we introduce the following scoring formula which determines the popularity of a research field:

$$S_{1,Y}^{f_n} = |P_Y^{f_n}| + \sum_{i=1}^{|P^{f_n}|} |P_{c,Y}^{p_i}| + \sum_{i=1}^{|P_Y^{f_n}|} |A^{p_i}| \tag{5.3}$$

The criteria which render a research area attractive for new scientists are different. According to our discussion, a topic is suitable for new scholars if the papers which are relevant to it are:

$$[Multitudinous]\ AND\ [Recent]\ AND\ [Influential]$$
$$AND\ [Published\ in\ reputable\ journals]$$
$$AND\ [Authored\ by\ new\ scientists] \tag{5.4}$$

Now the parameters of 5.4 provide a qualitative solution to the problem of identifying attractive research areas for new scientists. In order to quantify our solution we must determine numerically the attractiveness of each scientific area and the following equation fulfils our goal:

$$S_{2,\nu,\mu}^{f_n} = \sum_{i=1}^{|P^{f_n}|} \frac{|P_c^{p_i}| h_\mu^{b_l}}{(\Delta Y_i)^\delta} \Big(\sum_{j=1}^{|A^{p_i}|} \frac{\lambda}{h_\nu^{a_j}} \Big) \tag{5.5}$$

where λ is a constant quantity used to assign the second sum a meaningfully large value, and δ is a parameter which determines the rate at which a publication becomes "old". A typical value for this parameter is $\delta = 1$.

To compute the $S_{2,\nu,\mu}^{f_n}$ scores we must initially map each article to the corresponding research field. It is also required to calculate the values of the $h_\nu^{a_j}$ and $h_\mu^{b_l}$ metrics, which indicate the reputation of the scientist who authored each paper and the prestige of the journal which published it, respectively. In the sequel, we iterate over all publications

belonging to the research area f_n and evaluate the desired scores by considering the number of citations each of these publications acquired.

Equation 5.5 can be further enhanced by taking into consideration that an area could be attractive for a starting scholar, if the papers mapped to it receive recent citations. This reveals that the problems described in those works although they are old, still affect the scientific community. The following scoring formula incorporates this intuitive criterion:

$$S_{3,\nu,\mu}^{f_n} = \sum_{i=1}^{|P^{f_n}|} \frac{h_\mu^{b_l}}{(\Delta Y_i)^\delta} \Big(\sum_{x=1}^{|P_c^{p_i}|} \frac{1}{(\Delta Y_x)^\delta} \sum_{j=1}^{|A^{p_i}|} \frac{\lambda}{h_\nu^{a_j}} \Big) \tag{5.6}$$

Notice that the usage of the time interval in the denominator of the first sum of 5.5 and 5.6 denotes that we are mainly interested for research areas which attracted multiple publications recently. In addition, the placement of the $h_\nu^{a_j}$ metric in the denominator of the second sum reveals our goal to reward the publications authored by new scientists. Finally, the selection of placing $h_\mu^{b_l}$ in the numerator is justified by our intention to highlight the articles that have been published in prestigious journals.

5.4.4.1 Topic-Sensitive extensions

Often, many scientists contribute knowledge to more than one scientific fields and publish projects in multiple adjacent areas of research. Therefore, it is possible for a scientist to be distinguished in some research fields, whereas in others, the impact of his/her works to be limited. For instance, a scholar may have authored broadly acceptable articles regarding "Fiber optics", but his/her publications that are relevant to "Performance Analysis" not to be equally influential.

The existing metrics are not sensitive to this concept; they take into account all the publications of an author and provide a single value indicating the productivity and/or impact. For this reason, we introduce here a set of *Topic-Sensitive (TS)* extensions, which can be applied to all three previous approaches. The idea is to divide the works of a scientist according to the research field they belong to and then compute multiple metric values, one for each research field. The *Topic Sensitive h-index (TSh-index)* incorporates this idea:

Definition. A researcher a_j has TSh-index $h_{1,f_n}^{a_j}$ for the research field f_n, if $h_{1,f_n}^{a_j}$ of his/her $|P^{a_j}|$ articles that discuss a topic belonging to f_n have received at least $h_{1,f_n}^{a_j}$ citations each and the rest $(|P^{a_j}| - h_{1,f_n}^{a_j})$ articles have received no more than $h_{1,f_n}^{a_j}$ citations.

This metric calculates how broad the research work of a scientist is for a specific research area and identifies the scientists who are experts and reputable in a particular field of expertise.

Now let us examine how the time-variants of the h-index can be extended by applying the topic sensitivity approach. Regarding the contemporary h-index, we convert the scores presented in [125] to the ones of equation 5.7:

$$S_c^{p_i,f_n} = \gamma \frac{|P_c^{p_i,f_n}|}{(\Delta Y_i)^\delta} \tag{5.7}$$

That is, instead of evaluating all the articles of an author, we take into consideration only the papers belonging to the area of research for which we desire to rank a scientist. These scores $S_c^{p_i,f_n}$ are used to phrase the definition of the *contemporary TSh-index*:

Definition. A researcher a_j has contemporary TSh-index $h_{2,f_n}^{a_j}$ for the research field f_n, if $h_{2,f_n}^{a_j}$ of his/her $|P^{a_j}|$ articles that discuss a topic belonging to f_n, get a score of $S_c^{p_i,f_n} \geq h_{2,f_n}^{a_j}$ and the rest $(|P^{a_j}| - h_{2,f_n}^{a_j})$ articles get a score of $S_c^{p_i,f_n} < h_{2,f_n}^{a_j}$.

Similarly to the original contemporary h-index, this metric rewards the scholars who are currently active, or the new scientists who have currently published only a small number of influential works. The difference is that this procedure is performed on a per-topic level and one scientist can be assigned different rankings according to the research of area that we examine each time.

The trend h-index can be also extended by adopting an identical approach. Therefore, the original scores of [125] are modified according to the equation 5.8:

$$S_t^{p_i,f_n} = \gamma \sum_{n=1}^{|P_c^{p_i,f_n}|} \frac{1}{(\Delta Y_n)^\delta} \tag{5.8}$$

Based on these modified scores $S_t^{p_i,f_n}$, the definition of the *trend TSh-index* follows:

Definition. A researcher a_j has trend TSh-index $h_{3,f_n}^{a_j}$ for the research field f_n, if $h_{3,f_n}^{a_j}$ of his/her $|P^{a_j}|$ articles that discuss a topic belonging to f_n, get a score of $S_t^{p_i,f_n} \geq h_{3,f_n}^{a_j}$ and the rest $(|P^{a_j}| - h_{3,f_n}^{a_j})$ articles get a score of $S_t^{p_i,f_n} < h_{3,f_n}^{a_j}$.

ν	**Symbol**	**Meaning**
1	$h_1^{a_j}$	h-index
2	$h_2^{a_j}$	contemporary h-index
3	$h_3^{a_j}$	trend h-index
$1, f_n$	$h_{1,f_n}^{a_j}$	Topic-Sensitive h-index
$2, f_n$	$h_{2,f_n}^{a_j}$	contemporary TSh-index
$3, f_n$	$h_{3,f_n}^{a_j}$	trend TSh-index

Table 5.10: Summary of metrics for evaluating the work of a scientist

In contrast to the contemporary TSh-index which is sensitive to the age of each publication, this metric takes into consideration the year that each article received its citations. We

anticipate that this approach will rank higher the authors whose work in a specific scientific field is considered pioneering (since it still attracts references) and could set a new line of research.

In Table 5.10 we summarize all the metrics that we have previously discussed, including the Topic-Sensitive extensions. The left column denotes the value that ν receives for each case; the middle column contains the corresponding symbol for each metric, whereas in the last column we record its respective name.

5.4.5 Experiments

To conduct a thorough experimental analysis of the proposed methods, it is required that we construct or select an existing taxonomy of research fields. Furthermore, it is essential that we obtain a dataset of research articles which must be large enough to provide reliable results. For each paper of our dataset we need to acquire all the accompanying metadata including the authors, the year of publication, its keywords, the publishing journal, its references and if supported, its classification into one or more research fields of our employed taxonomy.

Apparently, a percentage of the articles of the dataset must support the given taxonomy. This is necessary in order to train the model of our classification algorithm. To the best of our knowledge, there are not any publicly available datasets satisfying all the aforementioned requirements. The strict policy applied by the digital libraries in order to protect their records, prevents us from accessing their databases. Nonetheless, *CiteSeerX*[7], a scientific digital library and search engine, allows its users to access its records[8] and provides a harvest mechanism[9] for retrieving the entire database and the full text of the articles. The majority of these papers are related to the wide fields of Engineering, Mathematics and Computer Science. From these papers we have removed some duplicate articles and some which were not accompanied by the desired meta-data (i.e. authors, journal or date of publication). At the end of this filtration process, our dataset was comprised of 1,429,398 distinct articles.

After the elimination of the problematic articles (i.e. duplicate entries and entries missing the required meta-data), we applied our machine learning classification algorithm introduced in 5.5. According to this method, the category of each paper depends on the category of its neighboring (i.e. citing) articles. Moreover, before applying the algorithm, it is required that we determine the set of categories (the taxonomy) where the items of our

[7] http://citeseerx.ist.psu.edu/

[8] http://citeseerx.ist.psu.edu/about/metadata

[9] http://citeseerx.ist.psu.edu/oai2

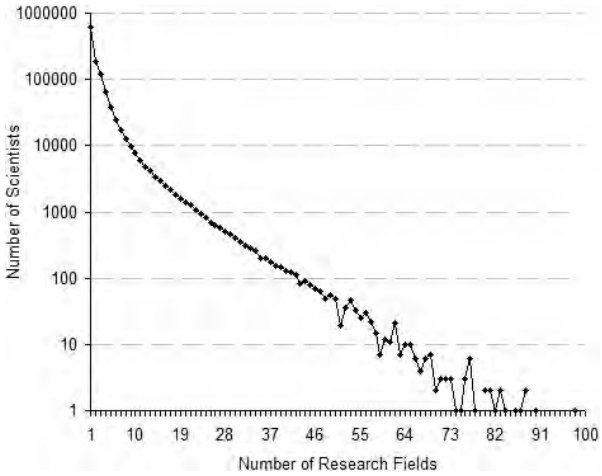


Figure 5.5: Number of Authors vs Number of Research Areas

collection will be classified.

Regarding the taxonomy structure, we considered a number of existing propositions. For instance, *Google Scholar*[10], is a vertical search engine designed to facilitate searching for articles and authors. It employs a classification model that categorizes the articles into nine generic research fields. However, the search engine classifies articles belonging to different research areas to the same category (i.e. papers regarding Mathematics and Computer Science are all classified into the same category). Apart from this notification, we firmly believe that these nine categories are not adequate to provide satisfactory information. We need a more precise mechanism that divides the main research fields into multiple levels of smaller research fields.

IEEE and *ACM* utilize a common taxonomy structure to categorize the articles they publish. That structure if far more informative than that of *Google Scholar's*, since it divides the generic term "Computer Science" into a large number of levels and sub-levels of research fields and furthermore, the classification is hierarchical. It consists of 11 first-level research fields divided into 81 second-level and 276 third-level classes.

In our experiments we focus on research areas and articles which are related to the Computer Science and we employ the aforementioned taxonomy structure. However, the ideas and the concepts we describe here can also be used with other taxonomies with no additional effort.

[10]http://scholar.google.com

Author	$\|P^{a_j}\|$	$h_1^{a_j}$
H. Garcia-Molina	328	46
J. Ullman	195	40
S. Shenker	170	40
P. Hanrahan	113	36
D. Estrin	142	36
C. Faloutsos	246	35
D. E. Culler	116	35
D. J. DeWitt	163	34
J. Widom	130	34
J. Han	290	34
C. Papadimitriou	253	34
W. B. Croft	201	34
R. Agrawal	147	34
T. Anderson	138	34
R. Fagin	114	34

Author	$h_2^{a_j}$
H. Garcia-Molina	51
Philip S. Yu	35
B. Forouzan	33
D. E. Culler	31
P. Hanrahan	31
S. Shenker	30
D. Estrin	30
T. Anderson	29
R. Motwani	29
M. Abadi	29
R. Kumar	28
M. D. Hill	27
W. B. Croft	27
J. Ullman	27
P. A. Bernstein	26

Author	$h_3^{a_j}$
S. Shenker	53
H. Garcia-Molina	51
A. K. Jain	47
J. Han	46
J. Widom	44
D. J. DeWitt	42
M. Stonebraker	42
M. D. Hill	42
J. Ullman	41
B. Shneiderman	41
R. Motwani	41
C. Faloutsos	39
P. Hanrahan	39
D. Estrin	39
T. Anderson	38

Table 5.11: Authors rankings (all research areas) according to h-index (left), contemporary h-index (center), trend h-index (right).

5.4.5.1 Identifying Reputable Scientists

In this subsection we apply the current state-of-the-art approaches for ranking scientists, as well as our proposed Topic-Sensitive extensions. Notice that all the metric values we present in this work have been calculated by using our test dataset; for other collections of papers these values can vary significantly. The articles of our dataset were authored by 1,209,316 scholars, a value which is translated to about 1.18 articles per author. However, the vast majority of them (about 70%) has published only once.

Figure 5.5 illustrates the distribution of authors with respect to the number of research areas their papers belong to. The vertical axis of this graph is in logarithmic scale. From this representation we conclude that a significant percentage of 54.4% of the authors have dealt with only one field. Only 18.9% of the authors of our dataset have published articles in more than three areas of research.

Table 5.11 contains rankings of the top-15 scientists of our dataset, according to three popular scientometrics. The h-index metric has been used for the left ranking, contemporary h-index determines the middle ranking, whereas trend h-index determines the right ranking. The third column of these rankings represents the total number of publications of a particular author, whereas the last column indicates the value the metric receives.

The scientist with the widest impact according to h-index is *H. Garcia-Molina* with 328 publications and $h_1^{a_j} = 46$, followed by *J. Ullman* (195 papers and $h_1^{a_j} = 40$) and *S. Shenker* (170 papers and $h_1^{a_j}$=40). Regarding the ranking according to the contemporary h-index, *H. Garcia-Molina* is again the top-scientist since his works not only are numerous and receive many citations, but also are recent. Recall that this metric is sensitive to the age of each publication and the score each article receives decays as time elapses. However,

Language Classifications

Author	$\lvert P_{f_n}^{a_j} \rvert$	$h_{1,f_n}^{a_j}$
C. Chambers	31	20
G. L. Steele, Jr	60	19
S. P. Jones	73	16
P. Wadler	35	16
M. Felleisen	43	15
K. Kennedy	39	14
N. Wirth	39	14
D. Ungar	35	14
B. Liskov	35	13
M. Wand	25	12

Network Architecture and Design

Author	$\lvert P_{f_n}^{a_j} \rvert$	$h_{1,f_n}^{a_j}$
D. Estrin	86	27
H.Balakrishnan	60	24
S. Shenker	67	22
D. E. Culler	42	20
N. H. Vaidya	106	20
Lixia Zhang	63	19
F. Floyd	30	19
I. F. Akyildiz	83	17
R. Morris	32	17
J. A. Stankovic	69	16

Information Search and Retrieval

Author	$\lvert P_{f_n}^{a_j} \rvert$	$h_{1,f_n}^{a_j}$
W. B. Croft	153	33
Cheng Xiang Zhai	83	19
G. Salton	123	18
C. Buckley	57	18
J. Callan	70	18
Wei-Ying Ma	112	18
H. Garcia-Molina	61	17
S. T. Dumais	61	17
S. E. Robertson	58	16
S. Lawrence	27	16

Database Applications

Author	$\lvert P_{f_n}^{a_j} \rvert$	$h_{1,f_n}^{a_j}$
Jiawei Han	192	34
Philip Yu	138	19
Jian Pei	96	18
R. Agrawal	32	17
M. J. Zaki	90	17
H.-P. Kriegel	96	16
E. Keogh	53	16
R. Srikant	22	14
R. T. Ng	44	14
Ke Wang	53	13

Table 5.12: Authors ranking according to TSh-index for various research areas.

J. Ullman, the second most reputable scientist according to h-index, is ranked in the 14^{th} position and *S. Shenker* is ranked sixth. The second best performing scientist according to $h_2^{a_j}$ is *Philip S. Yu*, who does not appear in the top-15 h-index based ranking.

In contrast to the contemporary h-index, Trend h-index $h_3^{a_j}$ is sensitive to the age of each citation. The top-level scientist according to it is *S. Shenker* who is apparently the author whose works are still being referenced by the recent publications. *H. Garcia-Molina* is ranked second in this occasion, whereas *J. Ullman* is located in the ninth position of the Table.

Now let us study the rankings constructed by our proposed Topic-Sensitive extensions. Recall that these metrics are not applied in the entire set of an author's publications, but it is required that we isolate the papers which are mapped to a specific field of research. In Table 5.12 we present the ten most highly-ranked scholars according to TSh-index, for four different research areas: *Language Classification*, *Network Architecture and Design*, *Information Search and Retrieval* and *Database Applications*. The second column of each ranking denotes the number of publications which are both authored by a specific scientist and are mapped to the examined research field. The third column records the value that the applied metric receives.

We shall discuss the *Information Search and Retrieval* research field, however, the conclusions we extract from this discussion can be generalized and are valid for the other fields too. The author who is ranked first in that particular field is *W. B. Croft* who has authored

Language Classifications

Author	$\lvert P_{f_n}^{a_j} \rvert$	$h_{3,f_n}^{a_j}$
C. Chambers	31	17
P. Wadler	35	14
G. L. Steele, Jr	60	13
M. Felleisen	43	13
S. P. Jones	73	12
Krishnamurthi	26	12
D. Ungar	35	11
D. Grove	22	11
B. G. Ryder	32	11
D. F. Bacon	25	11

Network Architecture and Design

Author	$\lvert P_{f_n}^{a_j} \rvert$	$h_{3,f_n}^{a_j}$
D. Estrin	86	30
H. Balakrishnan	60	24
D. E. Culler	42	23
N. H. Vaidya	106	22
S. Shenker	67	21
Lixia Zhang	63	20
R. Morris	32	20
I. F. Akyildiz	83	18
M. Srivastava	64	18
R. Govindan	48	18

Information Search and Retrieval

Author	$\lvert P_{f_n}^{a_j} \rvert$	$h_{3,f_n}^{a_j}$
W. B. Croft	153	28
Wei-Ying Ma	112	23
Cheng Xiang Zhai	83	21
S. T. Dumais	61	20
J. Callan	70	19
S. E. Robertson	58	18
H. Garcia-Molina	61	17
C. Buckley	57	17
A. Spink	76	16
Jiawei Han	44	16

Database Applications

Author	$\lvert P_{f_n}^{a_j} \rvert$	$h_{3,f_n}^{a_j}$
Jiawei Han	192	34
Philip Yu	138	22
Jian Pei	96	21
M. J. Zaki	90	19
R. Agrawal	32	17
C. Faloutsos	76	17
G. Karypis	32	16
E. Keogh	53	16
H.-P. Kriegel	96	15
Ke Wang	53	14

Table 5.13: Authors ranking according to Trend TSh-index for various research areas.

153 relevant articles and has $h_{1,f_n}^{a_j} = 33$. Notice that this author is ranked 12^{th} according to the plain h-index metric, and has authored in total 201 works. Nonetheless, when TSh-index is applied, only 153 of these works are considered. A similar notification can also be made for *H. Garcia-Molina* who has authored in total 328 articles, but only 61 of them are related to the field of *Information Search and Retrieval*.

Table 5.13 contains author rankings for the aforementioned areas of research according to the Trend TSh-index. This metric rewards scholars for a particular research field, if their works continue to be cited until presently. *W.B. Croft* is still on the top of the list for the *Information Search and Retrieval* research field, However, *Wei-Ying Ma* has climbed in the second place (he was sixth according to TSh-index), whereas *G. Salton* is no longer among the top-10 authors. This observation leads to the conclusion that the works of the latter author do not receive many recent citations; potentially the problems discussed in those works have been addressed, or the topics are outdated.

5.4.5.2 Identifying Prestigious Journals

We continue our processing by attempting to detect the prestigious journals, since this information is valuable for identifying the attractive research fields. Recall that if a large number of articles associated with a particular scientific area is published in reputable journals, then this area becomes attractive for other scholars.

Due to limited space we focus primarily on the h-index for journals and the impact

Journal Name	$h_1^{b_l}$	$\lvert P_c^{b_l} \rvert$
Communications of the ACM	10741	122
International Conference on Computer Graphics and Interactive Technology	8812	111
International Conference on Management of Data	2632	92
IEEE Trans. on Pat. Analysis and Machine Intelligence	3619	90
Journal of the ACM	2752	85
Applications, Technologies, Architectures, and Protocols for Computer Communication	1377	76
Conference on Human Factors in Computing Systems	7557	76
Artificial Intelligence	1987	73
ACM Computing Surveys	1300	72
IEEE Transactions on Software Engineering	3043	71
Very Large Data Bases	2406	70
International Symposium on Computer Architecture	1491	69
ACM Conference on Research and Development in Information Retrieval	2252	68
Symposium on Principles of Programming Languages	1188	67
Conference on Programming Language Design and Implementation	772	66

Table 5.14: Journals Ranking according to h-index

factor. In Table 5.14 we present the ranking of the journals we encountered in our dataset according to this metric. As previously, the rankings presented here should not be treated as representations of the value of a journal; it is possible that multiple papers from a journal are missing and the same could also be valid for their citations.

Table 5.15 illustrates the ranking of the journals for 2009 according to the impact factor. Notice that the only journal which is common in these two rankings is *Applications, Technologies, Architectures, and Protocols for Computer Communication*. This is an indication that a per-year journal evaluation leads to significantly different results than an all-year evaluation process.

Journal Name	$h_{6,2009}^{b_l}$	$\lvert P_c^{b_l} \rvert$
ACM Symposium on Operating Systems Principles	7.29	175
Web Search and Web Data Mining	5.27	137
ACM Computing Surveys (CSUR)	4.71	146
International Symposium on Computer Architecture	4.46	370
Proceedings of the 6th USENIX Conference on File and Storage Technologies	3.90	82
Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation	3.80	114
Applications, Technologies, Architectures, and Protocols for Computer Communication	3.67	588
Computational Linguistics	3.41	218
Internet Measurement Conference	3.38	243
Conference on Programming Language Design and Implementation	3.33	276

Table 5.15: Journals Ranking for 2009 according to impact factor

5.4.5.3 Popular Research Areas

In this subsection we are based on our dataset to present the research areas which are the most popular. Recall that a research field is considered as popular in case many relevant

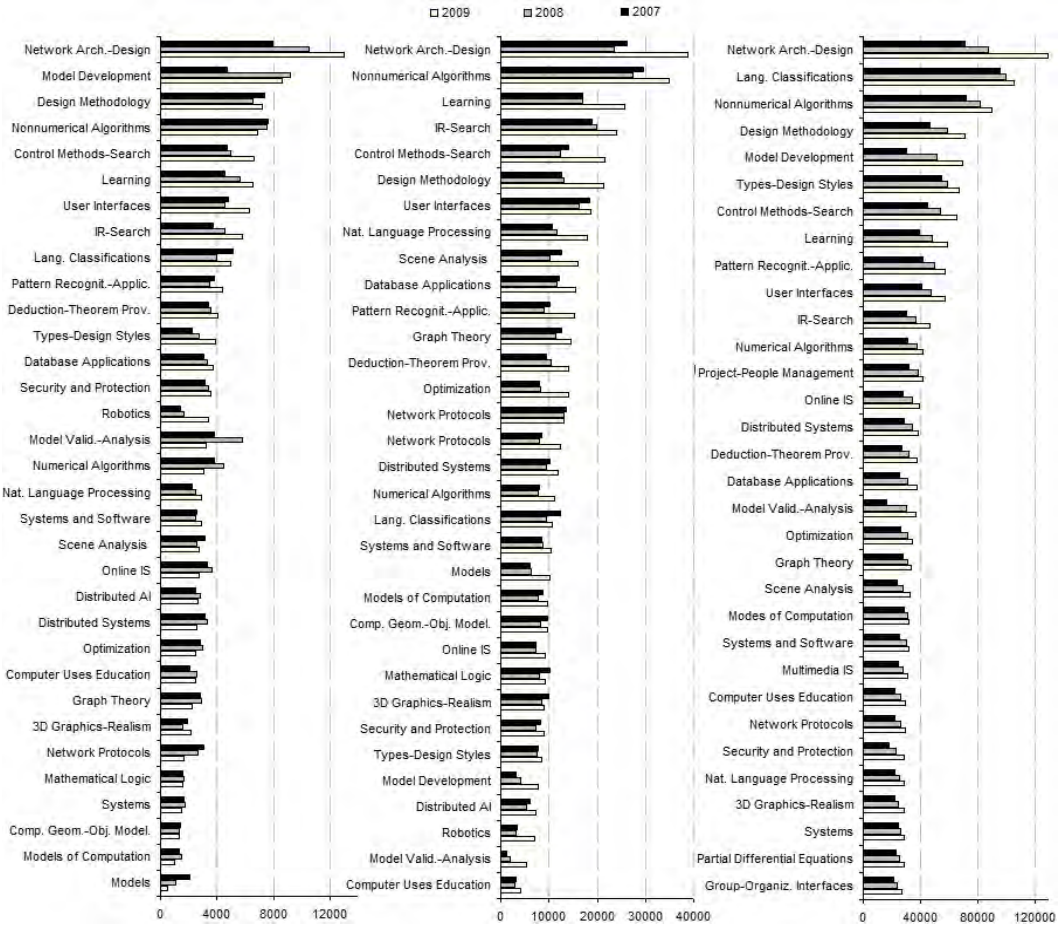


Figure 5.6: Popular Research Fields in the last 3 years by number of published papers (left) number of incoming citations (center), and number of distinct authors (right)

articles are published and these articles have significant impact on the scientific community. Finally, the number of authors dealing with its problems is another indication of popularity.

Figure 5.6 illustrates the 35 most popular research fields in the last three years. The left part of the diagram depicts the number of relevant articles for each area, the middle part determines their popularity according to the number of incoming citations, whereas the right part reveals the number of distinct authors addressing problems which are relevant to the respective area.

Let us study the data displayed in these diagrams. The area which attracted the most publications in all three years is *Network Architecture and Design*; 12,992 articles of 2009 were mapped to this category. The second most popular area for 2008 and 2009 is *Model Development*. However, the second most popular field of research in 2007 was *Design Methodology*.

Regarding the number of incoming references, *Network Architecture and Design* is

again the most popular field for 2009. Nevertheless, the area of *Non-numerical algorithms and problems* occupied the first position in 2007 and 2008. Other top-ranked research fields according to the number of in-links is *Learning* and *Information Search and Retrieval*. Although these fields had fewer papers than *Model Development* and *Design Methodology*, these papers attracted much more citations. This indicates that these papers affected more scientists.

The third part which determines the popularity of a research field according to the equation 5.3 is the number of authors publishing articles that are relevant to this particular field. The right diagram of Figure 5.6 indicates that *Network Architecture and Design* was the most popular area for 2009. However, in the previous two years the field of research of *Language Classifications* was attracting more scientists. *Non-numerical algorithms and problems*, *Model Development*, and *Design Methodology* are the next three highest ranked scientific topics.

In Figure 5.7 we illustrate the value of the $S_{1,Y}^{f_n}$ score for the 20 most popular research areas of 2007, 2008, and 2009. *Network Architecture and Design* has been the most popular topic of research during 2008 and 2009. On the other hand, *Non-numerical algorithms and Problems* and *Language Classifications* were the most widespread scientific areas of 2007. This notification leads to the conclusion that in the past two years, there has been a significant increase in the research conducted towards *Network Architecture and Design*; this increase has rendered this area as the most popular in 2008 and 2009. The top-5 popularity ranking of Figure 5.7 also includes *Design Methodology* and *Control Methods and Search*.

Finally, the reader should notice that although *Learning* and *Information Search and Retrieval* are the third and fourth most cited research areas (middle diagram of Figure 5.6), they are not among the most popular. This is a strong indication that popularity is a generic metric which keeps plenty of useful information hidden.

5.4.5.4 Attractive Research Areas for New Scientists

In this subsection we present the research areas which according to the discussion of subsection 5.4.4 are the most attractive for new scientists. In the following discussion we attempt to experimentally verify whether the popular research areas are all suitable for new scientists. In addition, we shall try to identify other topics which although they are not so popular as others, they could prove themselves promising for this class of scientists.

Recall that the scores of the equations 5.5 and 5.6 depend on both $h_{\nu}^{a_j}$ and $h_{\mu}^{b_l}$ metrics which evaluate the work of an author a_j and the prestige of a journal b_l respectively. However, since the number of possible combinations of these two metrics is quite large, we only

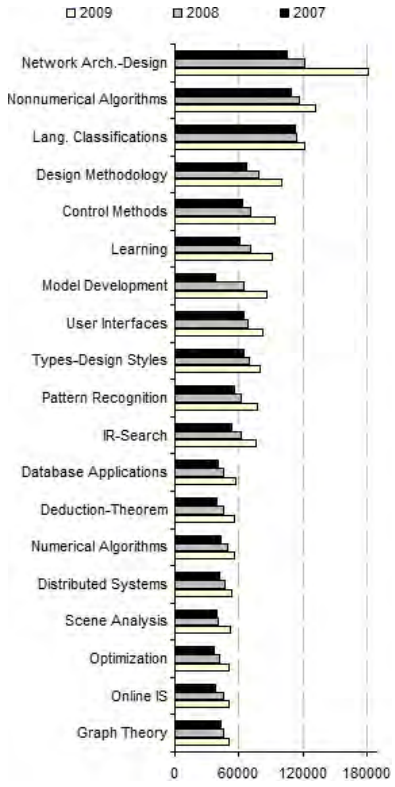


Figure 5.7: The 20 most popular research fields according to $S^{f_n}_{1,Y}$ in the 3 last years

provide results for some representative cases.

Initially we attempt to identify the research fields which are attractive for new scientists according to $S^{f_n}_{2,\nu,\mu}$. In Tables 5.16 and 5.17 we record four different such rankings for various combinations of author and journal evaluation metrics. The left ranking of Table 5.16 is produced by using h-index for both authors and journals ($\nu = 1$, $\mu = 1$), whereas the right ranking is constructed by employing the trend h-index for authors and the plain h-index for journals ($\nu = 3$, $\mu = 1$). Regarding the lists of Table 5.17, the left one shows the 15 most attractive research fields in case the Topic-Sensitive h-index is used to evaluate the work of a researcher and plain h-index is used to determine the prestige of a journal ($\nu = 1, f_n$ and $\mu = 1$) whereas the right ranking is generated by selecting the Topic-Sensitive Trend h-index for authors and the plain h-index for journals ($\nu = 3, f_n$ and $\mu = 1$).

According to the left ranking of Table 5.16, the area which is the most attractive for new scientists is *Non-numerical Algorithms and Problems*, followed by *Network Architecture and Design*. Recall from Figure 5.7 that the latter is most popular than the former, however, new scientists will not find it equally attractive. Surprisingly, the third most attractive research field for new scholars is *User Interfaces*, a topic which is ranked eighth in the corresponding popularity list. Another field of research which is attractive for new

Research Field	$S^{f_n}_{2,1,1}$
Non-Num. Algorithms-Problems	75,369
Network Architecture-Design	53,635
User Interfaces	47,393
Information Search-Retrieval	43,032
Design Methodology	42,164
Learning	40,067
Natural Language Processing	37,401
3-D Graphics and Realism	34,837
Systems	33,416
Graph Theory	32,445
Scene Analysis	31,846
Applications	31,832
Prob. Solving-Cont. Methods	31,175
Deduction-Theorem Proving	30,262
Comp. Geometry-Obj. Modeling	29,279

Research Field	$S^{f_n}_{2,3,1}$
Non-Num. Algorithms-Problems	78,626
Network Architecture-Design	52,719
User Interfaces	47,950
Information Search-Retrieval	43,187
Natural Language Processing	40,186
Design Methodology	39,907
Learning	38,695
Systems	37,427
3-D Graphics and Realism	35,907
Graph Theory	32,853
Language Classifications	32,480
Applications	31,178
Prob. Solving-Cont. Methods	30,742
Comp. Geometry-Obj. Modeling	30,491
Scene Analysis	30,475

Table 5.16: Attractive research fields for new scientists according to $S^{f_n}_{2,\nu,\mu}$ scores, for various author and journal evaluation metrics. Left: $\nu = 1$, $\mu = 1$. Right: $\nu = 3$, $\mu = 1$.

Research Field	$S^{f_n}_{2,1,f_n,1}$
Non-Num. Algorithms-Problems	151,373
Network Architecture-Design	89,276
Information Search-Retrieval	89,019
Graph Theory	84,130
Design Methodology	81,356
User Interfaces	79,736
Learning	76,624
Prob. Solving-Cont. Methods	71,169
Systems	64,558
Applications	64,424
Modes of Computation	60,279
Language Classifications	58,116
Systems and Software	58,009
User/Machine Systems	57,943
3-D Graphics and Realism	57,565

Research Field	$S^{f_n}_{2,3,f_n,1}$
Non-Num. Algorithms-Problems	155,311
Network Architecture-Design	87,913
Information Search-Retrieval	86,720
Graph Theory	84,959
User Interfaces	78,688
Design Methodology	77,521
Learning	75,659
Prob. Solving-Cont. Methods	70,929
Systems	70,487
Applications	64,971
Language Classifications	64,833
Modes of Computation	64,618
3-D Graphics and Realism	59,076
User/Machine Systems	58,560
Deduction-Theorem Proving	57,831

Table 5.17: Attractive research fields for new scientists according to $S^{f_n}_{2,\nu,\mu}$ scores, for various author and journal evaluation metrics. Left: $\nu = 1, f_n$, $\mu = 1$. Right: $\nu = 3, f_n$, $\mu = 1$.

scientists but not so popular is *Information Search and Retrieval*.

Additionally, there are several popular research fields which are totally unappropriate for new scientists. The most representative example of such cases is *Languages Classifications*. This topic is the third most popular, however, it is not ranked among the 15 most attractive research fields. Apparently, the problems related to this research area are difficult to confront or even understand and they are not suitable for starters.

The data recorded in this Table leads to two important conclusions: At first, popularity does not coincide with attractiveness for new scientists. There are popular research fields which are not attractive and they can be characterized as "hostile" for starting scientists, such as *Language Classifications*. On the other hand, there are research fields which

Research Field	$S^{f_n}_{3,1,1}$
Non-Num. Algorithms-Problems	88,847
Network Architecture-Design	68,286
Design Methodology	66,094
User Interfaces	58,970
Learning	55,308
Information Search-Retrieval	55,142
Scene Analysis	45,804
Applications	43,277
Deduction-Theorem Proving	42,339
Prob. Solving-Cont. Methods	41,713
Natural Language Processing	41,603
Graph Theory	39,058
Numerical Algorithms-Problems	34,354
3-D Graphics and Realism	34,133
Optimization	31,966

Research Field	$S^{f_n}_{3,3,1}$
Non-Num. Algorithms-Problems	87,722
Network Architecture-Design	64,577
Design Methodology	60,868
User Interfaces	57,081
Information Search-Retrieval	52,865
Learning	51,771
Scene Analysis	42,763
Natural Language Processing	41,847
Applications	40,960
Deduction-Theorem Proving	39,191
Prob. Solving-Cont. Methods	39,036
Graph Theory	37,704
3-D Graphics and Realism	33,769
Numerical Algorithms-Problems	31,812
Systems	31,353

Table 5.18: Attractive research fields for new scientists according to $S^{f_n}_{3,\nu,\mu}$ scores, for various author and journal evaluation metrics. Left: $\nu = 1$, $\mu = 1$. Right: $\nu = 3$, $\mu = 1$.

although unpopular, they provide excellent opportunities at the scientists in question. Examples of such cases are *User Interfaces* and *Information Search and Retrieval*.

The second ranking of Table 5.16 employs the trend h-index for evaluating the work of a researcher. Recall that this metric is sensitive to age of the incoming citations of an article. Compared to the previous case the top-4 entries are left unchanged, however, in the fifth position we encounter another interesting case. *Natural Language Processing* which is not among the twenty most popular research fields, is quite attractive for new scientists.

Regarding the rankings of Table 5.17, the Topic-Sensitive extensions of h-index and trend h-index are employed for authors. In these cases, to compute the value of $S^{f_n}_2$, we need to store for each author and each research area the value the corresponding metric. That is, an author does not perform equally at every scientific topic; this allows us to identify the individuals who are possibly very experienced, but they are considered as starters for a particular research area. The two most attractive research areas for new scientists are the same once again, whereas *Information Search and Retrieval* is found in the third position. The usage of TSh-index in the $S^{f_n}_2$ highlights *Graph Theory* and considers is as the fourth most suitable scientific toping for new scholars.

Tables 5.18 and 5.19 contain rankings of the most attractive fields of research according to the $S^{f_n}_{3,\nu,\mu}$ score. The left list of Table 5.18 is constructed by using the plain h-index metric for both authors and journals. Compared to the left list of Table 5.16 the ordering of the topics is slightly different. Therefore, *Non-numerical Algorithms and Problems* and *Network Architecture and Design* are again the most appropriate research fields for new scientists, however in the third position *User Interfaces* is replaced by *Design Methodology*. The usage of this metric highlights two significant points: the eighth position of *Three*

Research Field	$S^{f_n}_{3,1,f_n,1}$
Non-Num. Algorithms-Problems	174,918
Design Methodology	124,155
Information Search-Retrieval	111,011
Network Architecture-Design	108,845
Learning	103,935
User Interfaces	96,847
Graph Theory	93,240
Prob. Solving-Cont. Methods	87,801
Applications	86,004
Systems and Software	74,004
Scene Analysis	73,129
Deduction-Theorem Proving	72,581
Numerical Algorithms-Problems	70,121
Optimization	69,350
Models	68,961

Research Field	$S^{f_n}_{3,3,f_n,1}$
Non-Num. Algorithms-Problems	174,565
Design Methodology	116,441
Information Search-Retrieval	105,393
Network Architecture-Design	103,921
Learning	100,244
User Interfaces	93,061
Graph Theory	92,073
Prob. Solving-Cont. Methods	85,573
Applications	85,142
Systems and Software	70,879
Deduction-Theorem Proving	70,229
Scene Analysis	69,260
Numerical Algorithms-Problems	67,792
Models	66,540
Optimization	65,760

Table 5.19: Attractive research fields for new scientists according to $S^{f_n}_{3,\nu,\mu}$ scores, for various author and journal evaluation metrics. Left: $\nu = 1, f_n$, $\mu = 1$. Right: $\nu = 3, f_n$, $\mu = 1$.

Dimensional Graphics and Realism and the ninth place of the *Systems* research fields. Both of them are not among the twenty most popular areas, however they can be considered at least promising for new scholars.

Now let us summarize the results we presented in this subsection. In almost every ranking *Non-numerical Algorithms and Problems* and *Network Architecture and Design* are considered as the most attractive research fields for starting researchers. Other topics also include *User Interfaces*, *Information Search and Retrieval* and *Graph Theory*. The comparison of these results to the popularity ranking of Figure 5.7, leads to the conclusion that popularity and attractiveness do not coincide; there are popular research fields which are not suitable for starters (such as *Language Classifications*), whereas some others, not so popular, are ideal for them.

5.5 Research articles classification

The digital libraries and academic search engines have always been a precious tool for the researchers. Their main functionality is focused on providing search capabilities and further information regarding scientific articles, citations, journals and authors. Multiple such services are in operation on the Web; examples include the ACM Digital Library[11], Google Scholar[12], Microsoft Academic[13], and others.

The problem of the automatic classification of research articles is of remarkable impor-

[11] http://dl.acm.org/

[12] http://scholar.google.com/

[13] http://academic.research.microsoft.com/

tance for these services, since it enables increased functionality and improved performance. For instance, a robust classification strategy allows the user to perform searches by focusing on only a specific portion of the indexed documents, thus increasing both effectiveness and efficiency. Additional potential benefits include similar documents recommendations, collaborative filtering, query expansion facilities, expert identification, and so on.

Several methods have been proposed to address the issue of identifying the research field a scientific article belongs to. These methods include keyword extraction algorithms which attempt to identify repeated textual patterns and extract the most representative keywords from the article. In the sequel, they employ traditional classification approaches such as *k-nearest neighborhood (k-NN)* to identify the research field that best describes the content of the article. Another family of methods adopt citation analysis algorithms which study several citation properties, such as the phenomenon of two or more papers being cited together by multiple articles. These methods have two significant drawbacks: initially, it is not always possible to construct a complete graph of interlinking papers because some nodes and edges are simply not available. At second, a reference to an article does not necessarily reveal thematic affinity.

In this section we attempt to address these issues by proposing a new algorithm for classifying research papers. More specifically, we introduce a model which has its origins in the traditional k-NN approach however, it also takes into consideration several aspects regarding the particular problem which we examine. These aspects include the authors history, co-authorship information, selection of keywords, and the previous publications of a journal. Our classifier is experimentally compared against two state-of-the-art generic text classification methods, namely support vector machines and AdaBoost.MH. We show that the inclusion of the aforementioned parameters leads to improved classification performance by roughly 6%.

5.5.1 Related work

Document classification is a well-established data mining problem and the issue of scientific papers classification is a specialization of this problem posing its own challenges.

The methodologies encountered in the literature can be divided into two wide categories: link-based and text-based categorization. The first category includes works which are mainly based on the document linking and the information extracted out them. For instance, in [59] the authors introduce a statistical framework for modeling link distributions and based on that knowledge, they classify a document according to the category its links belong to. Link-based classification is particularly essential for categorizing graph nodes (i.e. labeling the nodes of a graph, [25], or networked data classification[90]). Furthermore,

similar approaches can be also applied on the Web, where the document interlinking can be used for a variety of purposes. An important survey of such methods is provided in [110].

On the other hand, machine learning (ML) text categorization has a gained substantial attention by the data miners; A complete survey on the most effective ML text categorization approaches is provided in [123]. Moreover, [152] and [151] provide detailed evaluations of the primary statistical and machine learning approaches to text categorization. Furthermore, Joachims employed support vector machines (SVM) for the same task [73], whereas [158] introduced AdaBoost.MH, a multi-class expansion of the traditional two-class AdaBoost algorithm.

Nevertheless, none of the aforementioned approaches take into consideration problem-specific information such as the authors of an article, co-authorship and the publishing journal. In this paper we introduce a new method which is based not only on the paper's keywords, but also in the previous activity of both the contributing authors and the publishing journal.

5.5.2 Classification algorithm

Our analysis begins by introducing a number of fundamental sets that will assist us in establishing a baseline for our algorithm. Initially we define P as the set containing all publications, and J as another set including the journals[14] where the items of P have been published. Note that since each paper is published in exactly one journal, each entry $p \in P$ is mapped to a single element $j \in J$. Moreover, we define A as the set including all the involved authors, and C which consists of the research fields (also mentioned as categories, or labels) where the papers of P belong to. In other words, C represents the given taxonomy structure.

A large number of publications contain *keywords* which are used by the authors to explicitly represent the content of their work. The algorithm we present here exploits these keywords and for this reason, we define a set K which contains all the keywords encountered in all papers of P. In the same set K we also include the keywords extracted from the titles of the articles, since these words represent the documents contents as well.

In addition, we introduce the subset $K^p \subset K$ containing all the keywords of a single article p, the subset $P^k \subset P$ which stores all the publications including the keyword k, and the subset $P^{k,c} \subset P^k$ which contains the publications which both include k, and are mapped to the field c. In Table 5.20 we summarize all the above notifications.

[14] In this work we use the word journal to collectively refer to journals, magazines, books, and conference proceedings.

Symbol	Meaning
P	The set containing all papers
A	The set containing all authors
C	The set containing all research areas (taxonomy)
J	The set containing all journals
K	The set containing all keywords
A^p	The authors who created p
C^p	The research areas that p belongs to
K^p	The keywords included in p
P^c	The papers belonging to c
P^a	The papers authored by a
$P^{a,c}$	The papers authored by a and belong to c
P^j	The papers published in j
$P^{j,c}$	The papers published in j and belong to c
P^k	The papers containing k
$P^{k,c}$	The papers containing k and belong to c
$\mathcal{T}$	The training set

Table 5.20: Summary

5.5.2.1 Model Training

In this phase we process the training set $\mathcal{T}$ and we construct a classification model with respect to the taxonomy C. This procedure includes three stages where we correlate keywords, authors and journals to one or more labels from C. We also record several frequency values which will be used later by the classification algorithm to effectively determine the labels of the unclassified papers.

The majority of the research articles includes a set of keywords placed between the abstract and the first section. Moreover, the words occurring in the title are also considered representatives of the document's content and can also be used in our model. Now consider a paper $p \in P$ drawn from the training set $\mathcal{T}$ which includes the keywords K^p and is categorized into one or more research fields $C^p \subset C$. Our objective is to create correlations between each keyword of p and each of the research fields of C^p. Since a keyword k may appear in multiple papers belonging to different research areas, we adopt a strategy similar to the k-NN; that is, we examine how frequently this keyword has been mapped to each field c. This is achieved by the construction of (k, c) pairs which we store in a *relevance description* vector $\mathcal{K}$ [54]. We also compute two frequency values $|P^k|$ and $|P^{k,c}|$: The former represents the number of papers including the keyword k, whereas the latter signifies the number of papers which both include k and are mapped to the field c.

Algorithm 10 shows the steps required to train $\mathcal{K}$. For each paper p of the training set we initially identify all the research fields C^p and the keywords K^p. For each research field

$c \in C^p$ we create a (k, c) pair and we search for it within $\mathcal{K}$. If the search is not successful, we insert the pair and we set $|P^{k,c}| = 1$; otherwise we merely increase $|P^{k,c}|$ by one. This procedure leads to the vector $\mathcal{K}$, which contains all the keywords of the papers accompanied by a global frequency value $|P^k|$ and a list of the corresponding research fields.

The previous activity of the authors who contribute to a research paper can also provide an indication of the research field the paper discusses. Learning the areas of expertise of a scholar is important since it can be exploited to classify his/her unlabeled articles. However, a scientist usually conducts research in multiple areas of science. For instance, consider an author X who has published articles in the areas of *databases* and *data mining* and for the production of these articles has co-operated with Y and Z respectively. Intuitively, an article authored by both X and Z should be labeled as a *data mining* paper.

To capture these intuitions we create a vector $\mathcal{A}$ which for each author a accommodates a list of all the encountered co-authors a' (*AA list*). Each co-author entry is accompanied by an array with the research fields of the articles authored by both a and a'. Hence, in case a and a' are encountered again in an unlabeled article, we retrieve the research fields of their common articles from the aforementioned list. Furthermore, for each author a we also create one more list (*AP list*) which stores all the research fields of all the papers of a. This record will be used to classify articles authored by multiple authors, but no previous co-authorship information between a and the co-authors is available.

Similarly to the previous stage, each research field is accompanied by two frequency values $|P^a|$ and $|P^{a,c}|$. The former represents the number of publications of a, whereas the latter denotes the total number of publications of a which are mapped to the research area c. The steps 14–30 of Algorithm 10 describe the construction process of $\mathcal{A}$. For each author a, the correlations (a, c) are all inserted in the *AP* list (steps 18–23). In the sequel, we iterate through each co-author an we create (a, a', c) tuples which correlate the author, his/her co-authors, and each field (steps 24–30).

Finally, the publishing journal can provide an indication about the research area that a paper belongs to. This is due to the fact that journals are also categorized and do not publish articles which deal subjects that are foreign to their interest area. For instance, a journal which is focused on *Data Engineering* would not publish a paper which discusses a problem related to *Systems Security*.

The aforementioned notifications lead to the enhancement of our trainer with its third part (Algorithm 10, steps 31–39). Here we identify the research areas that the majority of the papers published in a journal j belong to. Compared to the other two stages this one is slightly simplified because a paper is only published in one journal and hence, we do not have to iterate through multiple journals. The outcome of this process is the $\mathcal{J}$

Algorithm 10 Model training

initialize $\mathcal{K}$, $\mathcal{A}$, $\mathcal{J}$
for each paper $p \in \mathcal{T}$
    $C^p \leftarrow$ ExtractResearchAreas(p)
    Phase 1: Processing of the keywords
    $K^p \leftarrow$ ExtractKeywords(p)
    for each keyword $k \in K^p$
        $|P^k| \leftarrow |P^k| + 1$
        for each research area $c \in C^p$
            Create pair (k, c)
            if $\mathcal{K}$.search(k, c) =**false**
                $\mathcal{K}$.insert(k, c)
                $|P^{k,c}| \leftarrow 1$
            else
                $|P^{k,c}| \leftarrow |P^{k,c}| + 1$
    Phase 2: Processing of the authors
    $A^p \leftarrow$ ExtractAuthors(p)
    for each author $a \in A^p$
        $|P^a| \leftarrow |P^a| + 1$
        for each research area $c \in C^p$
            Create pair (a, c)
            if $\mathcal{A}.AP$.search(a, c) =**false**
                $\mathcal{A}.AP$.insert(a, c)
                $|P^{a,c}_{AP}| \leftarrow 1$
            else
                $|P^{a,c}_{AP}| \leftarrow |P^{a,c}_{AP}| + 1$
            for each author $a' \in A^p$
                Create tuple (a, a', c)
                if $\mathcal{A}.AA$.search(a, a', c) =**false**
                    $\mathcal{A}.AA$.insert(a, a', c)
                    $|P^{a,c}_{AA}| \leftarrow 1$
                else
                    $|P^{a,c}_{AA}| \leftarrow |P^{a,c}| + 1$
    Phase 3: Processing of the journals
    $j \leftarrow$ ExtractJournal(p)
    $|P^j| \leftarrow |P^j| + 1$
    for each research area $c \in C^p$
        Create pair (j, c)
        if $\mathcal{J}$.search(j, c) =**false**
            $\mathcal{J}$.insert(j, c)
            $|P^{j,c}| \leftarrow 1$
        else
            $|P^{j,c}| \leftarrow |P^{j,c}| + 1$

vector, which contains for each journal, a list of research fields accompanied by their corresponding frequency values $|P^{j,c}|$. These values indicate how many times a journal j has

published papers belonging to c.

5.5.2.2 Articles Classification

We can now employ the trained model to classify an unlabeled article $p \in P$. Similarly to the training phase, classification is also conducted in three phases. During each phase, the involved research fields are assigned scores according to their correlation with the keywords, authors, and journal of p. In Algorithm 11 we describe these procedures.

In the first phase (steps 2–6), we initially extract the paper's keywords K^p and for each retrieved keyword k we perform a search in the relevance description vector $\mathcal{K}$. In case this search is successful, we retrieve the list of the associated research areas along with the respective $|P^{k,c}|$ values. Then, for each research field c we compute a score $\mathcal{S}_k^c$ according to a scoring function $\mathcal{S}_k^c = F_k(P^k, P^{k,c})$. This function can implement simple schemes such as the traditional *idf* (i.e. $|P^{k,c}|/|P^k|$), or more complex ones. The steps 2–6 of Algorithm 11 illustrate the exact process.

Classification is further enhanced by taking into account the information regarding the authors of p. However, in this case the process is more complex, since we must consider the co-authorship data and retrieve the correct record from the vector $\mathcal{A}$. Initially, we identify the set of authors A^p of p. In the sequel, we search among the *AA* co-authorship records and in case a correlation between two authors is found, each corresponding research field is assigned a score $\mathcal{S}_a^c = F_a(P_{AA}^a, P_{AA}^{a,c})$ (steps 10–14). If such a correlation is not present in $\mathcal{A}$, we search in the *AP* list and we use the associated *AP* scores (steps 15–18).

Moreover, our classifier takes into consideration the research fields of specialization of a journal. Hence, in case the publishing journal j of an unlabeled paper p has been encountered during the training phase, we can exploit the correlated research fields (stored within the vector $\mathcal{J}$) to classify p. The third part of Algorithm 11 describes our approach. After the identification of the journal j of p, we perform a look-up in $\mathcal{J}$. In case searching is successful, we retrieve the research fields that the published articles of j belong to, along with their respective frequency values. For each of these fields we calculate a score given by a third function $\mathcal{S}_j^c = F_j(P^j, P^{j,c})$.

The total score assigned to a research field c is a linear combination of the three aforementioned scores:

$$\mathcal{S}^c = w_k \mathcal{S}_k^c + w_a \mathcal{S}_a^c + w_j \mathcal{S}_j^c \tag{5.9}$$

where w_k, w_a, and w_j are constant parameters used to regulate the contribution of the keywords, the authors, and the publishing journal of an article. These three constants are

Algorithm 11 Paper Classification

$Classify(p_i, F, \mathcal{K}, \mathcal{A}, \mathcal{J})$
1. **for each** unlabeled article p

Phase 1: Keyword-based classification
2. $K^p \leftarrow$ ExtractKeywords(p)
3. **for each** keyword $k \in K^p$
4. **if** $k \in \mathcal{K}$
5. **for each** $(k, c) \in \mathcal{K}$
6. $\mathcal{S}_k^c \leftarrow F_k(P^k, P^{k,c})$

Phase 2: Author-based classification
7. $A^p \leftarrow$ ExtractAuthors(p)
8. **for each** author $a \in A^p$
9. $coauthor \leftarrow$ **false**
10. **for each** author $a' \in A^p$
11. **if** $(a, a') \in \mathcal{A}.AA$
12. $coauthor \leftarrow$ **true**
13. **for each** $(a, a', c) \in \mathcal{A}.AA$
14. $\mathcal{S}_a^c \leftarrow F_a(P_{AA}^a, P_{AA}^{a,c})$
15. **if** $coauthor =$ **false**
16. **if** $a \in \mathcal{A}.AP$
17. **for each** $(a, c) \in \mathcal{A}.AP$
18. $\mathcal{S}_a^c \leftarrow F_a(P_{AP}^a, P_{AP}^{a,c})$

Phase 3: Journal-based classification
19. $j \leftarrow$ ExtractJournal(p)
20. **if** $j \in \mathcal{J}$
21. **for each** $(j, c) \in \mathcal{J}$
22. $\mathcal{S}_j^c \leftarrow F_j(P^j, P^{j,c})$

tuned with the aim of satisfying the following limitation:

$$w_k + w_a + w_j = 1 \tag{5.10}$$

To identify the research field a paper belongs to, we merely have to sort the $\mathcal{S}^c$ scores in descending order and select the first entry (the one received the highest score, $max(\mathcal{S})$). In this way, each article is mapped to only one research field. We can raise this limitation and classify an article into multiple research areas, by introducing a coefficient $\epsilon \in (0, 1]$. Then, each paper is assigned additional research areas if the scores of these areas satisfy the following condition:

$$\mathcal{S}^c \geq \epsilon \, max(\mathcal{S}) \tag{5.11}$$

The value of ϵ determines how strict this condition can become. For $\epsilon = 1$ we tolerate no fields with scores lower than the maximum and an article is mapped to the research area

(or areas) which received the highest score.

5.5.3 Experimental evaluation

In this section we measure the effectiveness of our proposed algorithm. Initially we describe the employed dataset and the taxonomy structure, and in the sequel we present the results of our performance measurements.

5.5.3.1 Dataset and taxonomy characteristics

To experimentally attest the effectiveness of our classification approach it is required that we firstly determine the set of labels which will be used by the classifier (i.e. the taxonomy), and a dataset comprised of an adequate number of articles. In addition, a subset of these articles must support the selected taxonomy, that is, the items of this subset must be mapped to at least one of the labels of the taxonomy.

The strict data protection policies applied by the digital libraries renders the collection of bibliometric data a rather challenging task. Since the crawling of such repositories is forbidden, we are limited in using only open access document collections. The largest among these collections is the CiteSeerX [33] dataset, an open repository comprised of approximately 1.8 million scientific articles. These articles are related to the wider fields of computer and communications engineering, and a significant portion of them are mapped to the local taxonomies employed by their publishers. Of course, each publisher applies its own taxonomy structure; consequently, the first issue we need to address is to determine a unique taxonomy which will be used to classify the rest of the articles.

Since our primary goal is to build a training set comprised of largest possible number of articles, we simply scanned our dataset to identify the organization which published the most documents. Our analysis proved that the 63% of the articles of the CiteSeerX dataset has been published by either ACM or IEEE. These publishers employ a common categorization policy; they classify their published articles into a taxonomy[15] of research fields which mainly includes areas and sub-areas from the Computer Science, Engineering, Communications, and Mathematics. The structure consists of three levels of categorization, i.e. 11 first-level research fields divided into 81 second-level and 276 third-level classes. To achieve extensive and unbiased measurements of the effectiveness of our algorithm, we employed each level as a different taxonomy and we gave the names $C11$, $C81$ and $C276$.

After the selection of the taxonomy, the training set is immediately identified. All 1,159,634 articles which are mapped to one ore more labels of the ACM/IEEE taxonomy,

[15] http://www.acm.org/about/class/ccs98-html

$\|\mathcal{T}\|$	C	$\{w_k, w_a, w_j\}$	**Acc.**	**SVM**	**Ada**
10,000	$C11$	{0.3, 0.1, 0.6}	94.0%	88.2%	88.8%
	$C81$	{0.2, 0.1, 0.7}	87.5%	82.9%	83.4%
	$C276$	{0.2, 0.1, 0.7}	80.7%	78.4%	80.1%
100,000	$C11$	{0.3, 0.2, 0.5}	95.1%	89.6%	-
	$C81$	{0.3, 0.1, 0.6}	88.2%	84.3%	-
	$C276$	{0.2, 0.2, 0.6}	80.9%	79.0%	-
1,159,634	$C11$	{0.3, 0.2, 0.5}	95.9%	94.1%	-
	$C81$	{0.3, 0.2, 0.5}	89.0%	87.9%	-
	$C276$	{0.3, 0.1, 0.6}	81.3%	80.8%	-

Table 5.21: Optimal tuning of the w_k, w_a, and w_j parameters for the three employed taxonomy structures and for training sets of different sizes.

are automatically becoming members of the training set. Our goal now is to assign labels to the rest 684,638 articles.

5.5.3.2 Model training

The model training process includes two separate phases: initially, we construct the relevance description vectors $\mathcal{K}$, $\mathcal{A}$, and $\mathcal{J}$ and in the sequel, we attempt to evaluate the w_k, w_a, and w_j parameters of equation 5.9 which maximize the performance of our classifier.

For this reason, we applied a cross validation strategy according to which the training set was split in three equally-sized parts. The first two thirds were used for building $\mathcal{K}$, $\mathcal{A}$, and $\mathcal{J}$. In the sequel, we used the last third of the training set to measure the classification performance for all the possible combinations of w_k, w_a, and w_j. In particular, we continuously modified the values of all three parameters in the range $[0.0, 1.0]$ (with respect to equation 5.10) and we recorded the number of papers for which our classifier assigned correct labels. To verify the correctness of our results and to eliminate any random effects, we experimented with different training set sizes. More specifically, we repeated our measurements by using training sets comprised of 10,000, 100,000, and all the 1.16 million articles.

In Table 5.21 we report the results of this experiment. The first column denotes the training set sizes, whereas in the second column we show which taxonomy structure is employed. In the third column we record the values of the w_k, w_a, and w_j for which our classifier achieved maximum performance, whereas in the last three columns we report the accuracy of our algorithm against two methods based on support vector machines (SVM) and AdaBoost.MH (Ada).

Notice that our proposed classifier exhibited equally high effectiveness for multiple

combinations of w_k, w_a, and w_j. From Table 5.21 we conclude that among the three exploited types of data (i.e. keywords, authors, and journal), the publishing journal is the most important indication of the research field an article belongs to. On the other hand, the previous works (i.e. the history) of the authors is the weakest one. A second conclusion is that our algorithm is relatively insensitive to the training set size; its performance degrades by only a small percentage (i.e. 1-2%) when the training set size is decreased by a factor of 10. In all cases, a satisfactory setting for the three constants is $w_k \in [0.2, 0.3], w_a \in [0.1, 0.2]$ and $w_j \in [0.5, 0.6]$.

In addition, we observe that performance decreases as the number of the available labels increases. This is expected since as the size of the taxonomy increases, the possibility of an erroneous prediction also increases (the classifier has more available choices). However, the possibility is not proportional to the taxonomy size; Although $C81$ includes about 8 times more labels than $C11$, the effectiveness degrades by only a percentage of 9-10%. Finally, we point out the remarkable 94% of successful labeling in the case of our small $C11$ taxonomy.

Now let us compare our approach against the state-of-the-art method based on SVMs. Since the available label sets consist of multiple entries, it is required that we use multi-class SVMs. In particular, we use the *one-against-all* strategy which given a set of y labels, it requires the construction of one binary classifier per label. To decide which labels to assign to each article, we take the classes that present the largest margins. The features we selected for SVM training were identical to the ones of our proposed algorithm, i.e. keywords, authors and journals. The one-against-all strategy is far preferable than the other existing approach, *one-against-one*, which performs pairwise classifications and requires the construction of y^2 classifiers.

To construct each binary SVM classifier, we created an equal number of training sets. For instance, for the experiments with the $C11$ label set we created 11 training sets. Each training set comprised of all the papers mapped to this specific research field (positive examples), followed by the rest of the articles which were declared as negative examples. We then created one binary classifier for each label, by using the SVMLight Program [74]. Finally, we scanned the outputs of these classifiers and each article was assigned the label which presented the largest margin. The precision results are recorded in the last column of Table 5.21.

Our approach outperformed the SVM-based approach in all of the examined cases. The results of Table 5.21 reveal that the performance gap between the two methods increases as the training set size becomes smaller. More specifically, in case the training set consists of 10 thousand articles, the precision of our proposed method is higher than the precision

Vector	Records	Most Frequent	Articles
$\mathcal{K}$	475,308	system	144,295
$\mathcal{A}$	497,604	Philip S. Yu	654
$\mathcal{J}$	3,915	Theor. Computer Science	13,295

Table 5.22: Trained Model Statistics

	Label	Articles
1	Comp. Methodologies	278,179
2	Inform. Systems	99,958
3	Systems Organization	71,635
4	Math. of Computing	69,140
5	Software	66,391

	Label	Articles
1	Artificial Intelligence	174,272
2	Comp.-Comm. Networks	83,410
3	Numerical Analysis	62,211
4	Software Engineering	47,962
5	Interfaces & Presentation	29,617

	Label	Articles
1	General	185,718
2	Net. Archit.-Design	49,423
3	Nonnum. Algorithms	43,153
4	Applications	20,161
5	Types-Design Styles	19,218

Table 5.23: Classification results for the three experimental taxonomy structures, for $\epsilon = 1$: (i) Left: $C11$, (ii) Center: $C81$, and (iii) Right: $C276$

of the SVMs by a margin ranging between 2% and 6%. On the other hand, the smallest performance gap was observed in the scenario where the employed label set is $C276$ and the training set consisted of all the 1.16 million articles.

We also compared our algorithm against another high-performing classification algorithm, AdaBoost.MH. For our small training set comprised of 10,000 articles, our approach achieved better performance by a percentage ranging between 1% and 6%. In the other two larger training sets, AdaBoost.MH consumed all the available memory (12GB) of our machine and the experiments failed to complete. On the contrary, the in-memory data structures of our approach occupied roughly 1.1 GB for the largest training set.

In Table 5.22 we present some interesting statistics of our model. The keywords relevance description vector is comprised of about 475 thousand keywords, and the most frequent keyword is *system*. Furthermore, the most frequent author and journal were *Philip S. Yu* and *Theoretical Computer Science*, which authored and published 654 and 13,295 articles respectively.

5.5.3.3 Classification Results

Now we employ the trained model of the previous phase to classify the 684,638 unlabeled articles of our dataset. As previously, we conducted the same experiment three

	Label	Articles
1	Comp. Methodologies	347,739
2	Inform. Systems	144,668
3	Math. of Computing	97,514
4	Systems Organization	92,136
5	Software	91,331

	Label	Articles
1	Artificial Intelligence	224,183
2	Comp.-Comm. Networks	101,391
3	Numerical Analysis	85,764
4	Software Engineering	65,010
5	Interfaces & Presentation	41,545

	Label	Articles
1	General	237,761
2	Nonnum. Algorithms	64,204
3	Net. Archit.-Design	60,994
4	Applications	31,826
5	Design Methodology	29,369

Table 5.24: Classification results for the three experimental taxonomy structures for $\epsilon = 0.85$: (i) Left: $C11$, (ii) Center: $C81$, and (iii) Right: $C276$

times and each time we employed one of the taxonomy structures $C11$, $C81$, and $C276$. In Tables 5.23 and 5.24 we illustrate the five most popular research fields for each of our employed taxonomy structures for $\epsilon = 1$ and $\epsilon = 0.85$ respectively.

In the case of $\epsilon = 1$ the unlabeled articles are assigned only one research field (the one which received the highest score according to equation 5.9). Regarding the $C11$ taxonomy, the most popular research field was *Computing Methodologies*; the 40.6% of the articles were classified to this category. Furthermore, *Artificial Intelligence* and *General* were the most popular research areas of the $C81$ and $C276$ taxonomy structures respectively. In particular, the former label was assigned to the 25.4% of the unlabeled articles, whereas the latter gathered the 27.1% of the articles.

On the other hand, in the case of $\epsilon = 0.85$ the scientific documents can be assigned more than one labels, if the scores of the additional labels obey to the limitation of equation 5.10. Therefore, it is anticipated that each research field is assigned to more articles than in the previous case. For this reason, the orderings of Table 5.24 are slightly modified compared to the ones of Table 5.23.

5.6 Conclusions

In this chapter we studied four interesting problems related to academic search engines and digital libraries. We described another dimension of publication methodologies, the existence of coterminal citations and set forth an effort to discover such patterns in citation networks. The proposed f-index represents a computerized, automated way to assign weigh/value to citations, although it is not alone sufficient to determine the function and value of citation and, for instance, their cognitive background should also be taken into consideration [55].

The astute reader will have realized by now that in our efforts to recognize and weigh coterminal citations, we have in our arsenal the research works dealing with Web link spam [66] e.g., TrustRank, BadRank and so on. Unfortunately, the situation is radically difficult in citation networks, because they consist of entities richer than the Web pages and the Web links encountered in Web spam. Each node i.e., a citing article, in a citation network consists of entities i.e., co-authors, which form a complex overlay network above the article citation network.

In addition, we studied the problem of identifying attractive research areas for new scientists. Since this is a new issue, we initially described the properties of the space where the problem is set and solved. In the sequel, we identified the characteristics of the new scholars and the attributes of the attractive research areas. We distinguished popular research

areas from attractive, and we stated that popularity does not render a topic of research attractive for new scientists. Therefore, to measure the attractiveness of a research field for a new scholar, we presented two scoring schemes which incorporate multiple different parameters such as the number and the recency of the published articles and their citations, the number and the reputation of the involved authors and the reputation of the publishing journals.

Our methods have been attested experimentally by employing a large set of self-crawled research articles. The experiments provided some significant conclusions: The first is that there are exist some research fields which despite their popularity, they are not attractive for scholars who are now starting their career. On the other hand, some research fields are unpopular however, they provide excellent opportunities at these scientists.

Although the computation of scientometrics for small datasets is relatively easy, for larger volumes of data the problem becomes more complex. The modern academic search engines now store tens or hundreds of millions of research articles; this data size in combination with their increasing popularity, has rendered the issue of computing scientometrics in parallel both interesting and challenging. Here we studied the issue of computing the scientometrics in large-scale academic search engines with MapReduce. We introduced four methods to compute three of these metrics, h-index, and two variations, the contemporary and trend h-indexes. However, these methods can be applied to compute a wider variety scientometrics with no additional effort. We proposed optimizations with the aim of decreasing the size of the data exchanged among the nodes of the system, and we conducted experiments with the CiteSeerX dataset, a large repository comprised of about 1.8 million research articles. Our experiments demonstrated the usefulness of method 1-C, a strategy which achieves both effective and efficient execution.

Finally, we introduced a supervised machine learning algorithm for classifying research articles. The problem we examine is particularly useful for academic search engines and digital libraries, since a robust solution can provide improved functionality and performance benefits. Our algorithm operates by using a predefined list of labels (i.e. taxonomy structure) and a training set comprised of labeled articles. The training process we proposed is based on the creation of three vectors which correlate each article keyword, author, and journal with a number of labels of our given taxonomy. During the classification process of the unlabeled articles, we use the data stored within these vectors to compute scores for each label. Our experimental evaluation on a large set of 1.5 million research articles demonstrated that our approach achieved successful classification by a percentage above 90%.

CHAPTER VI

Rank Aggregation Methods

6.1 Introduction

The lack of any specific structure and the vast amount of information published on the Web, makes it extremely difficult for the user to find the information s/he desires without any external help. As of September 2012, there are at least 19 general-purpose search engines[1], as well as numerous special-purpose search engines (called vertical search systems). Their population is mainly justified by two reasons: a) no ranking algorithm is broadly acceptable, although many users tend to consider *Google*'s ranking method as the most successful; b) no engine can achieve large coverage and high scalability. It is a common belief [133, 92] that a single general purpose search engine for all Web data is unrealistic, since its processing power cannot scale up to the rapidly increasing and unlimited amount of Web data.

The tool which attracted the acceptance among the users is *metasearch engines* [94]. These systems operate like a filter of the various crawler-based or directory-based search engines which they combine. Metasearch engines run simultaneously a user query across multiple *component search engines*, retrieve the generated results and then aggregate them. Finally, they present the best among them to the user.

The advantages of metasearch engines against search engines are significant [94]:

- They increase the search coverage of the Web, providing a higher *recall*. The overlap among the major search engines is usually very small [132] and it can be as small as 3% of the total results retrieved. On the other hand, the unique results can be as high as 85% of the total results retrieved by all component engines.
- They solve the scalability problem of searching the Web and they facilitate the exploitation of multiple search engines enabling consistency checking [97].

[1] See http://www.searchenginewatch.com

- They improve the retrieval effectiveness providing higher precision, due to the "chorus effect" [144].

Consequently, metasearch engines and their Web 2.0 successors, mash-up services are important tools and they are becoming increasingly popular. The core of any such system is the ranking function it employs, because this function defines the final ranked result list from the results provided by the component search engines. Hence, devising effective ranking algorithms is a problem of particular importance for metasearch engines and mash-up services.

The problem of rank aggregation is quite old and has been studied for a century, starting from a need to design fair elections. It can be thought of as the unsupervised analog to regression, with the goal of discovering a combined ranking which minimizes the distance to each individual ranking. Despite its seeming simplicity it is surprisingly complicated; finding the optimal combined ranking is NP-hard [45] under certain conditions. Thus, several recent efforts describe approximation algorithms for the rank aggregation problem [7, 6, 35], after showing its relation to the *feedback arc set problem on tournaments* [7]. Some of these are extensively applied to many different research domains such as bioinformatics [42], Web spam detection [45], pattern ordering [134], Web metasearching [87, 114, 122, 124, 106] and many more.

Web metasearching in combination to rank aggregation, is a problem posing its own unique challenges. The results that a metasearch system collects from its component engines, are not similar to votes or any other single dimensional entities: Apart from the individual ranking it is assigned by a component engine, a Web result also includes a title, a small fragment of text which indicates its relevance to the submitted query (textual snippet) and a uniform resource locator (URL). Apparently, the traditional rank aggregation methods are insufficient for providing a robust ranking mechanism suitable for metasearch engines, because they ignore the semantics accompanying each Web result.

Based on these remarks, we conclude that ranking in Web metasearching is a more complex problem than rank aggregation. Individual rankings might be noisy, incomplete or even disjoint, hence they should not be the only parameter affecting ranking. Further processing is required in order to filter the results and allow the final result list of the metasearch engine to be free of unwanted, devious and unfairly highly ranked Web pages. Since commercial interests might frequently and unpredictably affect the results of searching, the user is not clearly protected against the interests of individual search engines. Therefore, the ranking algorithm employed by a real metasearch engine, should be able to provide results that are as free as they can be from paid listings and links.

In this chapter we introduce a family of rank aggregation methods suitable for Web

metasearch engines. The basic family members are the *KE Algorithm* and *QuadRank*, accompanied by a group of special-purpose variations. All the approaches we present in this chapter are positional ranking methods designed to deal with top-k lists returned by Web search engines. Their main features are:

- They assign scores to the candidate results by considering multiple parameters such as the number of the search engines where a particular item appeared, the total number of exploited search engines, the size of the top-k list returned by each search engine, and others.
- They refrain from using any training data in order to perform the rank aggregation, because there is usually no evidence about the underlying data properties and their distributions.
- They do not count upon the *scores* of the individual search engine rankings in order to perform the rank aggregation, because most of the search engines do not provide such scores.

The algorithms are evaluated on real-world data retrieved from four major search engines against individual search engines listings as well as results returned by the most popular metasearch engine Dogpile[2], using *QuadSearch* [3], a metasearch engine developed, among others, as a testbed for rank fusion and rank aggregation. There is also an independent performance study of metasearch engines [18], comparing QuadSearch, Dogpile and Mamma, which showed that QuadSearch outperformed its adversaries in that limited query load.

We also compare our proposed methods to two other existing rank aggregation methods. The first is the well-established *Borda Count* method which assigns scores to the collected documents, by accumulating the individual rankings they received by the component engines. The second method is the *Outranking Approach*, an order-based method presented in [52], which orders the items by specifying a set of thresholds and by comparing each document with all the other collected documents. You can see section 6.2 for a brief description of these two methods and a discussion on their differences from our proposed algorithm.

Initially, we test these methods by utilizing the results from the Web Adhoc task of the Web Track of the TREC-2009 Conference [34]. In the sequel, we report the performance of the examined methods in the real-world environment of QuadSearch.

[2]http://www.dogpile.com

[3]A publicly accessible prototype of QuadSearch is available under http://quadsearch.csd.auth.gr.

The rest of this chapter is organized as follows: in section 6.2 we provide some necessary background material and survey the relevant rank aggregation methods. In section 6.3 we present the KE Algorithm and its accompanying variants, whereas in section 6.4, we describe QuadRank and its implementation details behind the developed metasearch engine. In section 6.5 we present an evaluation of the proposed methods, and finally, section 6.6 contains the basic architectural elements of QuadSearch along with some performance benchmarks.

6.2 Preliminaries and related work

We start with a universe U of items (documents in the context of metasearching); eachitem has a unique identifier c. A ranked list r of items $c_1, c_2, \ldots, c_n$ drawn from the universe U, is an ordered subset $S \subseteq U$, such that $r = [c_1 \geq c_2 \geq \cdots \geq c_n]$, where $\geq$ is an ordering relation on S. Each item $c \in S$, has the attribute $r(c)$ which represents the ranking of c in list r. Rankings are always positive, the best ranking an item could get is 1, and higher ranks show lower preference (reduced relevance to a query, in the context of metasearching).

If r contains all the items of U, then it is said to be a *full or complete list*; if $|r| < |U|$, then it is said to be a *partial list*, and if $|r| = k$, where k is a fixed constant, it is said to be a *top-k list*. Apparently, a top-k list is a special case of a partial list. The ideal scenario for rank aggregation is when each search engine gives a complete list of all the items of the universe related to the keyword terms of a given query. Unfortunately this is not possible since either each component engine has a partial coverage of the Web, or for reasons of speed or protection of the proprietary ranking algorithms, the engine returns only a top-k list. The worst but unusual scenario is when the result lists of component search engines have no overlapping elements. In this case there is nothing that a standard rank aggregation algorithm can do. However, as we will see later, QuadRank takes into account the metadata accompanying each item, in addition to the individual rankings of the search engines and this is an advantage of our method over the other methods.

Two families of rank aggregation techniques exist [114]: a) the *score-based* policies [145], which assign a score to each entity of the individual ranking lists and then use these scores to perform the ranking, and b) the *order-based* (or *rank-based*) policies [45, 122, 23], which work upon the order (rank) information that each entity received in the individual ranking lists. Although there have been proposed a lot of algorithms for rank aggregation, when it comes to applying these techniques to real-world metasearch engines the problem becomes even more complicated.

The methods belonging to the first category, were utilized by the first metasearch engines and they assign a weight to each item of a ranked list, which usually originates from the respective component search engine. The aggregation is performed using these scores; examples of this practice include the works in [145, 21]. Although almost no search engine provides the ranking scores, it is possible to convert local ranks into ranking scores. Although score-based methods appear to be more effective for rank fusion [114], the absence of scores (or denial to reveal) from many search engines' rankings turns these methods problematic.

This shortcoming of the score-based algorithms lead to the generation of the second family of rank aggregation methods, i.e., the rank-based methods, the mainstream for modern metasearch engines. The methods of this family exploit only the relative position of the items in each ranked list to perform the fusion, thus they are also called *positional* methods. A primary advantage of positional methods is their efficiency in calculation, since they can be implemented in time linear w.r.t. the number and size of ranked lists. Unfortunately though, this efficiency in calculation is not accompanied by guarantees to satisfy the Condorcet criterion [153].

A popular positional aggregation method is the Borda Count method [45, 114], which assigns scores based on the positions of the item in the ranked lists. Using terminology from the voting literature, we can see each item of a ranked list as a candidate and each search engine as a voter. Each candidate receives points from each voter according to its rank in the voter's list. For example, the top ranked candidate will receive n points, where n is the number of candidates in the respective ranked list. The total Borda score of the candidate will be the sum of its scores due to each ranked list where it appears. In case that the candidate is not in the top-k list of some voter, then it will receive a portion of the remaining points of the voter (each voter has a fixed number of points available for distribution) or a constant number (0 or 1), depending on the variation of the method [116]. The Borda Count method can be found in different versions, like the weighted Borda Count method [131], where each voter also takes a score and therefore his opinion for a candidate is not treated equally against other voters.

A relatively recent method is the Outranking Approach introduced by [52], which is based on decision rules identifying positive and negative reasons for judging whether a document should get a better rank than another. The method operates by performing pairwise comparisons of each item to all other items in the set S. If c_1 and c_2 are two documents of the set and $r(c_1)$, $r(c_2)$ are their rankings in the list r, then the item c_1 should be ranked higher than c_2 (symbolized as $c_1 \sigma c_2$) if the two following conditions are satisfied:

- The concordance condition which ensures that the majority of the input rankings

are concordant with the ordering $c_1 \sigma c_2$. Formally, the concordance coalition is $C_{s_p}(c_1 \sigma c_2) = \{r(c_1) \ \leq \ r(c_2) - s_p\}$, where s_p is a preference threshold which determines the boundaries between an indifference and a preference situation between documents.

- The discordance condition which ensures that none of the discordant input rankings strongly refutes the ordering the ordering $c_1 \sigma c_2$. Formally, the discordance coalition is $D_{s_u}(c_1 \sigma c_2) = \{r(c_1) \ \geq \ r(c_2) + s_u\}$, where s_u is a veto threshold which determines the boundaries between a weak and a strong opposition to $c_1 \sigma c_2$.

Based on these conditions, a generic outranking relation is defined by the following formula:

$$O(c_1 \sigma c_2) \ \Leftrightarrow \ |C_{s_p}(c_1 \sigma c_2)| \ \geq \ c_{min} \ AND \ |D_{s_u}(c_1 \sigma c_2)| \ \leq \ d_{max} \tag{6.1}$$

where c_{min} and d_{max} are the concordance and discordance thresholds respectively.

Other positional aggregation methods include the Markov chain based on [45], soft computing techniques [23], and median rank aggregation [51]. The first methods "blend" the ranked lists into a Markov chain (MC), where each distinct item of the lists corresponds to a state of the MC and the transition probabilities correspond in some way to the (partial) ranked lists. The goal of this modeling is to find the stationary distribution vector of this MC, which provides a total ordering upon the states of the MC, and thus an ordering upon the items of the ranked lists. Unfortunately creating such a MC takes time $\Theta(n^2k + n^3)$, where n is the number of distinct items and k the size of the (top-k) lists; this computational cost can be reduced to $O(n^2k)$ to obtain a very rough approximation. Soft computing methods make use of genetic or fuzzy logic algorithms to perform the rank fusion, whereas median rank aggregation uses the media rank for each item to perform the final ranking.

Improved positional methods for rank aggregation of partial lists by exploiting the similarity among the items are described in [122], whereas positional methods which are firstly trained and then used for rank aggregation (i.e., supervised rank aggregation) are described in [87]. Nevertheless, when the rank fusion method is going to be implemented as the heart of a metasearch engine, then fast computation of the aggregated rank and effective fusion are the major challenges to be met. Therefore, Markov chain based methods although claimed to be superior among the positional ones, are not the preferred choice over methods with linear time complexity computation cost.

6.3 KE algorithm and variations

The first member of our rank aggregation family is a positional method able to deal with partially ranked lists; actually it deals with top-k lists. The basic algorithm treats all four component search engines equally. The reason we do this is due to the following plain but significant observations: (i) all of them are considered by experts as the "major" search engines, (ii) they have been proved reliable during their lifetimes, and (iii) most users and metasearch engines prefer them to perform their searches. Later in this article, we present a variation of the main algorithm which gives the users the ability to define the importance of each component engine.

Let $\tau_1, \tau_2, \tau_3, \tau_4$ be four ranked lists corresponding to each component search engine. For each item $c \in S$ and list τ_i, we compute the quantity $K_i(c) = position\ in\ \tau_i$ and $\mathfrak{S}(c) = \sum_{i=1}^{4} K_i(c)$. The algorithm assigns to each returned ranked item a score based on the following formula:

$$ke(c) = \frac{\mathfrak{S}(c)}{n^m * (\frac{k}{10} + 1)^n}, \tag{6.2}$$

where $\mathfrak{S}(c)$ is the sum of all rankings that the item has received, n is the number of search engine top-k lists the item is listed in, m is the total number of search engines exploited, k is the total number of ranked items collected from each component engine. We named this weight as *KE*. The less the KE value for an item, the larger the final rank this item will take is.

6.3.1 KE Algorithm vs Borda Count

In this point, we must stress some differences between Borda Count and the KE method.

- The Borda Count method takes into consideration the total number of candidates, whereas KE takes into consideration the number of voters.
- Some Borda Count variations assign scores to each and every candidate; a candidate which is not included in the top-k list of a particular search engine takes a part of the remaining points. This does not hold for the KE method: a candidate will be assigned a score only when it is contained in the top-k list of a particular search engine, otherwise its score is zero.
- The KE method takes into consideration the total number of exploited search engines, the number of search engines where a candidate has been appeared and the size of the top-k list.

- The KE method has better "resolution", in the sense that the possibility of two scores being the same is less than that of the Borda Count.
- The lower the KE weight an item has the higher will be ranked in the final result list. In Borda Count holds the opposite.

6.3.2 Antispam version of KE algorithm

Informally, we say that a search engine has been spammed by a page in its result list when it ranks the page too highly with respect to the other pages, according to the view of a "typical" (average) user. This is unavoidable for search engines, because their ranking algorithms have "defects" that can be exploited by Web page developers in order to achieve an unfairly high page rank [65]. Thus, if a page spams all or even most of the search engines, then the metasearch engine could not handle this problem efficiently as well, because the aggregation function would work with bad data.

This method takes into consideration the *Condorcet Criteria* [153]. In the context of metasearching, these criteria tell us, in a few words, that an item which is enlisted in the top-k lists of some search engines should be ranked above an item that is ranked in the top-k lists of fewer search engines. The antispam method attempts to satisfy the intuition that if a page spams fewer than half of the search engines, then the majority of search engines will prefer a relatively good page to a spam page. The following pseudocode describes the antispam version:

1. *Find which items appear in more than half pages (let the number of these items be* c*).*
2. *Apply the KE method on these items.*
3. *Position them in the result list, starting at rank* 1.
4. *Apply the KE method on the rest of the items.*
5. *Position them in the result list, starting at rank* $1 + c$.

6.3.3 The GeoKE method

The analysis of a result's URL can lead to valuable information about the page hosted under it. The two or three final characters of the domain name (also known as the domain extension), usually reveal the originating country of that page.

A significant number of queries submited to a search engine is directly connected to a specific geographic region. Travelling, vacation and news are cases that fall into this category, as they are usually linked to two types of questions: *where* and *when*. Indeed, travel related queries usually return results that are relative to the travel's destination. Without any doubt, there is a significant possibility that pages hosted under affiliate domain extensions are the most informational.

The possibility that a page is written in a language that a user does not understand is also significant. Hence, an English user querying for *"flights to Moscow"* and is not familiar to Russian, will find the pages written in this language totally useless; these pages usually have the *.ru* domain extension. Apparently, pages having the *.uk* domain extension would probably be the best results. Pages with *.us*, *.au* or *.ca* domain extensions would also be good choices, as there is a great possibility that they are all written in English.

To cover such cases, we present an expansion to the KE algorithm; the GeoKE method that turns it into a geography-aware ranking algorithm. Based on the user's location that is automatically received by the system, the GeoKE method assigns scores to the collected results according to the following formula:

$$\textit{geoke}(c) = G\frac{\sum_1^4 K_i(c)}{n^m * (\frac{k}{10}+1)^n}, \tag{6.3}$$

where G is the the $GeoKE(c)$ which receives values depending on three parameters; the user's locality, the result's geographic origin (revealed by the domain extension) and a comparison between the user's speaking language and the language that the page is written in. Taking the above parameters into consideration, the value of the $GeoKE(c)$ is determined when:

- The result's domain extension and the user's region are the same.
- The user can understand the language the page is written in. In these cases, the system makes use of a local database table is used to store the relationships between the geographical regions and their respective languages.
- The domain's extension of the page does not reveal any information about its locality.
- When the page is written in a language that the user can't understand and does not fall to the former case.

In all four cases described above the $GeoKE(c)$ can have fixed values, but our implementation gives the user the ability to define these values himself. This indicates the

flexibility of $GeoKE(c)$ and enables its use in personalized search systems. By altering these values, the algorithm can lead to different rankings according to the user's selections.

The user's locality is automatically obtained by using a database that is specially designed for matching IP addresses against geographical areas. With this strategy the user is not obliged to submit any information about his/her locality, but there is always a small possibility that the module makes an erroneous prediction.

6.3.4 The weighted KE method

The basic KE algorithm treats all four component engines equally; in the same section we justified this choice. Nevertheless, there is an intuition, that a single search engine can't perform equally well for all types of queries. There are occasions where it can present qualitative results and others, in which the results are of medium or lower informational value. This derives from the fact that each engine maintains its own document index and every time a query is submitted, it searches among different documents and presents different results than another engine does.

In addition, users' confidence to a search engine cannot be ignored. For instance, some of them prefer using a search engine for certain types of queries and another engine for other types. On the other hand, some users prefer using only one search engine to conduct all their searches.

For these reasons, an effective rank aggregation method must provide the user with the ability to modify the importance of each component engine. The proposed Weighted KE algorithm takes this remark into consideration and assigns scores to the collected results according to the following equation:

$$wke(c) = \frac{\sum_1^4 [11 - e(c)] K_i(c)}{n^m * (\frac{k}{10} + 1)^n}, \tag{6.4}$$

where $e(c)$ is the Weight factor of the c-th Engine (EWF), which can receive integer values between 1 and 10. The equation above leads to the same outcome as the basic version of the KE algorithm (6.2), unless a different EWF is assigned to at least one engine. The results with lower scores will receive higher rankings, meaning that an engine considered to be more important than another, must receive a lower EWF. To make it more comprehensible for the user we subtract EWF from eleven, so that the the most important component receives the greatest EWF.

6.3.5 The URL-aware KE method

The use of subdomains when publishing pages on the Web, is a quite common option that the developers have. In fact, subdomains are simple folders in a Web server's public directory and are usually identified by the first part of their URL (after the communication protocol). All subdomains are tied to a domain name and may, or may not have similar contents. For example, the URL *http://inf.uth.gr* is a subdomain of the *uth.gr* domain name; it provides information regarding a University's Faculty.

Until now, all search engines treat subdomains as different domains. The top-10 list that Google returns for the query *Aristotle University of Thessaloniki* contains 10 Web pages, all hosted under the *auth.gr* domain name. It is obvious that the limitation of no more than two pages with the same domain name in the same result list does not cover subdomains. This policy introduces the problem of *domain spamming*, which occurs on situations similar to the one mentioned above.

Hence, there is a significant possibility that many pages with similar contents, or many pages from one source would appear in the engine's result list. This possibility becomes even greater in metasearch engines, where more than one component engines are being exploited. To confront this problem, two solutions are available. The first settlement that can be done is to completely block a number of results having the same domain name, including subdomains.

The second solution focuses on giving these pages a lower score; that is, reducing their ranking. The URL Aware algorithm has been implemented for this purpose. According to this method, the weight for each Web page is computed by using the following formula:

$$\textit{uke}(c) = (11 - D)\frac{\sum_1^4 K_i(c)}{n^m * (\frac{k}{10} + 1)^n}, \tag{6.5}$$

where D is the *Domain Awareness Constant* (DAC). Its value is subtracted from eleven to indicate that a higher value will decrease the result's weight, therefore improve its ranking. The value of DAC depends on how many times the domain name of a specific result is repeated in the list of candidates. If a candidate has a domain name that is not repeated more than two times, we set DAC = 10. In the opposite case, DAC is set equal to 5.

Similarly to GeoKE, the implementation of the URL Aware KE algorithm allows the definition of the value of DAC by the user for both cases.

6.4 The QuadRank scoring

In this section we present another positional method able to deal with partial ranked lists. The proposed method, namely QuadRank, deals with top-k lists originating from single crawler-based search engines. The algorithm treats all component search engines equally. The reason we do this is due to the following plain, but significant observations: (i) all of them are considered by experts as the "major" search engines, (ii) they have been proved reliable during their lifetimes, and (iii) most users and metasearch engines prefer them to perform their searches on.

Let $r_1, r_2, \ldots r_m$ be m ranked lists corresponding to each component search engine. We assume that the each of these lists consists of a fixed number of k items, consequently, the entire process involves in total km elements that have to be merged and ranked. Merging is a procedure which we employ in order to combine the m different result lists into a single list, by removing all the overlapping elements. Notice that this list remains unranked until the scoring function is applied to each of the list's elements.

The overlapping between two component engines varies across different queries and cannot be predicted, hence we assume that our final result list consists of N items. In Table 6.1 we describe the symbols we use in our presentation and the parameters employed by QuadRank.

In the sequel we present the main ideas implemented by our proposed method. In particular, we describe the methodology employed by QuadRank in order to deal with individual rankings and zone weighting and we examine the significance of the results' URL analysis. Finally, we discuss how all these different components can be combined together into a single scoring formula.

6.4.1 Dealing with individual rankings

The ranking that each element receives by the component engines is of primary importance for a rank aggregation method and the majority of the proposed ranking algorithms are mainly based on these rankings. Consequently, we must design our function in order to reward a result which achieves high rankings, since such entries are considered to be more relevant to a given query than others placed in lower positions.

Therefore, for each item $c \in S$ we introduce and evaluate the quantity

$$K(c) = \sum_{i=1}^{m} \Big(k + 1 - r_i(c) \Big) \tag{6.6}$$

where $r_i(c)$ is the ranking that the item is assigned by the i^{th} component engine. In

Symbol	Meaning
m	The number of exploited component engines
c	A single result (document) in a component list
$r_i(c)$	The ranking of c in the i^{th} component list
n_c	The number of component lists containing c
k	The number of items included in each component list
q	An arbitrary user query
t_q	A term of q
Q	The number of terms of q
N	The number of items included in the final merged list
N_{t_q}	The number of items containing t_q
$z_c(i)$	The i^{th} zone of c (see Table 6.3)
$W_{z_c(i)}$	The weight of $z_c(i)$
$f(c, t_q, z_c(i))$	The number of occurrences (frequency) of t_q in $z_c(i)$

Table 6.1: Summary

the special case where the item c is not ranked by list r_i, we assume that $r_i(c) = k + 1$. Obviously, the best score an item can receive is km (if it is ranked first on every component list), whereas the lower score is 1 and it is assigned on a result which was ranked last only in one component list.

The introduced score rewards the items which received high rankings by multiple component engines, however it is relatively frequent that two or more results are assigned equal $K(c)$ values. For instance, consider the occasion where two different items c_1, c_2 receive the following rankings by four top-10 lists r_1, r_2, r_3 and r_4.

Item	r_1	r_2	r_3	r_4
c_1	1	-	-	-
c_2	7	7	10	10

Table 6.2: Example

In the example of Table 6.2 it holds $K(c_1) = K(c_2) = 10$. Nevertheless, we firmly believe that c_2 should be ranked higher than c_1 since:

- it appears in more input rankings, consequently it outperforms c_1 according to the democratic symmetry [116].
- it appears in more than half of the input rankings and hence, there is a smaller probability of being a spam entry according to the Condorcet criterion.

To handle such cases, we must reward the results which are considered as relevant to a given query by as many component engines as possible. If $n_c \leq m$ is the number of the

component engines in which a result c occurs, then we introduce the $rank - based$ score which is determined by the following equation:

$$R(c) = mlog\Big(n_c K(c)\Big) \tag{6.7}$$

Where m is the total number of the exploited engines. Note that the logarithm in 6.7 is employed in order to reduce the deviation among the different values that $R_i(c)$ can receive. Additionally, although multiplying the logarithm with m makes no difference (since m is constant for all items), it is justified by our intention to assign the $R(c)$ score a larger value.

6.4.2 Zone weighting

Zone weighting is a well established methodology for ranking documents in traditional search systems. The main idea it is based on suggests partitioning each Web document into locations of special interest, namely *zones* or *fields*. Such partitioning is usually performed in structured or semi-structured documents (i.e. XML files), where the available information is distributed across multiple zones and represents different semantics (i.e. authorship, publication date, title, etc).

The introduction of zones allows us to compute scores by taking into consideration the physical location of a term within a document (i.e. in which field of an XML document a term appears). An example of such ranking scheme is the BM25F function presented in [89].

The main difficulty hidden behind the idea of using zone scoring in a rank aggregation method is that a Web result retrieved by a component search engine is not a complete document and only some limited representative information is provided to a metasearch engine. This information comprises of three individual semantics: the title, a small fragment of the document's text, called snippet, which indicates the relevance of the document to the user query and a uniform resource locator (URL).

Zone	**ID**	**Weight**
Title	1	10
Snippet	2	3
URL	3	5

Table 6.3: Zones

In this paper we attempt to integrate zone scoring characteristics to our ranking function. The idea we introduce here is to treat the three aforementioned semantics as separate document zones. Although zone weighting is not a new scheme in information retrieval,

to the best of our knowledge we are the first to apply its principles in the field of rank aggregation. Our motives are multiple: A Web result which contains the query terms on its title is possibly more relevant than another which does not. This is also valid for results containing the query terms in their snippets more times than others. Hence, the number of the occurrences of the query terms in the individual zones is another parameter which should be taken into consideration by an efficient ranking method.

As we have mentioned, there already exists a zone scoring scheme, BM25F. Nevertheless, we found that this method is not suitable for our case since:

- The BM25F function requires the length of each zone (in number of terms) to be computed, which undoubtedly is a time consuming operation.
- The second reason that turns the usage of zone lengths problematic is that the Web results returned by the component engines are not complete documents. The small fragments of text used to represent the similarity of each document to the submitted query, have similar or identical lengths for each entry. Consequently, zone lengths have a small contribution to the score of an item.
- It depends on several (typically three) user-defined parameters.

In this subsection we propose our own policy for zone scoring and we believe that our suggestions are more suitable for metasearch applications. At first, in Table 6.3 we determine the weights assigned to each zone. Based on these weights, we use the following formula to compute a weight factor:

$$Z(c) = \sum_{t=1}^{Q} log\frac{N}{N_t} \sum_{z=1}^{3} W_z f(c,t,z) \tag{6.8}$$

where N represents the total number of items included in the final merged list and N_t is the number of items containing the query term t. Furthermore, W_z denotes the constant weight of a zone, which we show in Table 6.3, whereas $f(c,t,z)$ represents the frequency of the query term t within zone z.

The logarithm $logN/N_t$ is drawn from the traditional tf/idf scoring functions used by search engines to rank Web documents (i.e. BM25, BM25F). It is used to reward the documents containing the query terms appearing a few times only, because such terms are expected to reveal the information need hidden behind each query.

Note that the $Z(c)$ score rewards the documents which include the query terms on their title, snippet or URL as many times as possible since it is sensitive to the corresponding frequency values. Consequently, the documents which include all or some of the query

terms in their title multiple times are considered more relevant to the given query and they are assigned higher scores.

To compute the desired frequency values, the text accompanying each result must be appropriately processed. Therefore, each document is tokenized (that is, we obtain all of its distinct terms) and each of the extracted distinct terms passes multiple filters which sequentially perform punctuation removal, case folding and stemming. These are CPU-intensive procedures and one would expect query processing to decelerate significantly. However, the small document population as well as their limited size (titles and snippets consist of a few terms only) render this delay rather inconsequential (see experimental subsection 6.6.1).

6.4.3 URL analysis

To the best of our knowledge, the current publicly known rank aggregation methods do not take into consideration the URLs of the results that the component engines return. In this subsection we attempt to mine the information revealed by the URLs, in order to improve the quality of the produced results.

Since a metasearch engine is a system which exploits multiple resources, it is possible that several results under the same domain name would appear in the final merged list. Although these entries have different URLs and probably include different content, their origin is identical. This observation forces us to conclude that this domain possibly contains adequate information relevant to the given query. Consequently, we should enhance our ranking function in order to reward such results.

On the other hand, a result list comprised of different items from the same resource is hardly informative. An effective search system must provide qualitative results from many sources, because this wide variety partially ensures that the user can locate the desired information. For this reason, the major crawler-based search engines include at most two documents with the same domain name in their result lists, even if there are additional relevant documents.

The problem is now becoming straightforward: from a pool of X documents having the same domain name we must identify and reward the best two among them, whereas we should discard the rest $X - 2$ of them. The challenge becomes even greater if maximum efficiency is required and fast system response is a key issue. QuadRank solves the problem by employing an auxiliary table of *domain accumulators* which it populates during the component engines' result list merging. When an item enters the list, we also search for its domain name into this accumulator table. If the record is not present we insert it and set the corresponding accumulator value equal to 1. In the opposite case we increase the

accumulator by one.

During the scoring process we consult the accumulator table and we assign additional scores according to the following formula:

$$u(c) = log\left(10\frac{2m - 1 + acc_c}{2m}\right) \tag{6.9}$$

where acc_c is the domain accumulator of c. Later, when the results are going to be presented to the user, only the two highest scored results with the same domain name are used; the rest of them are simply discarded. Note that for the results having domain names appearing only once, it holds that $acc_c = 1$ and the quantity $U(c)$ is assigned a value equal to 1.

The analysis of a result's URL can lead to additional information regarding the page it represents. More specifically, the two or three trailing characters of the domain name (also known as the domain extension), usually reveal the originating country of that page.

Additionally, a significant number of queries submitted to a search engine is directly connected to a specific geographic region. For instance the travel, vacation and news oriented searches are cases falling into this category. When such information is required, there is a significant possibility that the pages hosted under affiliate domain extensions are the most satisfactory.

To address such issues, we introduce an expansion to our ranking function, the $GeoFactor$, a parameter which is determined by the relationship between the geographic locality of the user and the proximity of each result c. The Geo Factor receives values according to the following formula:

$$G(c) = \begin{cases} 1 & \text{if domain extension and user locality are not identical,} \\ \lambda, \lambda > 1 & \text{otherwise.} \end{cases}$$

where λ is a predefined constant receiving values $\lambda > 1$; a typical option which we have used in our experiments is to set $\lambda = 1.2$.

By integrating the Geo Factor into the scores of equation 6.9 we introduce the $URLAware$ scores determined by the following formula:

$$U(c) = G(c)log\left(10\frac{2m - 1 + acc_c}{2m}\right) \tag{6.10}$$

The lowest value that $U(c)$ can receive is 1, for results appearing under a domain name

encountered only once in the m component lists and when this domain does not match the geographic locality of the user. On the other hand, if we assume that each component engine returns at most 2 results having the same domain name, then the largest value that $U(c)$ can be assigned is $\lambda log\Big((20m-5)/m\Big)$.

6.4.4 QuadRank scores

In this subsection we merge all the aforementioned components into a single scoring function, namely QuadRank. According to our proposed method, each collected result is assigned a score which is determined by the following formula:

$$Q(c) = U(c)\left(R(c) + \frac{1}{Q}Z(c)\right) \tag{6.11}$$

where Q is the total number of the query terms. As one may notice from equation 6.8, the $Z(c)$ score representing our zone weighting policy, is strongly depended on the number of query terms and increases proportionally to Q. Consequently, to regulate the contribution of zone weighting to the final scores, with respect to the other terms $R(c)$ and $U(c)$, we introduce the quantity $1/Q$.

6.4.5 QuadRank vs. Borda count

In this point, we must stress some differences between Borda Count and the scores of QuadRank.

- QuadRank is a rank aggregation method designed to operate on metasearch engines. Therefore, its scoring formula takes much more parameters into consideration such as the zone weighting, the domain characteristics and the number of the occurrences of each query term within each result.

- Some Borda Count variations assign scores to each and every candidate; a candidate which is not included in the top-k list of a particular search engine takes a part of the remaining points. This does not hold for the QuadRank method: a candidate will be assigned a score only when it is contained in the top-k list of a particular search engine, otherwise its score is zero.

- QuadRank also takes into consideration the total number of exploited search engines, the number of search engines where a candidate has been appeared and the number of items of each top-k list.

- The QuadRank method has better "resolution", in the sense that the possibility of two scores being the same is less than that of the Borda Count.
- Since our proposed method does not assign scores based completely on the individual rankings, it is more difficult to be "deceived" by a spam entry, that is a result which received an unfairly high ranking.

6.4.6 QuadRank vs. Outranking approach

In this subsection we describe the main differences between our proposed method and the Outranking approach.

- The Outranking approach takes into account only the individual rankings that each document received by the component engines. On the other hand, QuadRank takes much more parameters into consideration.
- The Outranking approach is an order-based method, therefore it is only based on comparison among the ranks only. It neither considers scores, nor hits whereas QuadRank assigns scores according to the ranking each item received by the component engines.
- The Outranking approach introduces four user-defined parameters (preference, veto, concordance and discordance thresholds). Selecting different values for these parameters can lead to significant modifications in the produced output ranking. In [52] it is shown that small changes in the values of these parameters lead to rankings with significant quality fluctuations. On the other hand, QuadRank involves settings of weights for each document zone (See subsection 6.4.2 for more details).
- Since the Outranking approach requires a pairwise comparison of each item to every other collected item, it can turn significantly slow particularly when multiple long input lists are involved (because there are more items to be compared).

6.5 Experimental Evaluation

The most important measure of a search system's performance is the quality of its search results. Quality is a decision made after the results evaluation by one or more human users. Moreover, it should not be assumed that quality is absolute; one user may well judge that a result is qualitative, while another says it is not.

In addition, quality is usually identified with the relevancy of the returned results to a given query. Every query represents an information need; the user who submits it, seeks for information that is somehow related to the terms of the query. Consequently, a document is judged to be relevant only if it addresses the stated information need, not because it happens to contain all the words of the query.

To evaluate the performance of a metasearch engine, a basic difference from search engines must be considered. A metasearch engine does not maintain its own document collection. It neither employs crawlers, nor indexers to construct a repository and an underlying index structure. Its output is based on the results that its component engines return. It does not even retrieve the entire set of these results; it works only with the first parts of this set, as these parts are the most promising to contain the best answers to a specified query. Therefore, in the extreme situation where all component engines send irrelative results, there is nothing a metasearch engine can do. But as we show later, a metasearch engine having an effective ranking fusion method, can mine the individual result sets and fuse them to a list that ranks the best items at the top.

A number of methods have been suggested for evaluating the effectiveness of a search system and its ranking algorithm. None of these methods are entirely satisfactory, but this is a natural consequence of attempting to represent multidimensional behavior with a single representative value.

To ensure a fair evaluation of QuadRank against its competitors in the context of retrieval effectiveness, we divide our experimentation into two phases: At first, we use data from the TREC-2009 conference which apart from the result sets, it also provides a list containing a set of queries and the corresponding relevant documents.

In the sequel, we measure the performance of the involved algorithms in the real-world environment of QuadSearch. For this series of experiments there is no ground truth regarding the relevancy of each document to our test queries. For this reason, we asked from six of our colleagues to judge the relevancy of each document and in our analysis we consider a document to be relevant if and only if more than half (four or more) of our colleagues judged its relevance positively.

6.5.1 Retrieval effectiveness evaluation with TREC data

The first phase of our experiments includes the application of our proposed method to the Web Adhoc (WA) Task of TREC-2009 Web Track [34]. This task contains 50 topics (test queries) and 72 participating teams. For each query, each team provides a ranking of about 1000 documents which is later evaluated by using a separate file containing all the documents that are relevant to the given topic. The performances of the 10 best runs of the

WA task are reported in Table 6.4.

Run	MAP	P@5	P@10	P@20	R-Precision
uvamrftop	10.2%	41.6%	37.4%	28.8%	13.6%
UMHOOsd	10.2%	34.8%	37.2%	35.4%	17.5%
UMHOOsdp	10.2%	34.8%	37.2%	35.4%	17.5%
NeuLMWeb300	10.0%	44.8%	44.2%	38.0%	16.7%
NeuLMWeb600	9.8%	39.6%	39.8%	36.1%	16.9%
uogTrdphCEwP	9.8%	42.4%	37.4%	32.1%	15.4%
UMHOObm25B	9.8%	34.8%	36.4%	34.3%	16.8%
WatSdmrm3we	9.5%	16.0%	16.4%	16.6%	14.4%
udelIndDMRM	9.4%	26.0%	31.8%	33.1%	17.0%
udelIndDRSP	9.4%	28.0%	32.8%	31.9%	15.8%

Table 6.4: Performances of the 10 best runs of the WA task of TREC-2009.

For evaluation, we used the 'trec_eval' standard program utilized by the TREC community in order to calculate several measures indicating the retrieval effectiveness of a system. These measures are Mean Average Precision (MAP), R-Precision and Precision@n ($P@n$) for $n = 5, 10$ and 20.

QuadRank is compared against the other two rank aggregation algorithms as well as against some high performing official results from TREC-2009. For each topic, we retrieve one list from each participating team, that is 72 rankings. From these rankings we retain only the first k documents and in the sequel, the 72 top-k lists are merged into one large list which is finally ranked by employing QuadRank, Borda Count and the Outranking approach. In the following tables we present results for different values of k.

To measure the performance QuadRank, it was necessary retrieve the full text of each candidate document in order to compute the $Z(c)$ scores. Since all the candidate documents are from the Clueweb09 dataset, we have developed the appropriate software which operates on the collection and extracts the desired information. $Z(c)$ scores are computed by considering the document's title and URL only, because of the absence of textual snippets. Furthermore, the locality of the user is unknown and the geographical relationship between the user and a document can not be determined. For this reason, we have disabled the Geographic extensions of the $U(c)$ scores in this experimental phase.

Regarding the setting of the parameters introduced by the Outranking approach, we used the same values as those mentioned by [52]. Therefore, we considered that each input ranking is a complete order ($s_p = 0$) and that an input ranking strongly refutes an ordering between two documents when the difference of both document positions is large enough ($s_u = 75\%$). We also supposed that the majority of the rankings must be concordant

($c_{min} = 50\%$) and that every input ranking can impose its veto ($d_{max} = 0$).

Run	MAP	P@5	P@10	P@20	R-Precision
Best TREC Run	10.2%	41.6%	37.4%	28.8%	13.6%
Borda Count (30)	18.5%	39.2%	37.8%	33.8%	26.0%
Outranking Approach (30)	15.5%	42.0%	38.4%	36.8%	22.2%
QuadRank (30)	19.5%	43.2%	39.2%	36.4%	25.4%
Borda Count (100)	19.0%	44.0%	42.8%	41.1%	23.4%
Outranking Approach (100)	18.4%	50.0%	48.0%	40.5%	25.1%
QuadRank (100)	20.9%	46.4%	43.8%	42.3%	25.0%
Borda Count (200)	18.8%	47.6%	44.2%	42.4%	23.1%
Outranking Approach (200)	18.8%	51.6%	47.6%	44.9%	24.3%
QuadRank (200)	20.6%	49.8%	45.9%	44.7%	24.0%
Borda Count (500)	18.2%	50.4%	47.8%	42.1%	22.9%
Outranking Approach (500)	18.7%	50.4%	47.4%	41.3%	23.7%
QuadRank (500)	19.8%	50.4%	48.2%	42.6%	23.4%
Borda Count (1000)	17.3%	49.6%	48.0%	42.7%	22.0%
Outranking Approach (1000)	18.2%	50.8%	45.8%	39.4%	22.0%
QuadRank (1000)	19.9%	51.2%	49.0%	43.1%	21.4%

Table 6.5: Performance of different rank aggregation methods for $m = 72$ and variable numbers of retained documents.

Table 6.5 shows the performance of the three examined rank aggregation methods in the WA task for variable number of retained documents. If we consider Mean Average Precision (MAP) as the comparison measure between the three methods, it is apparent that QuadRank outperforms both Borda Count and the Outranking Approach for all values of k. It is also remarkable that Borda Count performs better than the Outranking Approach for small values of k, whereas the situation is reversed as k increases.

Regarding the average Precision values at cut off points 5, 10 and 20, the performance of the three methods varies significantly. For instance, QuadRank achieved the highest P@10 values for $k = 500$ and $k = 1000$, but for smaller values of k the Outranking Approach becomes the winning method. Furthermore, for $k = 100$ QuadRank achieves the highest P@20 value whereas for $k = 200$ the Outranking Approach fetched the most qualitative top-20 list.

6.5.2 Retrieval effectiveness evaluation with test queries

To examine the retrieval effectiveness of the new ranking algorithm, we have integrated its implementation within *QuadSearch*, our experimental metasearch engine. Various queries were submitted to the system and the results' lists that the proposed ranking

fusion methods returned are recorded and analyzed in this subsection. For all tests demonstrated here we exploited four major component engines (Google[4], Yahoo![5], Live[6] and Ask[7]). The system dispatched our submitted queries to each of these component engines and requested top-30 lists to be returned. Consequently, for the setup that we examine here it holds that $m = 4$ and $k = 30$.

Let us suppose that a user searches for tickets for the forthcoming UEFA Champions League final game and thus, he/she phrases and submits the query *"tickets for uefa champions league final 2010"*. The informational need is overt in this query. As mentioned above, a document not only has to contain the query terms, but also to address the stated information need to be considered as relevant. For the specified example query, a document is relevant only when:

- it is relevant to the *"uefa champions league"* and
- it provides information about *"tickets"* and
- is about the *final* game that will be conducted in *"2010"*.

All documents containing information about the *"uefa cup"* or past *"uefa champions league finals"* (e.g. 2008, 2009) are considered as irrelevant, since they do not satisfy the user's information need.

QuadSearch requested and received 120 results (thirty results from each component engine). After the merging of the results' lists, a set of candidates comprised of 88 unique items is constructed.

The top-20 list that *QuadRank* produced for this query, is displayed in ranked order in Table 6.6. The first column denotes the QuadRank ranking, whereas in the second column we choose to display only the domain name of the returned result, to form a compact and legible table. The next column shows which of the documents are relevant to the given query (with respect to the judgement made by our colleagues) and the symbol R_i is used to indicate the i^{th} relevant document of the QuadRank's top-20 list. The next four columns signify the ranking that the component engines gave to this document[8]. The dash symbol represents the absence of this result from an engine's top-20 list. In the last two columns we compute and display the ranking generated by the Borda Count method and the Outranking approach (column headers *BC* and *OA* respectively), when they are applied on the same result set.

[4]http://www.google.com
[5]http://search.yahoo.com
[6]http://www.live.com
[7]http://www.ask.com
[8]Column headers: *G* for Google, *Y* for Yahoo!, *L* for Live Search and *A* for Ask

	URL	R	G	Y	L	A	BC	OA
1	www.championsleagueticketservice.com	R_1	1	1	-	6	3	5
2	www.championsleagueticketshop.com	R_2	15	3	5	11	2	2
3	www.worldticketshop.com	-	6	-	14	-	16	15
4	en.wikipedia.org	R_3	8	-	1	-	7	10
5	www.soldoutentertainments.com	R_4	10	2	7	7	1	1
6	www.1st4footballtickets.com	R_5	-	-	13	-	45	62
7	www.uefa.com	-	3	-	9	-	9	14
8	www.1st4footballtickets.com	R_6	-	-	3	-	28	61
9	www.livefootballtickets.com	R_7	16	-	10	-	18	13
10	www.championsleagueticketservice.com	-	2	-	-	-	10	78
11	www.nwtix.com	R_8	29	-	22	-	23	21
12	www.roadtrips.com	R_9	4	4	12	-	4	3
13	www.globalticketshop.com	R_{10}	-	-	16	-	31	51
14	www.uefa.com	-	-	-	6	4	8	9
15	www.1st4footballtickets.com	R_{11}	-	-	-	3	29	27
16	www.ticketcity.com	-	12	11	8	-	5	4
17	soccerlens.com	-	14	-	11	-	17	11
18	www.webuytickets.net	R_{12}	23	-	-	-	27	81
19	www.livefootballtickets.com	-	-	24	26	-	19	20
20	www.freetickets.org.uk	R_{13}	5	9	-	-	15	17

Table 6.6: Top-20 list and relevant documents for the query *"tickets for uefa champions league final 2010"* when the *QuadRank* algorithm is applied.

A significant number of results that the search engines returned, were about past *"champions league finals"*. Particularly we found that all component lists contained results regarding the final game of 2009 in their top-10 lists. Some other entries were about tickets for the uefa cup final, whereas others included no information about tickets at all. All these results were considered as irrelevant.

Now let us examine the first ten results returned by each algorithm. Ask performed poorly in this query, since it produced only four relevant documents in its top-10 list, whereas Google and Yahoo were more accurate as they returned six relevant documents. On the other hand, the top-10 ranking that QuadRank produced was the most qualitative, since it included seven relevant results. Note that QuadRank achieved to construct a result list which is improved compared to the rankings of the combined component engines. These notations are all summarized in Table 6.7. The symbol R is used to signify a relevant document, the dash symbol is used to mark the irrelevant ones, whereas S is used for sponsored entries. The last column shows the total number of relevant documents returned by each component engine or ranking algorithm in their top-10 list.

In the same Table we present the top-10 list provided by the most widespread metasearch

Engine	1	2	3	4	5	6	7	8	9	10	R
QuadRank	R	R	-	R	R	R	-	R	R	-	7
Google	R	R	-	R	-	R	-	-	R	R	6
Yahoo!	R	R	R	R	-	R	-	R	-	-	6
Live Search	-	R	R	-	R	-	R	-	-	R	5
Ask	-	R	R	-	-	R	R	-	-	-	4
Dogpile	S	S	S	R	-	-	-	R	R	-	3
Borda Count	R	R	R	R	-	R	-	-	-	-	5
Outranking Approach	R	R	R	-	R	R	-	-	-	R	6

Table 6.7: Relevant Documents in the Top-10 Lists for the Query *"tickets for uefa champions league final 2010"*.

engine, Dogpile. The top three results are sponsored links found on Google Ads[9], an online advertisement service maintained by Google. The fourth result which is organic, is relevant to the query, but it leads to the same location as the second result. It is clear that Dogpile's policy to include paid listings within organic results leads to a confusing top-10 list, which contains different results leading to the same location. In total, Dogpile's top-10 list for this query contains three paid and only seven organic results. From these seven results, only three are relevant to the query, as the other four concern past *"Champions League finals"*.

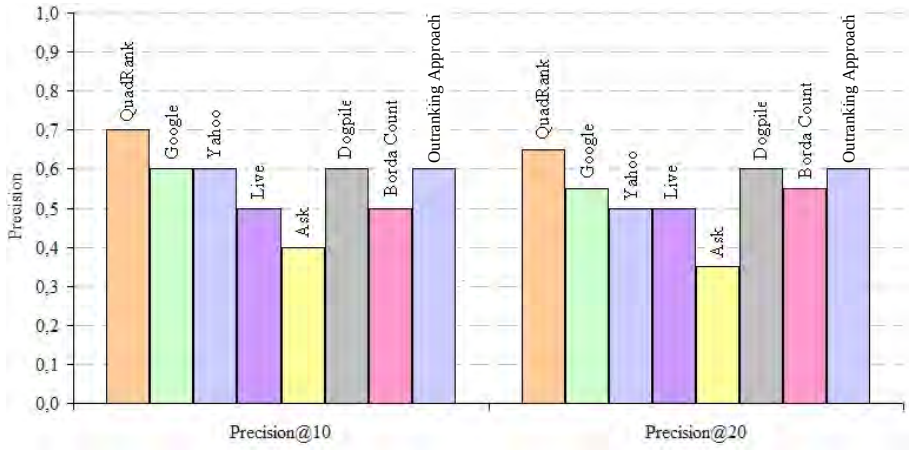


Figure 6.1: Measurements of Precision@10 and Precision@20 for various search engines for the query *"tickets for uefa champions league final 2010"*

In Figure 6.1 we illustrate the effectiveness of the search engines which we examine in our experiments. The two diagrams depict the precision values measured for the first 10 and 20 results respectively. QuadRank outperformed all of its opponents, since it produced

[9]https://www.google.com/adsense

Engine	QuadRank	G	Y	L	A	BC	OA
QuadRank	-	-0.26	0.16	-0.20	-0.07	0.41	0.31
Google	-0.26	-	0.09	0.05	-0.09	0.09	0.25
Yahoo!	0.16	0.09	-	-0.46	-0.42	0.03	0.16
Live	-0.20	0.05	-0.46	-	-0.32	-0.05	-0.03
Ask	-0.07	-0.09	-0.42	-0.32	-	-0.30	-0.63
Borda Count	0.41	0.09	0.03	-0.05	-0.30	-	0.84
Outranking Approach	0.31	0.25	0.16	-0.03	-0.63	0.84	-

Table 6.8: Rankings Correlation for the Query *"tickets for uefa champions league final 2010"*.

the most qualitative result list compared to the component engines, Dogpile, Borda Count and the Outranking approach

The QuadRank algorithm outperformed all of its components including the component engines, Dogpile and the Borda Count and Outranking methods, since it achieved the highest precision values for both the top-10 and top-20 lists. Google, Yahoo and Dogpile constructed top-10 lists of equal quality, whereas the latter's ranking algorithm performed better than the component engines. Regarding Borda Count, the top-10 and top-20 lists included five and eleven relevant results respectively, consequently, our algorithm presented more qualitative results. The Outranking approach was slightly more effective than Borda Count but it was also outperformed by QuadRank.

In addition, the reader should note an interesting case. If we compare the 8^{th} and 10^{th} entries of Table 6.6, we conclude that the latter received better ranking from the component engines than the former. However, since QuadRank is not relied completely on the individual rankings of the component engines, it positions them in the opposite manner. That decision was vindicated, since the 8^{th} entry is relevant to the given query, whereas the 10^{th} is not. There are many such cases in the rankings generated by QuadRank: For example, compare the 6^{th} entry to the 7^{th} and 15^{th} to 16^{th} where the individual rankings were correctly overlooked. On the other hand, the competitor rank aggregation methods could not distinguish the difference and provided incorrect rankings.

Now let us examine how the involved result lists correlate. To evaluate the correlation of the produced rankings, we employed the Spearman's rho measure. The results illustrated in Table 6.8, reveal that all algorithms produce lists that diverge significantly.

In the sequel, we measure the performance of the proposed algorithm for another situation, where a hypothetical engineer seeks information about how to construct an inverted index in a distributed environment. The submitted query is phrased as *"distributed index construction"* and dictates out metasearch engine to request 30 results from each of its

	URL	**R**	G	Y	L	A	BC	OA
1	nlp.stanford.edu	R_1	2	1	2	1	1	1
2	www.reedconstructiondata.com	-	4	4	-	3	2	5
3	www.ims.uni-stuttgart.de	R_2	-	8	5	-	8	6
4	www.reedconstructiondata.com	-	5	-	-	4	6	12
5	ilpubs.stanford.edu:8090	R_3	7	-	3	9	4	2
6	nlp.stanford.edu	-	-	-	4	-	10	21
7	www.ics.uci.edu	R_4	-	-	10	-	41	28
8	lecs.cs.ucla.edu	R_5	-	16	14	22	5	3
9	www.ics.uci.edu	R_6	-	-	12	-	49	27
10	nlp.stanford.edu	R_7	3	2	-	2	3	4
11	ilpubs.stanford.edu:8090	R_8	-	-	15	-	74	19
12	en.wikipedia.org	-	-	7	28	-	25	7
13	ilpubs.stanford.edu	R_9	-	9	-	-	28	42
14	www10.org	R_{10}	-	10	-	-	30	53
15	docs.huihoo.com	R_{11}	9	-	-	11	10	9
16	portal.acm.org	-	-	24	-	-	77	43
17	www.reedconstructiondata.com	R_{12}	-	3	-	-	18	52
18	www10.org	R_{13}	-	-	11	-	46	32
19	www.dcs.bbk.ac.uk	R_{14}	-	30	-	-	97	85
20	citeseerx.ist.psu.edu	-	6	-	-	5	6	8

Table 6.9: The top-20 list for the query *"distributed index construction"* when the *Quad-Rank* Algorithm is Applied.

component engines. *QuadSearch* received 120 results and after the merging process, a set of 100 candidates is generated and ranked.

A document is considered relevant to the given query, only if its content is relevant to *"indexes"* and it contains some instructions regarding *"distributed construction"* in its body. All the results that provided information regarding index compression, organization or single-node construction were marked as irrelevant.

Compared to the previous case (where the set of candidates contained 88 items), we conclude that for this query, the engines' coverage is significantly smaller. At first, the ranking that our algorithm generates, is studied. Table 6.9 illustrates the top-20 list returned by QuadRank and also displays the individual rankings that each result received by the component engines. The last two columns record the rankings that the Borda Count method and the Outranking approach produced.

The Google and Ask component engines provided answers of medium quality for this query; only five results out of ten were considered relevant, as only five contained the required information. Google concentrated on presenting scientific papers about *"indexing"* but some of these works were originating from other sciences such as biology. In addition,

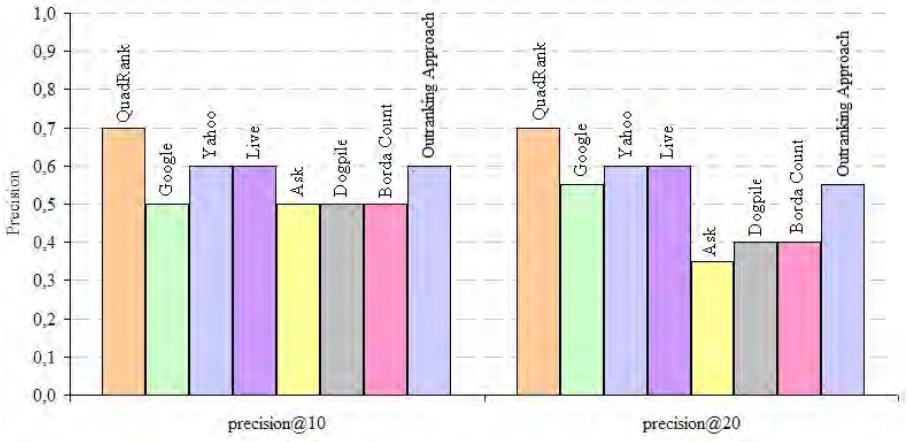


Figure 6.2: Measurements of Precision@10 and Precision@20 for various search engines for the query *"distributed index construction"*

Engine	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**R**
QuadRank	R	-	R	-	R	-	R	R	R	R	7
Google	-	R	R	-	-	-	R	R	R	-	5
Yahoo!	R	R	-	-	-	R	-	R	R	R	6
Live Search	R	R	R	-	R	R	-	-	-	R	6
Ask	R	R	-	-	R	R	-	-	R	-	5
Dogpile	-	R	-	-	-	-	R	R	R	R	5
Borda Count	R	-	R	R	-	-	-	R	-	R	5
Outranking Approach	R	R	R	R	-	R	-	-	R	-	6

Table 6.10: Relevant documents in the engines' top-10 lists for *"distributed index construction"*.

PageRank played its role, as most of the results are well known and institutional sources of scientific information. In addition, Ask returned a top-20 list of low quality, since it was deceived by the term *"index"*, a word that is commonly used to describe the home page of a Web site.

On the other hand, Yahoo! and Live Search performed better in this case, since their top-10 rankings contained six pages containing useful information. These engines also presented satisfactory top-20 lists containing twelve relevant entries each.

QuadRank outperformed all of its opponents, since its top-10 and top-20 lists included 7 and 14 relevant results. Dogpile's top-10 list was of equal quality to Google and Ask, but its top-20 list was more informative only than Ask's. Unlike the previous case where several sponsored links were included in the top-10 list of Dogpile, for this query the metasearch engine provided only organic results. Five of them were relevant to the query, whereas the

others provided information regarding distributed audio and video communication systems and distributed process control systems.

The Borda Count method generated results of equal quality to Dogpile; five relevant items in the top-10 list and only eight in the top-20. It becomes obvious that a simple rank aggregation method that is heavily depended on the individual rankings of the component engines, cannot always provide robust ranking. On the other hand, the component of the QuadRank algorithm (equation 6.8) which examines the titles, the snippets and the URLs of the collected elements, leads to result lists of improved quality.

Once again the Outranking approach was more effective than the Borda Count method since it fetched six relevant documents in its top-10 list and twelve in its top-20. However, the method did not produce as qualitative rankings as QuadRank.

In Table 6.10 we depict the relevancy of each item of the examined top-10 lists to the query in question. Moreover, in Figure 6.2 we illustrate the precision of each ranking algorithm, measured at cutoff points 10 and 20.

Finally, in Table 6.11 we evaluate the correlation of the various rankings of the experiment, by recording the values of the Spearman's rho measure.

Engine	QuadRank	G	Y	L	A	BC	OA
QuadRank	-	-0.42	0.16	-0.09	-0.44	0.56	0.64
Google	-0.42	-	-0.53	-0.33	-0.50	-0.31	-0.32
Yahoo!	0.16	-0.53	-	-0.18	-0.35	-0.20	-0.41
Live	-0.09	-0.33	-0.18	-	-0.27	-0.12	0.35
Ask	-0.44	-0.50	-0.35	-0.27	-	-0.38	-0.48
Borda Count	0.56	-0.31	-0.20	-0.12	-0.38	-	0.89
Outranking Approach	0.64	-0.32	-0.41	0.35	-0.48	0.89	-

Table 6.11: Rankings Correlation for the Query *"distributed index construction"*.

The third query we present here originates from the broader fields of health and medication. Here we are interested in locating the symptoms of the cancer of lungs, hence the query we phrase is *lungs cancer symptoms*. The information need is quite targeted in this query, since we desired to attest the accuracy of our algorithm in similar cases. Hence, the documents containing information regarding only the causes or the treatments of the disease are considered as irrelevant. The same holds for results whose content is about other types of cancers or other types of lungs maladies.

A page is considered relevant to the specified query, only if it contains the desired information itself; not because it includes sets of links to other pages which "claim" to provide such information. This rule was strictly followed by our colleagues during the identification of relevant results.

	URL	R	G	Y	L	A	BC	OA
1	lungcancer.about.com	R_1	13	7	-	-	10	15
2	www.merck.com	R_2	5	-	6	9	5	6
3	www.cancerssociety.org	R_3	-	-	-	14	42	74
4	www.webmd.com	R_4	10	2	7	-	4	4
5	www.emedicinehealth.com	R_5	2	-	3	5	2	3
6	www.cancerhelp.org.uk	R_6	4	-	-	2	7	9
7	www.medicinenet.com	R_7	1	4	1	3	1	1
8	lungcancer.about.com	-	12	6	-	18	6	5
9	www.webmd.com	R_8	-	1	5	-	8	7
10	www.emedicinehealth.com	-	3	5	-	6	3	2
11	www.macmillan.org.uk	R_9	-	-	-	4	26	75
12	www.mayoclinic.com	R_{10}	11	11	-	-	13	19
13	health.yahoo.com	R_{11}	14	13	-	-	14	14
14	www.cancerhelp.org.uk	R_{12}	-	-	-	11	38	73
15	www.lungscancer.com	-	-	-	2	-	24	34
16	www.cancer.gov	R_{13}	-	24	21	-	21	8
17	www.wrongdiagnosis.com	R_{14}	-	8	-	-	31	63
18	www.cancerbackup.org.uk	-	-	-	-	22	63	71
19	www.webmd.com	-	9	-	-	-	33	88
20	www.cancercenter.com	R_{15}	-	17	-	-	50	50

Table 6.12: The top-20 list for the query *"lungs cancer symptoms"* when the *QuadRank* algorithm is applied.

Engine	1	2	3	4	5	6	7	8	9	10	R
QuadRank	R	R	R	R	R	R	R	-	R	-	8
Google	-	R	-	R	R	R	R	-	-	R	6
Yahoo!	R	R	R	-	-	-	R	R	R	R	7
Live Search	-	-	R	R	R	R	R	R	-	R	7
Ask	R	R	-	R	R	-	-	-	R	R	6
Dogpile	-	-	-	-	-	R	R	R	R	R	5
Borda Count	R	R	-	R	R	-	R	R	-	R	7
Outranking Approach	R	-	R	R	-	R	R	R	R	R	8

Table 6.13: Relevant documents in the engines' top-10 lists for *"lungs cancer symptoms"*.

Similarly to the previous experiments, our metasearch engine requested 30 results from each component engine and after the merging process, a set of 91 elements is generated and ranked. In Table 6.12 we record the elements of the top-20 list that QuadRank produced and also, the individual rankings that each one received by the component engines. As before, that dash symbol represents the absence of the element from an engine's result list.

QuadRank presented the most qualitative top-10 list, since 8 results were relevant to

the given query. Furthermore, the first seven results were all of high informational quality, as they included adequate reference regarding the subject of the query. Live search and Yahoo! were also effective, although their top-10 rankings included one result less than QuadRank's list. Regarding the other two component engines, Google and Ask, they were slightly less efficient, by including six relevant results in their top-10 rankings.

The first opponent rank aggregation method, Borda Count, constructed a top-10 list comprised of seven relevant documents and it was outperformed by our proposed method. On the other hand, the Outranking approach returned results that were of equal quality to those of QuadRank. The other metasearch engine that we examine in our experiments, Dogpile, exhibited poor performance and it was the worst among our examined search engines, by building a top-10 list which contained only five relevant results. Table 6.13 provides a detailed image of the top-10 rankings of the examined search engines.

The top-20 ranking of Borda Count is significantly improved compared to the top-10 list and was equally qualitative compared to the ones constructed by our QuadRank algorithm and the Yahoo! component engine. In total, the top-20 lists of these three ranking methods included 15 relevant documents. The left and right diagrams of Figure 6.3 illustrate the precision values of the various systems and algorithms for the top-10 and top-20 listings respectively.

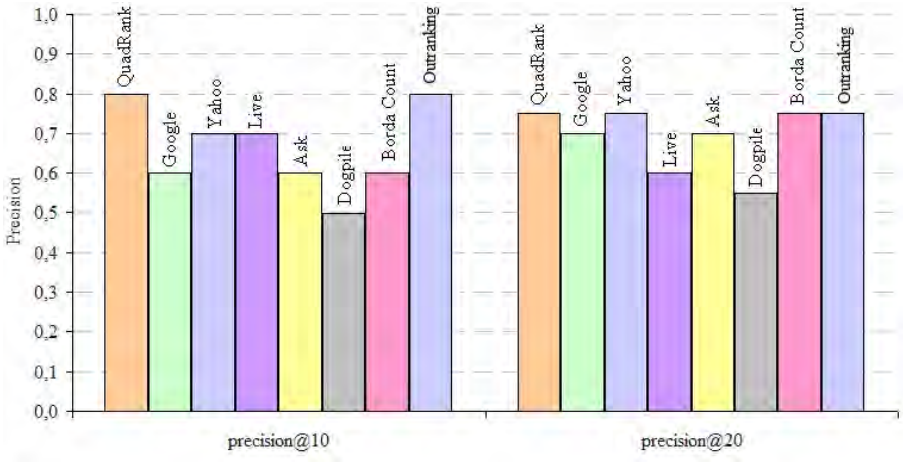


Figure 6.3: Measurements of Precision@10 and Precision@20 for various search engines for the query *"lungs cancer symptoms"*

Finally, in Table 6.14 we record the correlation of the rankings by using the Spearman's rho measure.

The example queries we have studied here reveal that a single search engine cannot perform equally well for all types of queries. Although Google was effective for *"tickets*

Engine	QuadRank	G	Y	L	A	BC	OA
QuadRank	-	-0.41	0.22	-0.26	-0.26	0.65	0.50
Google	-0.41	-	-0.14	-0.48	-0.01	0.82	0.63
Yahoo!	0.22	-0.14	-	-0.58	-0.39	-0.07	0.06
Live	-0.26	-0.48	-0.58	-	-0.50	-0.62	-0.59
Ask	-0.26	-0.01	-0.39	-0.50	-	0.16	-0.07
Borda Count	0.65	0.82	-0.07	-0.62	0.16	-	0.73
Outranking Approach	0.50	0.63	0.06	-0.59	-0.07	0.73	-

Table 6.14: Rankings Correlation for the query *"lungs cancer symptoms"*.

for uefa champions league final 2010", it did not provide equally informative results for the query *"distributed index construction"*. The opposite behavior was exhibited by Live Search and Yahoo, which presented informative top-10 lists in the second and third cases.

On top of that, our QuadRank scoring method performed steadily well on all submitted queries. This sense is present on most cases, as our metasearch engine manages to eliminate the component engines weak spots and combine their advantages efficiently.

Regarding the competitor rank aggregation methods, QuadRank outperformed Borda Count by a significant margin for all of the three queries we have tested. QuadRank was also more effective than the Outranking approach on two cases and equally accurate on the third.

6.6 QuadSearch

In this paper we briefly describe QuadSearch, an experimental metasearch engine that provides simultaneous access in four major conventional, crawler-based search engines. The heart of the system is based on the rank aggregation algorithms we presented previously in this chapter. The engine aims to combine speed, reliable rank aggregation method, "spam" free results, and detailed and enriched information. A publicly accessible interface for the new metasearch engine can be found at http://quadsearch.csd.auth.gr/.

Initially, we describe the core components of the system and in the sequel, we present some performance benchmarks which demonstrate the efficiency of QuadSearch.

6.6.1 Architecture

The most significant modules of QuadSearch are the Quad Bot, the Object Builder, the Classification Module and the Presentation Module. These modules are described in the next subsections. A schematic diagram of the architecture is depicted in the left part of Figure 6.4.

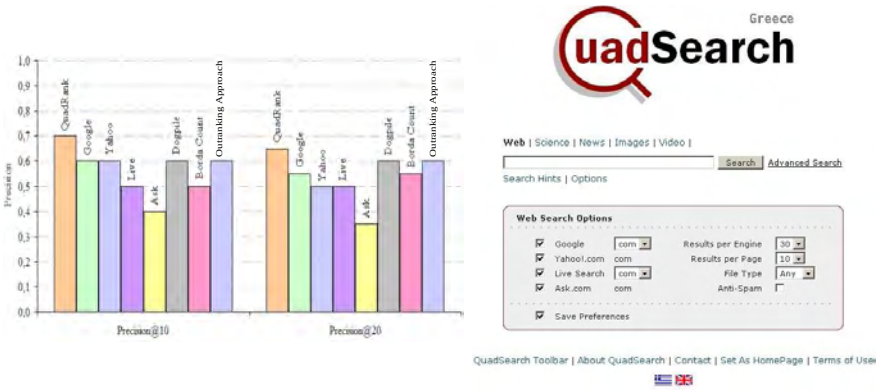


Figure 6.4: (Left) Architecture of QuadSearch. (Right) Quad Search's homepage.

6.6.1.1 User interface and database selector

The static content of Quad Search's user interface is built with plain HTML, while the dynamic procedures are being processed by PHP. Additionally, we invoked Cascading Style Sheets (CSS) for page formatting. We decided that the Web pages' layout should be as simple as possible, in order to ensure: a) short download times, b) compatibility with all major browsers, and c) convenient usage.

For these reasons we avoided using large graphics files, or embedded objects like ActiveX Controls, or Flash presentations. We also rejected Javascript, because many experienced users tend to deactivate it, due to security reasons. Until now, Quad Search supports the classic Web search and also searching procedures for news, images, video and audio sources. The ability to search for scientific articles in the most popular scientific databases is also supported, but not very efficient. The user can switch among these features from either the home page or the results page. Regarding the database selector, the default search will be run using all four search engines. The user however, has the option to choose which search engines will be exploited, as shown in the right part of Figure 6.4.

Apart from the classic text box, we included the most significant search preferences in the home page. The user can select the query resources (the search engines that will participate in the search process), the number of results to be retrieved per resource, the number of results that will be displayed per page etc. The interface provides the ability to store the values of these parameters, by setting cookies in the client's computer. Thus, the user is not obliged to define again these parameters for future queries. The interface includes an extra option to filter the results, to prevent spam records from entering the KE list. Finally, in the options page the user can select the ranking algorithm (KE or Borda Count).

6.6.1.2 QuadBot

QuadBot receives its inputs from both the database selector and the user interface. It is responsible for validating the input data and parameters, passing the query to the selected databases and collecting the results. Its internal structure is depicted in the left part of Figure 6.5.

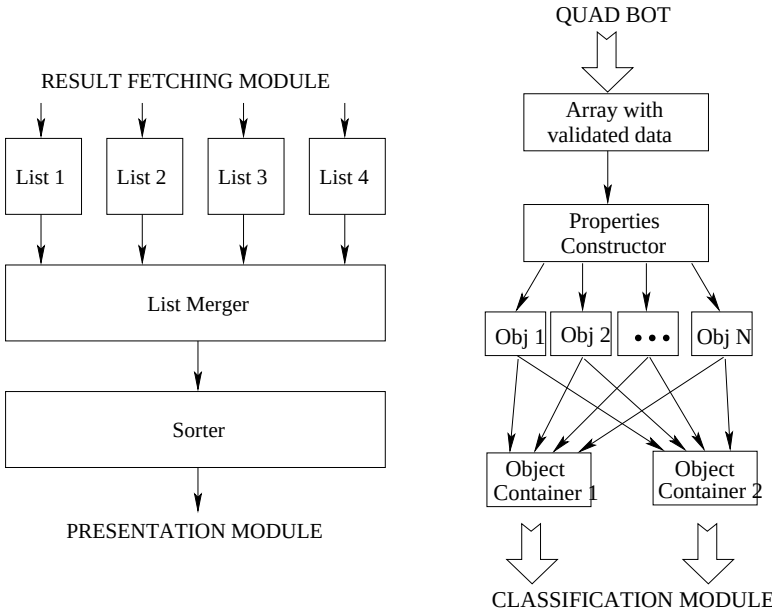


Figure 6.5: (Left) Quad Bot's structure. (Right) Object Builder's architecture.

Parameter Receiver/Validator. It accepts all the data coming from the database selector and the user. The validation process includes transformation of the inputs in a way that can be sent to the search engines. For example, the procedure removes all leading and trailing spaces from the query string and replaces all spaces by the character "+". The Result Validator also performs security checks to ensure that a "should-be" numeric parameter is really numeric, or that an attacking user does not send a malicious script instead of a query string. We developed this compartment by keeping in mind that all data coming from the Internet should be treated as suspicious. There are over a dozens of security checks that are performed for each parameter.

Query Dispatcher. The Query Dispatcher is the QuadBot's heart. It gets the validated data and creates http requests to the selected search engines. This is the slowest procedure of the whole system; its speed depends on the number of the invoked search engines, the requested results, the server's Internet connection, etc. We have accelerated this procedure by submitting all the requests to the search engines simultaneously. To achieve that, we had to employ the *libcurl* library with cURL (client URL) extensions 7.16.0 for PHP 5.1,

that support multiple connections at a time. By building the Query Dispatcher this way, we managed to shrink the idle time to no more than 1 or 2 seconds.

Result Collector. The Result Collector embraces the http responses transmitted by the search engines. Each involved search engine must respond to the Query Dispatcher's request, by sending the source code of its result page. The source code is being filtered by using pattern matching techniques. The module retrieves the ranking, the URL, the title and the snippet for each candidate. When it receives all the information, it stores it in temporary arrays and sends them to the next module for validation.

Result Validator. The Result Validator is the most complex compartment of this module, as it performs multiple conversions to the collected data. The URL validation part is responsible for the appropriate formatting of the collected URLs, so that the overlapping candidates could be correctly detected later. At first, a UTF-8 decoding function converts all UTF-8 encoded characters to their ISO equivalent. For example, the %27 set of characters is being converted to an opening single apostrophe character ('). At next, a formatting function trims the trailing slash from a URL (if exists), a third procedure checks if an engine has returned two identical URLs etc. As already mentioned above, most of these conversions will be presented later on.

6.6.1.3 Object builder

The Object Builder is a connecting bridge between the Quad Bot and the Classification module. We concluded that it is a good idea to treat our data as objects, so that we can use all the object-oriented programming features (such as inheritance, or support for multiple instances). This coding approach makes things very easy for the classification and presentation modules. In this section we describe how the collected results are being converted to objects. The Object Builder's architecture is depicted in the right part of Figure 6.5.

Array with validated data. The Object Builder's input is the array that the Quad Bot produces. It contains all the collected results that passed the Result Validator's checks.

Property Constructor. This module implements a class that describes the properties of our objects. The properties that are being assigned to the objects are the URL, the Title, the Abstract and one to four Rankings (depending on the number of the selected search engines).

Object Containers 1 and 2. The first advantage of using objects in the system's implementation comes almost immediately, as we are provided with the ability to create multiple copies (not just references) of the objects. In this compartment, all the objects (the results) are being transferred to two new, identical object containers. The results enter the containers in groups. The first group consists of the results that the first search engine returns, the

second group consists of the results that the second engine returns, etc. These containers will be the main tool in our effort to compare the search engine rankings and generate the final ranked list.

6.6.1.4 Classification module

The Classification Module accepts the two result containers from the Object Builder and performs the result ranking according to the selected ranking algorithm. Its architecture is illustrated in the left part of Figure 6.6.

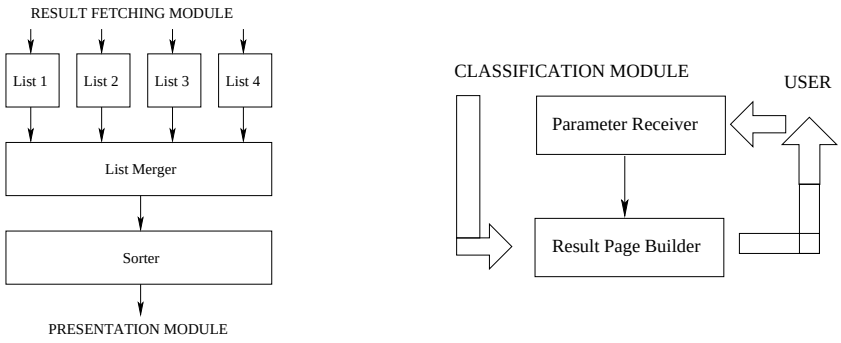


Figure 6.6: (Left) Classification Module. (Right) Presentation Module.

Overlapping Detector. This section is responsible for detecting the overlapping candidates and for creating the final candidate list. It receives input from the two object containers and compares each object from the first container, to all objects from the second container. When the URL properties of two objects are identical, this object is marked as overlapping. Finally, the procedure constructs one container that holds all candidates, overlapping or not. The overlapping candidates appear only once in this container.

Ranking Module. The Ranking Module accepts the candidate container that the Overlapping Detector constructs, but it also receives the ranking algorithm that the user selected. The Ranking Module will apply the QuadRank algorithm by default, unless the user selects another supported algorithm. Next, it computes the weight factors and/or the Borda Scores. Finally, it sorts the candidate list on ascending (for weight factors) or descending (for Borda Scores) order and passes the classified list to the Presentation Module.

6.6.1.5 Presentation module

The task of this module is to construct the result page that will be presented to the user. In comparison to the other system compartments, this one has the simplest architecture. In the right part of Figure 6.6, we illustrate a schematic diagram of its internal structure.

Parameter Receiver. In this section, a set of parameters and user preferences are being transferred to the module. These preferences may either derive from user's direct selections, or from a previously stored cookie. The innovative element here is the view selector. QuadSearch is capable of displaying the results with the classic way that a search engine displays its results, but also can present the results by using the array view. The array view will present a matrix that shows only the titles of the candidates and the rankings they received from each search engine. This feature has been developed because it offers an easier way to compare the candidates.

Result Page Builder. The Result Page Builder is a HTML code production factory. It accepts the ranked result list and the user preferences and constructs the source code of the result page from the scratch. Finally, the page is displayed to the user through the user interface.

6.6.2 Features

In this section, a brief list of QuadSearch's primary features is presented.

1. Classic/Array view switch. This feature has already been mentioned earlier. The user is able to view the results in the classic way, but can also select the array view that provides an easier way of comparing the collected results.

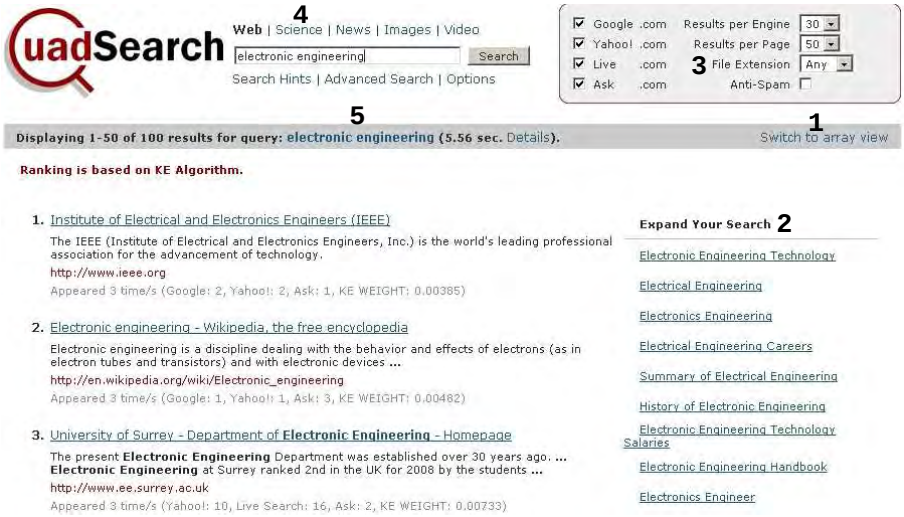


Figure 6.7: The results' page

2. Related searches. Apart from the desired results, the QuadBot is capable of grabbing almost everything from the results' pages that the exploited search engines transmit. In order to provide more specific results, the search engines prompt their users to submit the queries that they propose. The QuadBot can fetch these query strings and present it to the result page through the Presentation Module.

3. File type filter. Many users tend to search the Web for specific file types (e.g., Adobe Acrobat or Microsoft Word files) and QuadSearch includes a similar feature. The user can select one of the most popular file extensions and perform a Web search. At this time, the QuadSearch engine supports searches for the following file formats: PDF, DOC, XLS, PS, RTF and PPT.

4. Search for scientific articles. QuadSearch supports searches for scientific articles in the richest scientific databases. Google Scholar is also included in these databases. This type of search can be accessed from the "Science" link and will return papers, technical reports and books approved by the scientific community related to the query terms.

5. Query string explosion feature. This feature (see Figure 6.7) splits the query string to its search terms and gives the user the ability to perform 'single term' searches. For example, the query string 'electronic engineering' is being split to the terms 'electronic' and 'engineering'. By clicking on any of these words QuadSearch will perform a Web search.

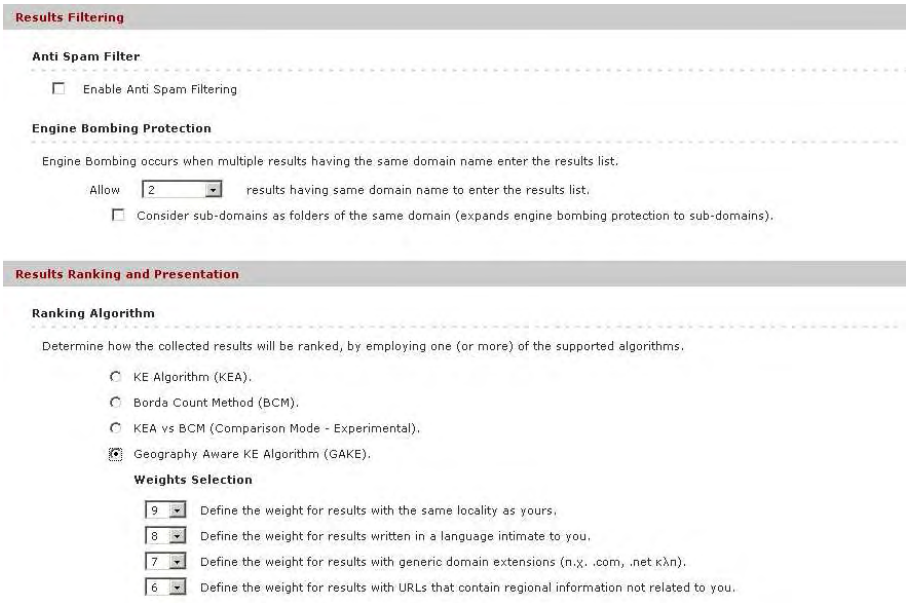


Figure 6.8: Part of the options' page is where the Ranking Algorithm Selector and the Engine Bombing Protection lay.

6. Ranking Algorithm Selector. This feature (see Figure 6.8) is only accessible from the options page and provides the user with the facility to determine how the collected results will be ranked, by employing one (or more) of the supported algorithms. At this time, QuadSearch supports our KE Algorithm and the Borda Count method. It also provides a third option that utilizes both algorithms and presents the results in array view (comparison mode). It is in our intentions to include more ranking algorithms in the system (e.g., Markov Chains).

7. Engine Bombing Protection. When various search resources are being exploited, a possibility that many similar results will enter the result's list always exists. This phenomenon is called *engine bombing*. For example, it is not very informative and useful for a user to submit a query and receive five or more results from the same domain in the top, say, twenty listing. Thus, we developed a feature (which can be enabled or disabled) to prevent multiple results coming from the same domain to enter into the result list; alternatively the user can select the maximum number of such results.

6.6.3 Efficiency Evaluation

In the previous subsection, we have demonstrated how the use of the proposed rank aggregation method can lead to better rankings, when compared to the rankings that the component engines produce. We have indicated that the term "better", concerns both Precision and quality of the returned results.

The second most important measure of a system's performance, is how quickly it responds to a given query. It may well produce qualitative results, but this could be close to useless, unless these results are produced in reasonable times.

Before we proceed to the evaluation of *QuadSearch*'s response, we discuss all the time penalties that any metasearch engine has to suffer, before it presents a list of results to the user. We also define *idle time*, as the total intermediate time between the moment the user submits a query and the moment that the engine presents the results. All idle times presented here, are expressed in seconds. In addition, henceforth, we will use the term *server*, to refer to the machine that hosts the search system.

The idle time is affected by numerous factors. Generally, we cannot exactly calculate it beforehand; we can only estimate a value, which is computed by using the following relationship:

$$t_{tot} = t_{req} + t_{res} + t_{ret} + t_{pr}, \tag{6.12}$$

where

- t_{req} represents the request time, that is, the time that the system needs to send the request to its component engines. It depends on the server's upload capabilities. This time is usually infinitesimal.

- t_{res} is the total time the metasearch engine has to wait for the component engines to respond. In other words, this is the time a single engine consumes to search its

index structure and apply its own ranking algorithm to form the results' list. As it is obvious, there is no technique that can be applied to improve this timing.

- t_{ret} signifies the results' pages downloading duration. This is the most unstable factor that affects a metasearch engine's response. It depends on the server's download capability, the number of concurrent users, the daytime and other parameters that are rigorous to manage. In general, improving the network capacity of the server, leads to smaller download delays.

- t_{pr} indicates the overhead added by the execution of the fusion and ranking algorithms of the metasearch engine. The application of additional filters (such as the anti-spam filter and engine bombing protection), further increases this time.

To evaluate the *QuadSearch*'s speed, we have formed a set of fifteen queries. During the tests we tried to identify some terms whose inverted lists were stored in the component engines' caches and some that were not. These queries have been submitted to *QuadSearch* with various parameters enabled and disabled, for all the proposed rank aggregation methods. We also exploit all of the four component engines available.

Query	t_{ret}	N	t_{pr}		
			Borda Count	Outranking	QuadRank
tokyo hotel	1.53	90	0.04	0.09	0.05
artificial fertilizers	1.41	87	0.04	0.08	0.05
free flv player	1.34	80	0.03	0.07	0.04
lewis hamilton	1.56	90	0.04	0.09	0.05
markov chains	1.38	79	0.03	0.07	0.04
greenhouse effect	1.30	77	0.02	0.07	0.04
h-index	1.42	87	0.04	0.08	0.04
marine biology	1.38	85	0.04	0.08	0.05
public key encryption	1.64	90	0.04	0.11	0.06
voice over ip	1.55	89	0.04	0.10	0.06
waterfall	1.29	94	0.04	0.10	0.04
scorched earth	1.34	85	0.04	0.08	0.05
apple	1.21	92	0.04	0.09	0.04
cold war	1.22	84	0.03	0.08	0.04
iran nuclear weapons	1.54	99	0.05	0.11	0.07
AVERAGE	1.41	87.2	0.037	0.086	0.048

Table 6.15: *QuadSearch* response times for various rank aggregation methods and 30 requested results per engine.

In Table 6.15 we record the time performance of various rank aggregation methods in the environment of QuadSearch for each query of our set, when the system requests $k = 30$

results from its component engines. The second column shows the retrieval times t_{ret}, that is, the time the metasearch engine needs to download the result lists from the component engines. In the third column we record the length of the list which derives from the fusion of the input rankings, whereas the three last columns contain the times each algorithm needs to rank the merged list.

Borda Count is the fastest algorithm and this is expected since the score of each document is computed by simply adding the individual rankings it received by the component engines. QuadRank is somehow slower (by a margin of about 23%) since the evaluation of the $Z(c)$ and $U(c)$ scores is slightly more expensive. The Outranking approach is by far the slowest method since each document must be compared with the rest $N-1$ documents in all 4 input rankings. Therefore, it is approximately 2.3 times slower than Borda Count and 1.8 times slower than QuadRank.

Notice that processing times for Borda Count and the Outranking Approach depend only on the number of documents of the merged list. Therefore, the more documents the merged list contains, the slower Borda Count and the Outranking Approach become. On the other hand, apart from the number of documents of the merged list, the timings of QuadRank also depend on the length of the submitted query due to the existence of zone weighting. Consequently, we expect from longer queries to be processed at lower rates.

Query	t_{ret}	N	t_{pr}		
			Borda Count	Outranking	QuadRank
tokyo hotel	2.01	276	0.23	0.82	0.31
artificial fertilizers	1.95	295	0.25	0.94	0.37
free flv player	1.94	259	0.21	0.74	0.35
lewis hamilton	2.13	277	0.22	0.85	0.30
markov chains	1.98	267	0.21	0.78	0.36
greenhouse effect	1.99	255	0.21	0.71	0.28
h-index	2.03	297	0.26	0.97	0.31
marine biology	2.13	269	0.22	0.80	0.29
public key encryption	2.07	292	0.25	0.93	0.38
voice over ip	2.22	296	0.26	0.96	0.57
waterfall	2.25	306	0.26	1.02	0.33
scorched earth	1.99	278	0.22	0.85	0.32
apple	1.91	274	0.22	0.82	0.26
cold war	2.07	286	0.24	0.88	0.33
iran nuclear weapons	2.40	314	0.27	1.08	0.40
AVERAGE	2.07	281.6	0.24	0.87	0.34

Table 6.16: *QuadSearch* response times for various rank aggregation methods and 100 requested results per engine.

In Table 6.16 we repeat the experiment by requesting top-100 lists. Apparently, in this case the average retrieval time increases significantly as more time is required to download larger lists. Furthermore, the fusion and ranking duration increases, since now we have to process more documents. Notice that while QuadRank remains slower than Borda Count by a relatively stable percentage of 24%, the performance of the Outranking Approach degrades considerably faster (2.5 times slower than QuadRank and 3.6 times slower than Borda Count).

6.7 Conclusions

In this chapter we discussed the problem of rank aggregation and we studied the current state-of-the-art approaches for solving the problem. We introduced QuadRank, a method which is especially designed for aggregating results collected from Web search engines (component engines), that is, it primarily concerns Web metasearch engines. QuadRank takes into consideration multiple attributes such as the individual rankings assigned by the component engines, the number of appearances of an item in the result lists of the component engines, and the physical location (zone or field) of a document where a query term occurs. QuadRank also performs a URL analysis and compares the locality of the page to the locality of the user and assigns scores accordingly.

QuadRank was exhaustively attested by using data from the 2009 Text Retrieval Conference. Moreover, we used results from our own test queries; in both experimental phases we concluded that QuadRank outperformed Borda Count, the Outranking Approach and the Condorcet method by a significant margin. Furthermore, we examined another family of rank aggregation methods, the KE family. The member functions of KE can be seen as the ancestors of QuadRank.

All the proposed methods have been implemented within QuadSearch, a prototype metasearch engine which have been developed as a testbed for designing new rank aggregation methods and general solutions for the wider problem of metasearching.

CHAPTER VII

Conclusions

In this chapter we conclude this dissertation and we report the most important results of the research conducted here.

In chapter II we studied issues regarding the improvement of the query throughput of Web search engines. Query throughput is a metric which reflects the rates at which a search engine responds to the incoming requests in a time unit. It is an extremely important measure of the efficiency, since the modern Web search engines now accept user queries at rates touching 10 thousands per second. Such tremendous workloads dictate that all the underlying subsystems of a search engine must be fully optimized.

Our effort was towards the proposal of efficient methods for the organization and the compression of the data stored within the inverted index. The inverted index is the most important data structure maintained by search engines in order to generate ranked results in response to user queries. They are mainly composed i) by the lexicon, a list which contains all the terms appearing in the document collection, and ii) the inverted file which stores for each term of the lexicon, one list (the inverted list) with the representations (or postings) of all the term occurrences in the document collection.

According to the recent literature, the most robust algorithms for compressing that data stored in the inverted index are those which are capable of encoding entire bundles of integers, such as PForDelta [69], and VSEncoding [129]. These methods combine satisfactory compression effectiveness with high decompression speed and are considered ideal for encoding the document identifiers (docIDs) and the frequency values of the postings of an inverted list. Nevertheless, regarding the positional data accompanying each posting they are not equally useful because during query processing it is required that we decode only limited data; consequently, the decompression of entire blocks of integers is apparently redundant. Furthermore, these methods do not support direct access to the desired data and they introduce a costly look-up operation.

To address these drawbacks we introduce *PFBC* (Positions Fixed Bit Compression) [9], a method which encodes the positions of an inverted list block by using a fixed number of bits. PFBC also dictates that we store within the skip table one additional pointer value for each block of an inverted list; this pointer shows the exact location of the starting point of the compressed positions. We show that such an organization allows direct access to the positional data, since we are able to compute the location of the required data instead of looking up for it. Moreover, we decode only the information actually needed. The experiments on various experimental index setups demonstrated that PFBC:

- Decodes 7–11 times fewer data than PForDelta and VSEncoding.
- Decodes the positional data required to process a query 5 times faster than the adversary approaches.
- Accesses the desired data instantly and no costly look-ups are required.
- Does not require the existence of an additional positions look-up data structure.
- Requires much fewer pointers than the other approaches such as indexed lists.
- Introduces a compression loss of about 3%, resulting in slightly increased index sizes. However, this cost is amortized by the smaller size of the accompanying data structures.

We also investigated strategies of including additional information within the inverted index by considering the structure of the Web documents. This structure is expressed by the usage of zones (or fields) and in chapter II we discussed how this additional information can be integrated in the index. We replaced the plain positional information by the notion of the *term occurrence*, an enriched representation which apart from the positional value, also stores the identifier of the zone where the term is located within the document.

Almost immediately, a new problem was created: how can we encode the additional information without suffering prohibitively large index sizes and without decreasing query throughput? To address this problem we were based on the main ideas behind PBFC. We introduced *TZP* [16], a method for compressing the aforementioned word occurrences. TZP dictates that each document zone is assigned a unique zoneID value and that each zoneID is accompanied by the corresponding positional value. This position-zoneID pair is initially encoded in a 32-bit space by reserving the z_b most significant bits for the zoneID. In the sequel, these encoded representations are again encoded by employing a fixed number of bits, in a fashion similar to that of PFBC. Due to this strategy TZP exhibits the same

advantages over the competitor compression algorithms, as PFBC. The experimental evaluation of TZP showed that the enriched inverted index which includes zone information is only 11–13% larger than a plain positional index, and it introduces no additional time costs during the evaluation of a query.

Chapter III was dedicated to the identification of influential bloggers in community blog sites. Since this is a relatively new problem, there was only one competitor approach: the influence-flow model of [5]. In this chapter we exposed the main drawbacks of this model and we introduced MEIBI and MEIBIX, two new metrics which eliminate these drawbacks. In short, the benefits of these two methods are:

- They are *time-sensitive*: Blogosphere, is extremely volatile; millions of blog posts are produced in short periods of time, new bloggers enter the community whereas others leave it at rapid rates. Therefore, a robust influence evaluation metric should not overlook this crucial parameter. MEIBI assigns influence scores to the bloggers by taking into consideration the elapsed time since the publication of his/her articles. On the other hand, MEIBIX considers an article influential if the article receives references in the present, regardless of its publication date. In contrast to the preliminary influence-flow model, both metrics identify the bloggers who are *presently* influential.

- They take into consideration the *productivity* of a blogger: The existing approach assigns scores to the bloggers by accounting only the best post, however, this is apparently inadequate; a blogger who publishes only a few very influential posts in log periods of time cannot be considered more influential than a frequent writer whose posts are equally influential.

- They are *simple*: The ι-scores of the influence-flow model are defined in a recursive manner similar to PageRank. This requires multiple iterations before the system converges and it could lead to stability problems due to the negative influence assigned to the outgoing links. On the other hand, the proposed approaches are based on simple algebraic scores which can be computed directly.

- They do not require *fine-tuning*: In contrast to the influence-flow model, MEIBI and MEIBIX do not include user-defined parameters.

Nevertheless, the plain identification of influential bloggers may not be adequate since the matter of productivity is equally important. For this reason we posed the issue of the bloggers classification not by assigning a single evaluation metric, but by determining a

point on a two dimensional productivity-influence plane. We introduced a new metric, *BP/BI-index*, an indicator which is used to classify the bloggers of a community into one of the following predefined classes: $\mathcal{A}$ (contains bloggers who are both productive and influential), $\mathcal{B}$ (contains those who are influential but not productive), $\mathcal{C}$ (the opposite of $\mathcal{B}$, consists of productive but not influential bloggers), and $\mathcal{D}$ (nor productive, nor influential) [14].

The proposed MEIBI, MEIBIX and BP/BI-index where attested by using data crawled from real-world community blog sites. The experiments demonstrated the usefulness of our methods by providing more sensible and effective blogger rankings and classifications.

In chapter IV we dealt with the problem of improving the retrieval effectiveness in Web search engines and the blog vertical search engines. Regarding the former, we examined the usefulness of combining robust scoring schemes such as term proximity and zone weighting into a single ranking function. We introduced BM25TOPF [16], a scheme which is based on the field structure of the Web documents discussed in chapter II, and requires that both zone and positional information is stored within the index. The main characteristics of BM25TOPF are summarized into the following list:

- It takes advantage of the enriched index representation of chapter II, which stores both positional and zone data within the inverted lists' postings.
- It combines the essence of BM25F which rewards the words appearing in specific locations (such as the title, or the URL) with the spirit of BM25TP which rewards the documents having the query terms in a close proximity.
- It takes into account the correct term ordering, that is, it boosts the documents which have the query terms in the same order as they appear in the submitted query.

The retrieval effectiveness of BM25TOPF was experimentally measured by using a set of 50 queries from the Adhoc task of the TREC 2009 conference. The document collection we employed was the ClueWeb09-B dataset, a large repository comprised of about 50 million Web pages, whereas the underlying index structure was of the form of chapter II. Our method outperformed BM25F, the most effective competitor function, by a margin of approximately 6%.

The second contribution of chapter IV concerns the opinionated document retrieval in blog search engines. The problem is of remarkable importance, since a high-quality opinion expressed by an influential blogger affects a large number of users and its impact is large. As mentioned in chapter IV, a positive opinion about a product can significantly increase its commercial success whereas in contrast, multiple negative statements about a

politician can decrease his/her publicity and affect the success of his/her political career. Similar examples include artists, events, travel locations, service providers, and generally every judgeable aspect of life.

Until now, the opinionated retrieval models did not consider the issue of the objective quality of a blog post. In other words, the retrieved documents were scored by accounting only their relevance to the given query, and the grade of their "opinionatedness". In this work we highlighted that similarly to the traditional Web retrieval, it is essential that we develop special query-independent metrics which indicate the overall importance of a blog entry in combination with the other two criteria. We introduced the *QUIQS (Query-Independent Quality Metrics)*, a set of indicators which determine the importance of a blog post according to the reputation of their author, the importance of the hosting blog site, and several query-independent statistics (i.e. number of incoming references, number of comments, word length etc). After the introduction of QUIQS, we developed a model which assigns scores to the blog posts by linearly combining its IR-score, its opinion score, and the value of QUIQS.

The experimental evaluation of this model was performed by using 20 queries from the Blog Track of the TREC 2009 conference, against Blogs08, a large repository of 28 million blog entries. Compared to the other state-of-the-art approaches which did not consider objective quality metrics, our model exhibited better MAP by about 7.5%.

In chapter V me presented numerous solutions to problems which are related to the academic search engines, digital libraries and scientific databases. Initially we studied the most important scientometrics, that is, indicators which evaluate the research work of a scientist. We introduced the notion of coterminal citations and we presented them as a form of self-citations and co-citation. Based on this spirit, we introduced the f-index, a new metric for quantifying the impact of coterminal citations in scientific networks.

The current academic search engines and digital libraries have recently included the scientometrics in the automated (or semi-automated) profile pages of the authors. Since until recently the sizes of these systems was small the problem of computing these values was of no special interest. However, the quick growth of their repositories[1] in combination with the increased users' interest have rendered such large-scale problems particularly challenging. In chapter V we presented four strategies for computing the most popular scientometric indicators in parallel by using Hadoop/MapReduce, a fault-tolerant framework for solving problems in large scales [8]. The experimental efficiency measurements of these strategies was performed on two different clusters by using the CiteSeerX dataset, a 30 GB

[1]Google Scholar's repository consists of hundreds of millions of scientific documents, whereas as of April 2013, Microsoft Academic contained approximately 40 million articles and 19 million author profiles.

collection of research articles. These experiments revealed that the usage of in-Mapper Combiners improves running times by about 35%, whereas it reduces network bandwidth by 43%.

In the sequel, we introduced the problem of identifying the most attractive research areas for new scientists [15]. Although this problem may seem similar to the generic determination of the emerging research areas, the one we study here is different, since there are emerging fields of science which are not suitable for starters. We defined the primary characteristics of the starters and we referred to their lack of experience and trust, two factors which may render some research topics "hostile" to them. The model we proposed is based on a number of features which combine these characteristics to the attributes of the emerging research areas. In the sequel, we experimentally demonstrated the effectiveness of this model and we concluded that not all popular or emerging research fields are suitable for new scientists.

The fourth contribution of chapter V concerns a popular data mining task, that of document classification. In our study we were mainly interested in proposing a robust method for classifying research articles, a problem which is slightly different than the one of standard document classification. Although these problems are different, their usefulness is equally high; an effective research article classification method can be exploited to develop robust search tools such as searches for relevant items, suggestions of similar papers and scientists, query expansion methods and several others.

In this chapter we also introduced a supervised machine learning (ML) algorithm [10]. Our method is based on a predefined set of labels and a set of papers (training set) which have already been assigned one or more labels from the label set. It extracts the keywords, authors, journals, and co-authorship information from the elements of the training set and constructs a data model comprised of three relevance description vectors. Each of these vectors stores some information which indicates and quantifies the relationship of each keyword, author/co-author, and journal to each candidate label. In the sequel, we apply this model in combination with a scoring function (i.e. inverse document frequency, logarithmic, etc) to assign labels to the elements of the testing set. The evaluation of our algorithm was performed by using the cross validation rule and the CiteSeerX dataset. We compared it against the two state-of-the-art ML algorithms, the Support Vector Machines and AdaBoost.MH; our algorithm was steadily more accurate than both of its adversaries, whereas AdaBoost.MH failed to complete some tests due to its enormous memory consumption.

Finally, in chapter VI we dealt with the generic problem of rank aggregation and with a specialization of it, the ranking fusion in Web metasearch engines. The issue in question

concerns the determination of effective algorithms in order to merge and rearrange ranked lists of results deriving from a number of information sources. Here we introduced a family of such methods, comprised of the *KE* algorithm, and a number of variants which takes into consideration additional parameters [17, 11]. The basic KE methodology introduces a scoring formula which is based on several statistics such as the items' individual rankings, the number and the length of the component lists', and the total number of each item appearances. We also introduced a set of variants with the aim of improving the effectiveness of KE in the context of metasearching.

Based on the strong points of the KE family, we defined another rank aggregation method, namely *QuadRank* [13] which also takes into consideration multiple attributes such as the individual rankings assigned by the component engines, the number of appearances of an item in the result lists of the component engines, and the physical location (zone or field) of a document where a query term occurs. QuadRank also performs a URL analysis and compares the locality of the page to the locality of the user and assigns scores accordingly.

QuadRank was exhaustively attested by using data from the 2009 Text Retrieval Conference. Moreover, we used results from our own test queries; in both experimental phases we concluded that QuadRank outperformed Borda Count, the Outranking Approach and the Condorcet method by a significant margin. Furthermore, we examined another family of rank aggregation methods, the KE family. The member functions of KE can be seen as the ancestors of QuadRank.

All the proposed methods have been implemented within *QuadSearch* [17], a prototype metasearch engine which have been developed as a testbed for designing new rank aggregation methods and general solutions for the wider problem of metasearching. An operational implementation of QuadSearch can be accessed under http://quadsearch.csd.auth.gr.

APPENDICES

APPENDIX A

Refereed Publications

A.1 Publications in journals and transactions

1. Leonidas Akritidis, Panayiotis Bozanis. Improving Opinionated Blog Retrieval Effectiveness with Quality Measures and Temporal Features, WWW Journal, Springer, to appear, June 2013.

2. Leonidas Akritidis, Dimitrios Katsaros, Panayiotis Bozanis. Identifying Attractive Research Fields for New Scientists, Scientometrics, Springer, vol. 91, no. 3, pp. 869-894, March 2012.

3. Leonidas Akritidis, Dimitrios Katsaros, Panayiotis Bozanis. Improved Retrieval Effectiveness by Efficient Combination of Term Proximity and Zone Scoring: A Simulation-based Evaluation, Simulation Modelling: Practice And Theory, Elsevier, vol. 22, no. 3, pp. 74-91, March, 2012.

4. Leonidas Akritidis, Dimitrios Katsaros, Panayiotis Bozanis. Identifying the Productive and Influential Bloggers in a Community, IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews, vol. 41, no 5, pp. 759-764, September 2011.

5. Leonidas Akritidis, Dimitrios Katsaros, Panayiotis Bozanis. Effective Rank Aggregation for Metasearching, Journal of Systems and Software, Elsevier, vol. 84, no 1, pp 130-143, January 2011.

6. Dimitrios Katsaros, Leonidas Akritidis, Panayiotis Bozanis. The f-index: Quantifying the Impact of Coterminal Citations in Scientists Ranking, Journal of the American Society for Information Science and Technology (Wiley), vol. 60, no. 5, pp. 1051-1056, May 2009.

A.2 Publications in international conferences

1. Leonidas Akritidis, Panayiotis Bozanis. A Supervised Machine Learning Classification Algorithm for Research Articles, In Proceedings of the 28th ACM Symposium on Applied Computing (SAC 2013), Coimbra, Portugal, accepted, March 18-22, 2013.

2. Leonidas Akritidis, Panayiotis Bozanis. Computing Scientometrics in Large-Scale Academic Search Engines with MapReduce, In Proceedings of the 13th International Conference on Web Information System Engineering (WISE 2012), Paphos, Cyprus, Lecture Notes in Computer Science (LLNCS), vol. 7651, pp. 609-623, November 28-30, 2012.

3. Leonidas Akritidis, Panayiotis Bozanis. Positional Data Organization and Compression in Web Inverted Indexes, In Proceedings of the 23rd International Conference on Database and Expert Systems Applications (DEXA 2012), Vienna, Austria, Lecture Notes in Computer Science (LLNCS), vol. 7446, pp. 422-429, September 3-7, 2012.

4. Leonidas Akritidis, Dimitrios Katsaros, Panayiotis Bozanis. Identifying Influential Bloggers: Time Does Matter, In Proceedings of the 2009 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2009), IEEE Press, Milan, Italy, pp. 76-83, September 15-18, 2009.

5. Leonidas Akritidis, Dimitrios Katsaros, Panayiotis Bozanis. Effective Ranking Fusion Methods for Personalized Metasearch Engines, In Proceedings of the 12th Panhellenic Conference on Informatics (PCI 2008), IEEE Press, Samos, Greece, pp. 39-43, August 28-30, 2008.

A.3 Chapters in books

1. Leonidas Akritidis, Dimitrios Katsaros, Panayiotis Bozanis. Modern Web Technologies, Chapter in New Directions in Web Data Management, (Athena Vakali & Lakhmi C. Jain, eds.), Springer, 2011.

A.4 Publications in national conferences

1. Leonidas Akritidis, George Voutsakelis, Dimitrios Katsaros, Panayiotis Bozanis. Quad-Search: A novel metasearch engine, In Proceedings of the 11th Panhellenic Conference on Informatics (PCI 2007), Patras, Greece, pp. 453-466, May 18-20, 2007.

A.5 Technical reports

1. Leonidas Akritidis, Dimitrios Katsaros, Panayiotis Bozanis. Identifying Influential Bloggers: Time Does Matter, Technical Report, May, 2009.

2. Dimitrios Katsaros, Leonidas Akritidis, Panayiotis Bozanis. Spam: It's Not Just for Inboxes and Search Engines! Making Hirsch h-index Robust to Scientospam, Technical Report, January, 2008.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel Abadi, Avi Silberschatz, and Alexander Rasin. Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads. *Proceedings of the VLDB Endowment*, 2(1):922–933, 2009.

[2] Jonathan Adams. The use of bibliometrics to measure research quality in uk higher education institutions. *Archivum immunologiae et therapiae experimentalis*, 57(1):19–32, 2009.

[3] Robert Adler, John Ewing, and Peter Taylor. Joint committee on quantitative assessment of research: citation statistics. *Australian Mathematical Society Gazette*, 35(3):166–88, 2008.

[4] Nitin Agarwal and Huan Liu. Blogosphere: Research issues, tools and applications. *ACM SIGKDD Explorations*, 10(1):18–31, 2008.

[5] Nitin Agarwal, Huan Liu, Lei Tang, and Philip S. Yu. Identifying the influential bloggers in a community. In *Proceedings of the 2008 international conference on web search and data mining*, pages 207–218, 2008.

[6] Nir Ailon. Aggregation of partial rankings, p-ratings and top-m lists. *Algorithmica*, 57(2):284–300, 2010.

[7] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM*, 55(5):23, 2008.

[8] Leonidas Akritidis and Panayiotis Bozanis. Computing scientometrics in large-scale academic search engines with mapreduce. *Web Information Systems Engineering-WISE 2012*, pages 609–623, 2012.

[9] Leonidas Akritidis and Panayiotis Bozanis. Positional data organization and compression in web inverted indexes. In *Proceedings of the 23rd International Conference on Database and Expert Systems Applications*, pages 422–429, 2012.

[10] Leonidas Akritidis and Panayiotis Bozanis. A supervised machine learning classification algorithm for research articles. In *Proceedings of the 28th ACM Symposium on Applied Computing*, pages 115–120, 2013.

[11] Leonidas Akritidis, Dimitrios Katsaros, and Panayiotis Bozanis. Effective ranking fusion methods for personalized metasearch engines. In *Proceedings of the 2008 Panhellenic Conference on Informatics*, pages 39–43, 2008.

[12] Leonidas Akritidis, Dimitrios Katsaros, and Panayiotis Bozanis. Identifying influential bloggers: Time does matter. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies*, pages 76–83, 2009.

[13] Leonidas Akritidis, Dimitrios Katsaros, and Panayiotis Bozanis. Effective rank aggregation for metasearching. *Journal of Systems and Software*, 84(1):130–143, 2011.

[14] Leonidas Akritidis, Dimitrios Katsaros, and Panayiotis Bozanis. Identifying the productive and influential bloggers in a community. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 41(5):759–764, 2011.

[15] Leonidas Akritidis, Dimitrios Katsaros, and Panayiotis Bozanis. Identifying attractive research fields for new scientists. *Scientometrics*, 91(3):869–894, 2012.

[16] Leonidas Akritidis, Dimitrios Katsaros, and Panayiotis Bozanis. Improved retrieval effectiveness by efficient combination of term proximity and zone scoring: A simulation-based evaluation. *Simulation Modelling Practice and Theory*, 22(3):74–91, 2012.

[17] Leonidas Akritidis, Georgios Voutsakelis, Dimitrios Katsaros, and Panayiotis Bozanis. Quadsearch: A novel metasearch engine. *Current Trends in Informatics*, pages 453–466, 2007.

[18] Jeff Allen. Comparison of metasearch engines. Technical report, 2009. Southern Methodist University, CSE8337, available at http://jdadesign.net/wp-content/uploads/2010/01/HW2b.pdf.

[19] Vo Ngoc Anh and Alistair Moffat. Structured index organizations for high-throughput text querying. *String Processing and Information Retrieval*, pages 304–315, 2006.

[20] Vo Ngoc Anh and Alistair Moffat. Index compression using 64-bit words. *Software: Practice and Experience*, 40(2):131–147, 2010.

[21] Javed A. Aslam and Mark Montague. Models of metasearch. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 276–284, 2001.

[22] Michael G. Banks. An extension of the hirsch index: Indexing scientific topics and compounds. *Scientometrics*, 69(1):161–168, 2006.

[23] MM Sufyan Beg and Nesar Ahmad. Soft computing techniques for rank aggregation on the world wide web. *World Wide Web Journal*, 6(1):5–22, 2003.

[24] Pratyush Bharati and Peter Tarasewich. Global perceptions of journals publishing e-commerce research. *Communications of the ACM*, 45(5):21–26, 2002.

[25] Mustafa Bilgic, Galileo Mark Namata, and Lise Getoor. Combining collective classification and link prediction. In *Proceedings of Workshop on Mining Graphs and Complex Structures at the IEEE International Conference on Data Mining*, pages 381–386, 2007.

[26] Paolo Boldi and Sebastiano Vigna. Compressed perfect embedded skip lists for quick inverted index lookups. *String Processing and Information Retrieval*, pages 25–28, 2005.

[27] Lutz Bornmann and Hans-Dieter Daniel. Does the h-index for ranking of scientists really work? *Scientometrics*, 65(3):391–392, 2005.

[28] Dhruba Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 11:21, 2007.

[29] Tibor Braun, Wolfgang Glänzel, and András Schubert. A hirsch-type index for journals. *Scientometrics*, 69(1):169–173, 2006.

[30] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1):107–117, 1998.

[31] Stefan Büttcher, Charles LA Clarke, and Brad Lushman. Term proximity scoring for ad-hoc retrieval on very large text collections. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 621–622, 2006.

[32] Flavio Chierichetti, Ravi Kumar, and Prabhakar Raghavan. Compressed web indexes. In *Proceedings of the 18th International Conference on World Wide Web*, pages 451–460, 2009.

[33] CiteSeerX. Citeseerx data. *http://csxstatic.ist.psu.edu/about/data*, 2013.

[34] Charles L. Clarke, Nick Craswell, and Ian Soboroff. Overview of the trec 2009 web track. In *Proceedings of TREC 2009*, 2009.

[35] Don Coppersmith, Lisa Fleischer, and Atri Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 776–782, 2006.

[36] Blaise Cronin. Hyperauthorship: A postmodern perversion or evidence of a structural shift in scholarly communication practices? *Journal of the American Society for Information Science and Technology*, 52(7):558–569, 2001.

[37] Blaise Cronin. Scholarly communication and epistemic cultures. *New review of academic librarianship*, 9(1):1–24, 2003.

[38] Blaise Cronin, Debora Shaw, and Kathryn La Barre. A cast of thousands: Coauthorship and subauthorship collaboration in the 20th century as manifested in the scholarly journal literature of psychology and philosophy. *Journal of the American Society for Information Science and Technology*, 54(9):855–871, 2003.

[39] Kushal Dave, Steve Lawrence, and David M. Pennock. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *Proceedings of the 12th International Conference on World Wide Web*, pages 519–528, 2003.

[40] Jeffrey Dean. Challenges in building large-scale information retrieval systems: invited talk. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 1–1, 2009.

[41] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[42] Robert P DeConde, Sarah Hawley, Seth Falcon, Nigel Clegg, Beatrice Knudsen, and Ruth Etzioni. Combining results of microarray experiments: A rank aggregation approach. *Statistical Applications in Genetics and Molecular Biology*, 5(1):1–23, 2006.

[43] Shuai Ding, Jinru He, Hao Yan, and Torsten Suel. Using graphics processors for high performance IR query processing. In *Proceedings of the 18th international conference on World wide web*, pages 421–430, 2009.

[44] Ying Ding, Gobinda G. Chowdhury, and Schubert Foo. Bibliometric cartography of information retrieval research by using co-word analysis. *Information Processing & Management*, 37(6):817–842, 2001.

[45] Cynthia Dwork, Ravi Kumar, Moni Naor, and Dandapani Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th international conference on World Wide Web)*, pages 613–622, 2001.

[46] Leo Egghe. Theory and practise of the g-index. *Scientometrics*, 69(1):131–152, 2006.

[47] Leo Egghe. Dynamic h-index: the hirsch index in function of time. *Journal of the American Society for Information Science and Technology*, 58(3):452–454, 2007.

[48] Jonathan L. Elsas, Jaime Arguello, Jamie Callan, and Jaime G. Carbonell. Retrieval and feedback models for blog feed search. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 347–354, 2008.

[49] Tamer Elsayed, Jimmy Lin, and Douglas W. Oard. Pairwise document similarity in large collections with mapreduce. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies*, pages 265–268, 2008.

[50] Andrea Esuli and Fabrizio Sebastiani. Sentiwordnet: A publicly available lexical resource for opinion mining. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation*, volume 6, pages 417–422, 2006.

[51] Ronald Fagin, Ravi Kumar, and Dandapani Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 301–312, 2003.

[52] Mohamed Farah and Daniel Vanderpooten. An outranking approach for rank aggregation in information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 591–598, 2007.

[53] James H. Fowler and Dag W. Aksnes. Does self-citation pay? *Scientometrics*, 72(3):427–437, 2007.

[54] Norbert Fuhr. *A probabilistic model of dictionary based automatic indexing*. Techn. Hochsch. Darmstadt, Fachbereich Informatik, 1985.

[55] Eugene Garfield. Can citation indexing be automated. In *Statistical association methods for mechanized documentation, symposium proceedings*, pages 189–192, 1965.

[56] Eugene Garfield. Citation analysis as a tool in journal evaluation. *American Association for the Advancement of Science*, 178(4060):471–479, 1972.

[57] Eugene Garfield. The application of citation indexing to journals management. *Current contents*, 33:3–5, 1994.

[58] Shima Gerani, Mark James Carman, and Fabio Crestani. Proximity-based opinion retrieval. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 403–410, 2010.

[59] Lise Getoor. Link-based classification. *Advanced Methods for Knowledge Discovery from Complex Data*, pages 189–207, 2005.

[60] Sanjay Ghemawat and Jeffrey Dean. Mapreduce: simplified data processing on large clusters. In *Symposium on Operating System Design and Implementation*, pages 137–150, 2004.

[61] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, 2003.

[62] Kathy E. Gill. How can we measure the influence of the Blogosphere? In *Proceedings of the Workshop on the Weblogging Ecosystems: Aggregation, Analysis and Dynamics*, 2004.

[63] Daniel Gruhl, Ramanathan Guha, Ravi Kumar, Jasmine Novak, and Andrew Tomkins. The predictive power of online chatter. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 78–87, 2005.

[64] Daniel Gruhl, Ramanathan Guha, David Liben-Nowell, and Andrew Tomkins. Information diffusion through blogspace. *Proceedings of the 13th international conference on World Wide Web*, pages 491–501, 2004.

[65] Zoltán Gyöngyi and Hector Garcia-Molina. Link spam alliances. In *Proceedings of the 31st international conference on Very large data bases*, pages 517–528, 2005.

[66] Zoltan Gyongyi and Hector Garcia-Molina. Spam: It's not just for inboxes anymore. *Computer*, 38(10):28–34, 2005.

[67] Ben He, Craig Macdonald, and Iadh Ounis. Ranking opinionated blog posts using OpinionFinder. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 727–728, 2008.

[68] Iina Hellsten, Renaud Lambiotte, Andrea Scharnhorst, and Marcel Ausloos. Self-citations, co-authorships and keywords: A new approach to scientists" field mobility? *Scientometrics*, 72(3):469–486, 2007.

[69] Sandor Heman. Super-scalar database compression between RAM and CPU cache. *Master's Thesis. University of Amsterdam. Amsterdam, The Netherlands*, 2005.

[70] Jorge E. Hirsch. An index to quantify an individual's scientific research output. *Proceedings of the National Academy of Sciences*, 102(46):16569–16572, 2005.

[71] Ken Hyland. Self-citation and self-reference: Credibility and promotion in academic publication. *Journal of the American Society for Information Science and Technology*, 54(3):251–259, 2003.

[72] Akshay Java, Pranam Kolari, Tim Finin, and Tim Oates. Modeling the spread of influence on the Blogosphere. In *Proceedings of the 15th international world wide web conference*, pages 22–26, 2006.

[73] Thorsten Joachims. *Text categorization with support vector machines: Learning with many relevant features*. Springer, 1998.

[74] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the International Conference on Machine Learning*, pages 200–209, 1999.

[75] Pairin Katerattanakul, Bernard Han, and Soongoo Hong. Objective quality ranking of computing journals. *Communications of the ACM*, 46(10):111–114, 2003.

[76] Dimitrios Katsaros, Leonidas Akritidis, and Panayiotis Bozanis. The f index: Quantifying the impact of coterminal citations on scientists' ranking. *Journal of the American Society for Information Science and Technology*, 60(5):1051–1056, 2009.

[77] Edward Keller and Jonathan Berry. *One American in ten tells the other nine how to vote, where to eat and, what to buy. They are The Influentials*. The Free Press, 2003.

[78] Apostolos Kritikopoulos, Martha Sideri, and Iraklis Varlamis. BlogRank: Ranking Weblogs based on connectivity and similarity features. In *Proceedings of the 2nd international workshop on Advanced architectures and algorithms for internet delivery and applications*, pages 8–16, 2006.

[79] Amy N. Langville and Carl D. Meyer. *Google's PageRank and beyond: The science of search engine rankings*. Princeton University Press, 2006.

[80] Stephen M. Lawani. On the heterogeneity and classification of author self-citations. *Journal of the American Society for Information Science*, 33(5):281–284, 1982.

[81] Dik L. Lee, Huei Chuang, and Kent Seamons. Document ranking and the vector-space model. *IEEE software*, 14(2):67–75, 1997.

[82] Yeha Lee, Seung-Hoon Na, Jungi Kim, Sand-Hyob Nam, Hun-young Jng, and Jong-Hyeok Lee. Kle at trec 2008 blog track: Blog post and feed retrieval. In *Proccedings of TREC 2008*, 2008.

[83] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429, 2007.

[84] Jimmy Lin. Scalable language processing algorithms for the masses: A case study in computing word co-occurrence matrices with mapreduce. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 419–428, 2008.

[85] Jimmy Lin and Chris Dyer. Data-intensive text processing with mapreduce. *Synthesis Lectures on Human Language Technologies*, 3(1):1–177, 2010.

[86] Yu-Ru Lin, Hari Sundaram, Yun Chi, Jun Tatemura, and Belle Tseng. Discovery of blog communities based on mutual awareness. In *Proceedings of the 3rd Annual Workshop on the Weblogging Ecosystem*, 2006.

[87] Yu-Ting Liu, Tie-Yan Liu, Tao Qin, Zhi-Ming Ma, and Hang Li. Supervised rank aggregation. In *Proceedings of the 16th international conference on World Wide Web*, pages 481–490, 2007.

[88] Paul Lowry, Denton Romans, and Aaron Curtis. Global journal prestige and supporting disciplines: A scientometric study of information systems journals. *Journal of the Association for Information Systems*, 5(2):29–80, 2004.

[89] Wei Lu, Stephen Robertson, and Andrew MacFarlane. Field-weighted xml retrieval based on bm25. *Advances in XML Information Retrieval and Evaluation*, pages 161–171, 2006.

[90] Sofus A. Macskassy and Foster Provost. Classification in networked data: A toolkit and a univariate case study. *The Journal of Machine Learning Research*, 8:935–983, 2007.

[91] Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146, 2010.

[92] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge University Press, 2008.

[93] Richard McCreadie, Craig Macdonald, and Iadh Ounis. On single-pass indexing with mapreduce. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 742–743, 2009.

[94] Weiyi Meng, Clement Yu, and King-Lup Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89, 2002.

[95] Donald Metzler and W. Bruce Croft. A markov random field model for term dependencies. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 472–479, 2005.

[96] Alistair Moffat and Justin Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379, 1996.

[97] Mark Montague and Javed A. Aslam. Metasearch consistency. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 386–387, 2001.

[98] Tony Mullen and Nigel Collier. Sentiment analysis using support vector machines with diverse information sources. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, volume 4, pages 412–418, 2004.

[99] Seung-Hoon Na, Yeha Lee, Sang-Hyob Nam, and Jong-Hyeok Lee. Improving opinion retrieval based on query-specific sentiment lexicon. *Advances in Information Retrieval*, pages 734–738, 2009.

[100] Gonzalo Navarro, Edleno Silva De Moura, Marden Neubert, Nivio Ziviani, and Ricardo Baeza-Yates. Adding compression to block addressing inverted indexes. *Information Retrieval*, 3(1):49–77, 2000.

[101] E. CM. Noyons, HF. Moed, and A. FJ. Van Raan. Integrating research performance analysis and science mapping. *Scientometrics*, 46(3):591–604, 1999.

[102] Ryosuke L. Ohniwa, Aiko Hibino, and Kunio Takeyasu. Trends in research foci in life science fields over the last 30 years monitored by emerging topics. *Scientometrics*, 85(1):111–127, 2010.

[103] I. Ounis, M. De Rijke, C. Macdonald, and G. Mishne. Overview of the trec 2006 blog track. In *Proceedings of TREC 2006*, 2006.

[104] Iadh Ounis, Craig Macdonald, and Ian Soboroff. Overview of the trec 2007 blog track. In *Proceedings of TREC 2007*, 2007.

[105] Iadh Ounis, Craig Macdonald, and Ian Soboroff. Overview of the trec 2008 blog track. In *Proccedings of TREC 2008*, 2008.

[106] B. Uygar Oztekin, George Karypis, and Vipin Kumar. Expert agreement and content based reranking in a meta search environment using mearf. In *Proceedings of the 11th international conference on World Wide Web*, pages 333–344, 2002.

[107] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 79–86, 2002.

[108] David Lorge Parnas. Stop the numbers game. *Communications of the ACM*, 50(11):19–21, 2007.

[109] Olle Persson, Wolfgang Glänzel, and Rickard Danell. Inflationary bibliometric values: The role of scientific collaboration and the need for relative indicators in evaluative studies. *Scientometrics*, 60(3):421–432, 2004.

[110] Xiaoguang Qi and Brian D. Davison. Web page classification: Features and algorithms. *ACM Computing Surveys*, 41(2):12, 2009.

[111] Vijay V. Raghavan and SKM Wong. A critical analysis of the vector space model for information retrieval. *Journal of the American Society for Information Science and Technology*, 37(5):279–287, 1986.

[112] R. Kelly Rainer Jr. and Mark D. Miller. Examining differences across journal rankings. *Communications of the ACM*, 48(2):94, 2005.

[113] Yves Rasolofo and Jacques Savoy. Term proximity scoring for keyword-based retrieval systems. In *Advances in information retrieval*, pages 207–218, 2003.

[114] M Elena Renda and Umberto Straccia. Web metasearch: Rank vs score based rank aggregation methods. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, pages 841–846, 2003.

[115] Stephen E. Robertson and K. Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science and Technology*, 27(3):129–146, 1976.

[116] Donald G. Saari. The mathematics of voting: Democratic symmetry. *Economist*, page 83, 2000.

[117] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):620, 1975.

[118] Khalid Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 2000.

[119] Ralf Schenkel, Andreas Broschart, Seungwon Hwang, Martin Theobald, and Gerhard Weikum. Efficient text proximity search. *String Processing and Information Retrieval*, pages 287–299, 2007.

[120] Michael Schreiber. Self-citation corrections for the hirsch index. *EPL (Europhysics Letters)*, 78(3):30002, 2007.

[121] András Schubert, Wolfgang Glänzel, and Bart Thijs. The weight of author self-citations. a fractional approach to self-citation counting. *Scientometrics*, 67(3):503–514, 2006.

[122] D. Sculley. Rank aggregation for similar items. In *Proceedings of the Seventh SIAM International Conference on Data Mining*, 2007.

[123] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[124] Milad Shokouhi. Segmentation of search engine results for effective data-fusion. *Advances in Information Retrieval*, pages 185–197, 2007.

[125] Antonis Sidiropoulos, Dimitrios Katsaros, and Yannis Manolopoulos. Generalized Hirsch h-index for disclosing latent facts in citation networks. *Scientometrics*, 72(2):253–280, 2007.

[126] Antonis Sidiropoulos and Yannis Manolopoulos. A citation-based system to assist prize awarding. *ACM SIGMOD Record*, 34(4):60, 2005.

[127] Antonis Sidiropoulos and Yannis Manolopoulos. A new perspective to automatically rank scientific conferences using digital libraries. *Information Processing & Management*, 41(2):289–312, 2005.

[128] Antonis Sidiropoulos and Yannis Manolopoulos. Generalized comparison of graph-based ranking algorithms for publications and authors. *Journal of Systems and Software*, 79(12):1679–1700, 2006.

[129] Fabrizio Silvestri and Rossano Venturini. VSEncoding: Efficient coding and fast decoding of integer lists via dynamic programming. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1219–1228, 2010.

[130] Henry Small. Tracking and predicting growth areas in science. *Scientometrics*, 68(3):595–610, 2006.

[131] Stefanos Souldatos, Theodore Dalamagas, and Timos Sellis. Sailing the web with captain nemo: A personalized metasearch engine. In *Proceedings of the ICML workshop: Learning in Web Search*, Bonn, Germany, 2005.

[132] Amanda Spink, Bernard J Jansen, Vinish Kathuria, and Sherry Koshman. Overlap among major web search engines. *Internet Research*, 16(4):419–426, 2006.

[133] Atsushi Sugiura and Oren Etzioni. Query routing for web search engines: Architecture and experiments. *Computer Networks*, 33(1–6):417–429, 2000.

[134] Pang-Ning Tan and Rong Jin. Ordering patterns by combining opinions from multiple sources. In *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 695–700, 2004.

[135] Mohammad A. Tayebi, S. Mehdi Hashemi, and Ali Mohades. B2rank: An algorithm for ranking blogs based on behavioral features. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 104–107, 2007.

[136] Frederik Transier and Peter Sanders. Engineering basic algorithms of an in-memory text search engine. *ACM Transactions on Information Systems*, 29(1):1–36, 2010.

[137] Yuen-Hsien Tseng, Yu-I Lin, Yi-Yang Lee, Wen-Chi Hung, and Chun-Hsiang Lee. A comparison of methods for detecting hot topics. *Scientometrics*, 81(1):73–90, 2009.

[138] Peter Turney and Michael L. Littman. Measuring praise and criticism: Inference of semantic orientation from association. *ACM Transactions on Information Systems*, 21(4):315–346, 2003.

[139] Peter D. Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 417–424, 2002.

[140] Howard Turtle and James Flood. Query evaluation: strategies and optimizations. *Information Processing & Management*, 31(6):831–850, 1995.

[141] S. Phineas Upham and Henry Small. Emerging research fronts in science and technology: patterns of new knowledge development. *Scientometrics*, 83(1):15–38, 2010.

[142] Anthony van Raan. Self-citation as an impact-reinforcing mechanism in the science system. *Journal of the American Society for Information Science and Technology*, 59(10):1631–1643, 2008.

[143] Olga Vechtomova. Facet-based opinion retrieval from blogs. *Information Processing & Management*, 46(1):71–88, 2010.

[144] Christopher C. Vogt. *Adaptive combination of evidence for information retrieval*. PhD thesis, University of California at San Diego, 1999.

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

[146] Wikipedia. The Hirsch h-index, Jan. 2009. Available from http://en.wikipedia.org/wiki/H-index.

[147] Ian Witten, Alistair Moffat, and Timothy Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, 1999.

[148] Hao Yan, Shuai Ding, and Torsten Suel. Compressing term positions in web indexes. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 147–154, 2009.

[149] Hao Yan, Shuai Ding, and Torsten Suel. Inverted index compression and query processing with optimized document ordering. In *Proceedings of the 18th international conference on World wide web*, pages 401–410, 2009.

[150] Hung-Chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D. Stott Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1029–1040, 2007.

[151] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1):69–90, 1999.

[152] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the International Conference on Machine Learning*, pages 412–420, 1997.

[153] H Peyton Young and Arthur Levenglick. A consistent extension of condorcet's election principle. *SIAM Journal on Applied Mathematics*, 35(2):285–300, 1978.

[154] Jiangong Zhang, Xiaohui Long, and Torsten Suel. Performance of compressed inverted list caching in search engines. In *Proceedings of the 17th international conference on World Wide Web*, pages 387–396, 2008.

[155] Min Zhang and Xingyao Ye. A generation model to unify topic relevance and lexicon-based sentiment for opinion retrieval. In *Proceedings of the 31st International ACM SIGIR Conference on Research and development in Information Retrieval*, pages 411–418, 2008.

[156] Wei Zhang, Clement Yu, and Weiyi Meng. Opinion retrieval from blogs. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management*, pages 831–840, 2007.

[157] Ying Zhou and Joseph Davis. Community discovery and analysis in Blogspace. In *Proceedings of the 15th international conference on World Wide Web*, pages 1017–1018, 2006.

[158] Ji Zhu, Saharon Rosset, Hui Zou, and Trevor Hastie. Multi-class adaboost. *Ann Arbor*, 1001(48109):1612, 2006.

[159] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2), 2006.

[160] Marcin Zukowski, Sandor Heman, Niels Nes, and Peter Boncz. Super-scalar RAM-CPU cache compression. In *Proceedings of the 22nd International Conference on Data Engineering*, page 59, 2006.