A HOLISTIC, SIMILARITY-BASED APPROACH FOR PERSONALIZED RANKING IN WEB DATABASES

by

ADITYA TELANG

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
August 2011

To my parents who set the example and who made me who I am.

ACKNOWLEDGEMENTS

I would like to express sincere gratitude to my advisor Sharma Chakravarthy for constantly motivating and encouraging me as well as providing valuable guidance and support during the course of my doctoral work. More importantly, his keen interest and enthusiasm in teaching and his pedagogical methods truly inspired me. It is the same type of enthusiasm that he devoted into advising that helped in making me a true researcher. He has spent a tremendous amount of time training me in research, writing, and giving professional talks. He has been brilliant and insightful in research discussions. His dedication, persistence, and sincerity in research deeply impressed me and have set up the high standards that I want to maintain in my future career.

I am also extremely grateful to Chengkai Li. It has been a great experience to collaborate with him during the last three years. Besides, as a recent graduate who has made a successful start to his career, he also becomes an immediate role model for me.

I want to thank my thesis committee members Gautam Das and Leonidas Fegaras for their interest in my research, for taking time to serve in my dissertation committee and for their comments, suggestions, guidance and help at the time of need. I would also like to thank Ramez Elmasri and Bahram Khalili for all their support, encouragement and guidance during the years I have been in UTA as a graduate teaching assistant and an assistant instructor. In addition, I would like to acknowledge the assistance I received, from Pamela McBride as well as the entire staff of the Computer Science and Engineering Department at UTA, during the course of my doctoral work.

I wish to thank all my colleagues (past and present) at Information Technology Lab (ITLab) for their support and encouragement and for making the stay at ITLab over the last five years thoroughly exciting. I am also grateful to my friends for their interest in my research and for their helpful discussions and invaluable comments. My appreciation especially goes to Rohan Prabhu, Jaydeep Inamdar, Apurv Dhadphale, Rushikesh Khasgiwale, Akshay Joshi, Kailas Inamdar and Vaibhav Landage for providing me with generous support and countless fun throughout the five years at UTA. I am grateful to all the professors and mentors who have helped me throughout my career, both in India and United States.

Finally, I am grateful to my parents Pratibha and Dilip Telang, my late uncle Pradeep Telang as well as other family members for their endless love and support. Without their encouragement and endurance, this work would not have been possible. Last but not the least, I express my appreciation for the goodness of an overriding Providence in my life.

July 6, 2011

ABSTRACT

A HOLISTIC, SIMILARITY-BASED APPROACH FOR PERSONALIZED RANKING IN WEB DATABASES

ADITYA TELANG, Ph.D.

The University of Texas at Arlington, 2011

Supervising Professor: Sharma Chakravarthy, Chengkai Li

With the advent of the Web, the notion of “information retrieval” has acquired a completely new connotation and currently encompasses several disciplines ranging from traditional forms of text and data retrieval in unstructured and structured repositories to retrieval of static and dynamic information from the contents of the surface and deep Web. From the point of view of the end user, a common thread that binds all these areas is to support appropriate alternatives for allowing users to specify their intent (i.e., the user input) and displaying the resulting output ranked in an order relevant to the users.

In the context of specifying an user’s intent, the paradigms of *querying* as well as *searching* have served well, as the staple mechanisms in the process of information retrieval over structured and unstructured repositories. Processing queries over known, structured repositories (e.g., traditional and Web databases) has been well-understood, and search has become ubiquitous when it comes to unstructured repositories (e.g., document collections and the surface Web). Furthermore, searching structured repositories has also been explored to a limited extent. However, there is not much work in querying unstructured sources which, we believe is the next step in performing focused retrievals.

Correspondingly, one of the contributions of this dissertation is a novel semantic-guided approach, termed Query-By-Keywords (or QBK), to generate queries from search-like inputs for unstructured repositories. Instead of burdening the user with schema details, this approach utilizes pre-discovered semantic information in the form of – taxonomies, relationship of keywords based on context, and attribute & operator compatibility amongst Web sources, to generate query skeletons that are subsequently transformed into queries. Additionally, progressive feedback from users is used to further improve the accuracy of these query skeletons. The overall focus thus, is to propose an alternative paradigm for the generation of queries on unstructured repositories using as little information from the user as possible.

Irrespective of the template for intent specification (i.e., either a search or a query request), the number of results typically returned in response to such intents are, often, extremely large. This is particularly true in the context of the deep Web where a large number of results are returned for queries on Web databases and choosing the most useful answer(s) becomes a tedious and time-consuming task. Most of the time the user is not interested in **all** answers; instead s/he would prefer those results, that are ranked based on her/his interests, characteristics, and past usage, to be displayed before the rest. Furthermore, these preferences vary as users and queries change.

Accordingly, in this dissertation, we propose a novel *similarity*-based framework for supporting user- and query-dependent ranking of query results in Web databases. This framework is based on the intuition that – for the results of a given query, similar users display comparable ranking preferences, and a user displays similar ranking preferences over results of analogous queries. Fittingly, this framework is supported by two novel and comprehensive models of – 1) Query Similarity, and 2) User Similarity, proposed as part of this work. In addition, this ranking framework relies on the availability of a small yet representative set of ranking functions collected across several user-query pairs, in order to rank the results of a given user query at runtime. Appropriately, we address the subsequent problem i.e., establishing a relevant *workload* of ranking functions that assists the similarity model in the best possible way to achieve the goal of user- and query-dependent ranking. Furthermore, we advance a novel *probabilistic learning model* that infers individual ranking functions (for this workload) based on the implicit browsing behavior displayed by users. We establish the effectiveness of this holistic ranking framework by experimentally evaluating it on Google Base's *vehicle* and *real estate* databases with the aid of Amazon's Mechanical Turk users.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Today, the term “information retrieval” has assumed a completely different connotation and encompasses several disciplines such as – text retrieval from *unstructured* document repositories, data retrieval from *structured* database systems, and retrieval of information from the Web in the form of – static Web pages from the *surface* Web and individual data records from the Web databases spread over the *deep* Web. From the point of view of the end user, a common thread that binds all these areas is to support appropriate alternatives for allowing user input and displaying the result output. Specifically, since both these aspects are significant from the point of view of utility of the system, almost all retrieval frameworks need to concentrate on concocting –

1. An appropriate mechanism for the users to specify their desired intent in a clear and simple manner, and
2. A suitable technique for presenting a subset of the most relevant results (amongst all the results retrieved in response to the specified intent) to the user.

1.1 Specification Of User Intent

The paradigms of *query* and *search* have served well, as the staple mechanisms for specifying user intent in the process of information retrieval over structured and unstructured repositories. A *search* request, typically expressed in the form of one or more keywords has been extensively used for retrieving –

- Textual data from large document collections using various techniques [1] [2] [3] [4],

- Individual Web pages from the surface Web using retrieval mechanisms namely *search engines* such as Google (`www.google.com`), *meta-search engines* like Clusty (`www.clusty.com`), and *faceted search engines* for instance, DBLP ((`www.dblp.l3s.de`),
- Natural language extracts from Question-Answering frameworks like START [5] and Yahoo! Answers (`www.answers.yahoo.com`), and
- Data records from structured repositories representing traditional databases (using techniques such as DBExplorer [6] and BANKS [7]) as well as Web databases like NetFlix (`www.netflix.com`), Travelocity (`www.travelocity.com`), Google Base (`www.base.google.com`) and so on.

Given the lack of a learning curve associated in expressing one's intent as a set of keywords, the *search* paradigm has become extremely popular across a large class of users and is extensively used by almost all retrieval frameworks.

In contrast, a *query*, conventionally specified using a language such as SQL [8] (or its variant such as OOQL [9], SemQL [10] and others) has been extensively used for retrieving data records from traditional database systems. Although precise and expressive in nature, specifying one's intent through a query language requires prior knowledge of the data sources, the data model, the operators of the data model, and the syntax and semantics of the query language. As a result, querying is the proclivity of those who are willing to spend time learning the intricacies of correct specification.

The advent of the *deep* Web [11] [12] led to the proliferation of a large number of Web databases and their corresponding applications (such as airline reservations on Travelocity (`www.travelocity.com`), vehicle search on Yahoo! Autos (`www.autos.yahoo.com`), and real estate scouting on Realtor (`www.realtor.com`). Due to the difficulties in specifying one's intent through a query language, most Web applications resorted to the use of template-based or form-based menus (loosely adapted from the Query-

By-Example (or QBE) paradigm [13] [14]). The users express their intent by entering keywords in the different field of these menus, and the system internally translates this input into a (SQL or similar) query for further processing.

In spite of the advances in the use of templates and menus as well as the enhancements to the expressive power of query languages, the notion of *querying* (unlike *search*) has been restricted to only structured repositories i.e., traditional and Web databases. It can be argued, based on the following example, that expanding the scope of the *query* paradigm to encompass both structured and unstructured repositories is the next step in performing focused retrievals.

Motivating Example-1: *Consider a business executive (U_1) who is on a vacation to United Kingdom. Let us presume that s/he is interested in seeking answers to the intent:* "**Determine a list of castles near London that can be reached in 2 hours by train**".

Although all the information for answering the above request is available on the Web, it is currently ***not possible*** *to frame it as a* ***single*** *query/search and get meaningful results. The current process for identifying answers to the above query essentially involves the following* ***manual*** *steps: i) get a list of cities in UK having castles from a static Web source (e.g., Google search on "castles in UK" followed by a link traversal), ii) check train connectivity with a specific city by using a deep Web source that provides train schedules in UK (e.g., Google search on "train schedules in UK" followed by a query on the deep Web application such as London Rail (`www.tfl.gov.uk`)), and iii) retrieve the time for the travel and the amount of time at the destination. In addition, if multiple cities/castles satisfy the request, they can be ranked based upon some meaningful criteria, such as age of the castle, amount of time available at the destination, and so on.*

Thus, the answer to the above request can be **manually** put together **with considerable effort** by searching/querying for different pieces of information from diverse Web sources, keeping relevant information, and combining the information in a meaningful/intelligent manner. Since the above, and similar such requests (illustrated in Chapter 2) span across several types of Web sources (i.e., both static pages on the surface Web and the data hidden behind forms on the deep Web), and require combining and presenting an integrated view of the information (instead of simple list of URLs pointing to Web pages) as output, search is not likely to be a preferred alternative.

Although there exist a number of systems, such as HAVASU [15], WHIRL [16], MetaQuerier [17], Ariadne [18] and DBLife [19], that allow queries to be expressed (in the form of filling keywords in templates or menus) across several sources, most of the techniques, to date, are restricted only to databases associated with deep Web sources in a single domain or a set of pre-defined domains[1]. In contrast, techniques proposed for querying unstructured repositories have taken the approach of converting the unstructured data into a structured form and then querying the structured data (e.g., DBPedia [20]). Thus, to the best of our knowledge, formulating queries spanning several types of sources (in terms of static Web pages as well as dynamic Web applications) spread across multiple domains has not received sufficient attention.

1.1.1 Querying The Web Using Keywords

One of the goals of this dissertation is to address the problems associated with specifying an intent (such as the one in *Example-1*) that spans multiple Web sources in several domains in order to obtain chunks of relevant information from each source, and combine them to form an answer. Specifically, we propose a technique termed as **Query-By-Keywords** [21] [22] (or QBK) for formulating a complete query from search-like keywords input that can be processed over sources in multiple domains to retrieve useful answers.

[1]The notion of a *domain* is subjective. In the context of this dissertation, a *domain* indicates a collection of Web sources providing specific information associated with individual concepts such as travel, books, literature, and so on.

As most of the users are familiar and comfortable with the concept of searching, our approach starts with a set of keywords as input and expands it to a complete query using different types of semantic information and minimal user feedback/interaction. The *semantic* knowledge-base employed by this approach involves – taxonomies (for context information such as travel in the above example), attributes associated with concept nodes in the taxonomy, their types, and whether they can participate in join, spatial or temporal conditions, alternative meanings of words as they appear in multiple taxonomies, compatibility of attributes across concept meanings, dictionary meaning and priority of word semantics from the dictionary to the extent possible, and finally user feedback on past interactions. For instance, in the case of the *Example-1*, one user may provide his input as a set of the following keywords: "`castle, train, London`" in any order. For the same query, another user may input: "`train, 2 hours, London, castle`". The semantic information (that is assumed to be separately discovered and collected for this purpose) will assist in formulating the correct complete query with minimal interactions with the user. We elaborate on the details of this Query-By-Keywords approach in Chapter 2.

1.2 Relevant Ranking Of Retrieved Information

Whether a search or a query, the number of results obtained in response to an intent is, often, extremely large. Most of the time the user is not interested in **all** answers; instead s/he would prefer those results, that are ranked based on her/his interests, characteristics, and past usage, to be displayed before the rest. Hence, *ranking* emerged as an important aspect of information retrieval since before the Internet. Its utility has been recognized for traditional databases as well.

In the field of retrieval from unstructured collections (such as traditional document repositories or the surface Web), *intent-specific* ranking (based on the similarity between the input keywords with the resulting documents/pages) has been extensively studied. The emergence of the vector-space model [23], the probabilistic model [24] and other derivatives [25] of these models have been elaborately used by a number of frameworks. Furthermore, sophisticated algorithms such as PageRank [26] and SimRank[27] have been specifically developed to rank the large number of Web pages returned in response to a search request on the surface Web. In addition, the emergence of recommender (i.e., collaborative [28] [29] [30] [31] as well as content filtering [32] [33] [34]) and user-personalization systems [35] [36] led to the rapid development of techniques to support *user-dependent* ranking. Currently, most applications built over the surface Web (e.g., Amazon `www.amazon.com`), aim towards utilizing the proposed models listed above for providing a combined i.e., *intent-* as well as *user-specific* ranking of the results.

In contrast, a database system follows a Boolean model wherein every tuple returned in response to a query has an equal significance to the conditions specified in the query. Consequently, in traditional databases, the concept of ranking was restricted to simply, a user-specified *ordering* of the tuples based on the values of a single (or multiple) attributes. Although acceptable in traditional systems, the emergence of the deep Web exposed significant drawbacks associated with this *ordering* mechanism adopted in database systems. For instance, when a large number of results are returned for queries on Web databases that span a small number of the schema attributes, choosing the most useful answer(s) by browsing through this large result set becomes a tedious and time-consuming task.

Currently, Web applications built over these databases simplify this task by adopting the existing *ordering* mechanism in databases and display the query results sorted on the values of a **single** popular attribute (such as Price, Rating, etc.). However, as the following example illustrates, most Web users would prefer to see only a *relevant* set of results derived using multiple attribute values, instead of viewing *all* retrieved results displayed in a pre-determined order. In addition these preferences tend to vary as users and their queries change over time.

Motivating Example-2: *Consider Google Base's* Vehicle *database that comprises of a table with attributes Make, Price, Mileage, Location, Color, and so on, where each tuple represents a vehicle for sale.*

*Two users – a business executive (U_1) and a student (U_2), seek answers to the same query (Q_1): "***Make = Honda AND Location = Dallas, TX***". In response to this query, more than 18,000 tuples are returned. Intuitively, U_1 would typically search for* ***new*** *vehicles with specific* ***color*** *choices (e.g., only* ***red*** *colored vehicles), and hence would prefer vehicles with "*Condition = New AND Color = Red*" to be ranked and displayed higher than the others. In contrast, U_2 would most likely search for* ***used*** *vehicles* ***priced*** *under a specific amount (e.g., "Price $<$ 5,000\$"); hence, for U_2, the results should be ordered such that vehicles with "*Condition = Used AND Price $<$ 5,000\$*" are displayed before the rest.*

*The same student user (U_2) now moves to Mountain View, CA as an intern with Google and asks a different query: "***Make = Pontiac AND Location = Mountain View, CA***". We can presume (since he has procured an internship) that he may be willing to pay a slightly higher* ***price*** *for a lesser* ***mileage*** *vehicle (e.g., "Mileage $<$ 100,000"), although he may still be searching for* ***used*** *vehicles. His preferences for this query would then require vehicles with "*Condition = Used AND Price $<$ 10,000\$ AND Mileage $<$ 100,000*" to be ranked higher than others.*

The above scenario illustrates that different Web users may have contrasting ranking preferences over the results of the same query. It further emphasizes that the same user can display different ranking preferences over the results of different queries (at different points in time). Thus, it is evident that in the setting of Web databases, where a large set of queries given by varied classes of users is involved, the corresponding results are likely to be better received when ranked and displayed in a *user*- and *query*-**dependent** manner.

In order to support such an integrated ranking scheme over Web databases, ranking preferences/functions for every possible user-query pair need to be acquired *a priori* (in order to rank the results at query time) – an impossible task. Certain extensions to SQL [37] [38] [39] [40] [41] have allowed the *manual* specification of ranking preferences (or functions) at query time; however, this approach is baffling for **most** Web users.

Although *automated* ranking of query results has been proposed for Web (as well as some traditional) databases, current techniques either perform only *query-dependent* ranking [42] [43] [44] [45] based on the frequency of attribute values in the database or query logs, or support only *user*-dependent ranking by building extensive user profiles [46] [36] and/or requiring users to individually order all tuples within the database [47] [48].

1.2.1 A Holistic Approach For Ranking In The Deep Web

The primary contribution of this dissertation is the advancement of a holistic framework for enabling *user*- and *query*-dependent ranking of query results in Web databases. Since obtaining the ranking preferences, in advance, for every user over the results of each query is inconceivable, we put forward a novel **Similarity Model** [49] [50] for supporting such a ranking scheme. The intuition behind this model is: for the results of a given query, similar users display comparable ranking preferences, and a user displays analogous ranking preferences over results of similar queries. Consequently, this ranking framework is supported by two novel and comprehensive models, namely – i) **Query Similarity**, and ii) **User Similarity**, that we formalize as part of this work.

The notion of *query similarity* is based on a formal model [51] that computes similarity between any given pair of arbitrary Select-Project-Join (SPJ) queries. A straightforward approach for defining such similarity would be to compare the results (tuples) generated for the given queries; we term this the *query-result* similarity. However, this definition forces the computation of results at the time of query submission and keeping track of the similarity for a large number of queries requires that the results of each query be stored. More importantly, this approach does not make use of the query components (i.e., selections, joins, projections) associated with a query, which we believe can be beneficially used for determining similarity between two queries. Therefore, we advance an alternative approach – *query-condition* similarity that compares query components along with the meta information that can be extracted *a priori* from the database.

On the other hand, the metric of *user similarity* is formally based on a combined model that encompasses – a *static*, and a *dynamic* user similarity model. The former establishes similarity between a pair of users by comparing the information in the profiles associated with these users, and is based on the current paradigm of user similarity adopted by collaborative filtering systems [28] [29] [30] [31] as well as others [46] [36] in the domain of ranking. In contrast, the latter model departs from this notion of static user similarity, and instead, determines similarity by analyzing the relationship between the browsing choices adopted by the users. As the browsing behavior of users vary over time, their corresponding similarities will also vary; thus, rendering this model to be dynamic in nature. The combined model, thus ensures, a consistent computation of similarities between any pair of users by using – the *static* component if browsing choices are limited and/or not available, and the *dynamic* component if elaborate browsing choices are present.

In addition to the *similarity* model, another important component of the ranking framework is a *workload* of ranking functions collected across several user-query pairs, where each function represents an individual user's preferences towards the results of a specific query. At the time of answering a query for which no prior ranking function exists, the *similarity* model can ensure a **good** quality of ranking only if a ranking function for a very similar user-query pair exists in this *workload*.

Accordingly, we address the problem of determining an appropriate set of user-query pairs to form a **Workload** [52] [53] of ranking functions. Specifically, we present a novel metric, termed *workload goodness*, that quantifies the notion of a “good” workload into an absolute value. The process of finding such a workload of optimal goodness is a combinatorially explosive problem; therefore, we propose a heuristic solution, and advance three approaches for determining an acceptable workload.

The *workload* employed by the *similarity* model, in turn represents a number of ranking functions (each of which depict the preferences of a specific user towards the results of a distinct query). Capturing these user preferences over query results becomes a pertinent challenge. Unlike relational databases, the nature of Web database applications allow users to browse and select the results that match their preferences (through an interaction with the Web pages containing the result records). Hence, although an user’s explicit ranking preferences over the results of a query are not available, we believe that the results chosen by an user aid in *implicitly* indicating his/her ranking preferences. However, these preferences need to be somehow translated into formal ranking functions that can be subsequently employed by the *similarity* model. In this work, we cast the task of translating user preferences into a formal ranking function as a *learning* problem, and propose a novel technique called **Probabilistic Data Distribution Difference** [49] [50] for the same.

Finally, we establish the effectiveness of this holistic ranking framework and demonstrate its applicability to real Web databases (in terms of efficiency) by presenting the results of an extensive experimental evaluation performed with the aid of Amazon’s Mechanical Turk (`www.mturk.com`) users and Google Base’s (`www.base.google.com`) *vehicle* and *real estate* databases.

1.3 Contributions

The overall goal of this dissertation is to build a general framework for processing an user intent spread across several domains on the Web and representing the returned pieces

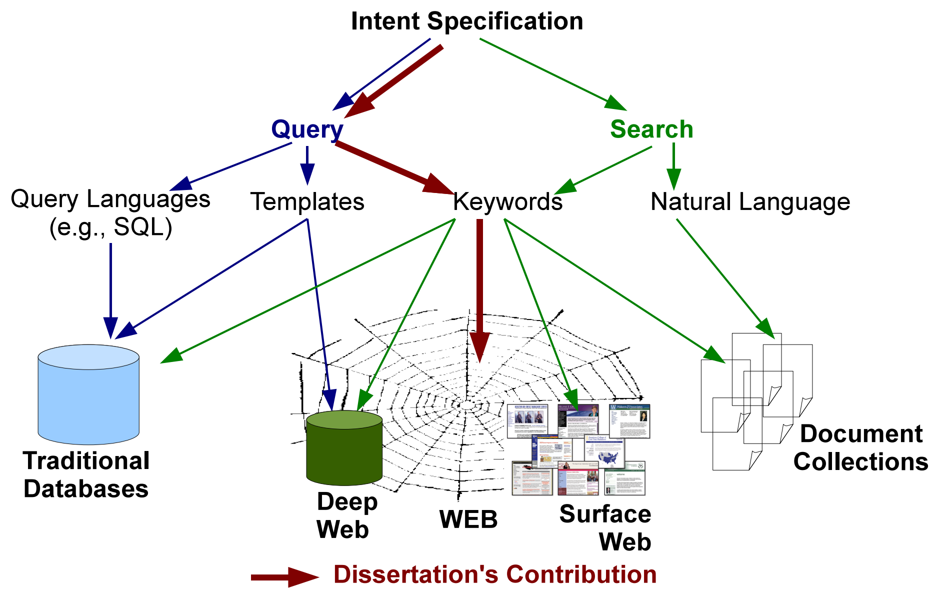


Figure 1.1. Querying The Web: Current Scene And Our Contribution.

of data in a user- and intent-specific order. Specifically, it should allow the users to express their desired intent in an appropriate and easy-to-understand manner. The underlying framework should then be able to – i) translate the intent into a query, ii) process this query over a multitude of disparate Web sources in an efficient manner, iii) integrate the corresponding results from different sources in an elegant manner, and iv) display the results ranked in a user- and query-dependent environment. It is obvious, that the problem is difficult to address in its entirety in a single dissertation. Hence, we limit the scope of our work to the following two topics:

Querying The Web: We formalize a *Query-By-Keywords* approach [21] [22] that translates an user intent expressed as a collection of keywords, to a structured query, that can be processed over disparate Web sources.

Ranking On The *Deep* Web: As part of this work, we present a *similarity*-based ranking framework [49] [50] for addressing the task of performing user- and query-dependent ranking over Web databases. Specifically, we advance two elaborate models – *Query*

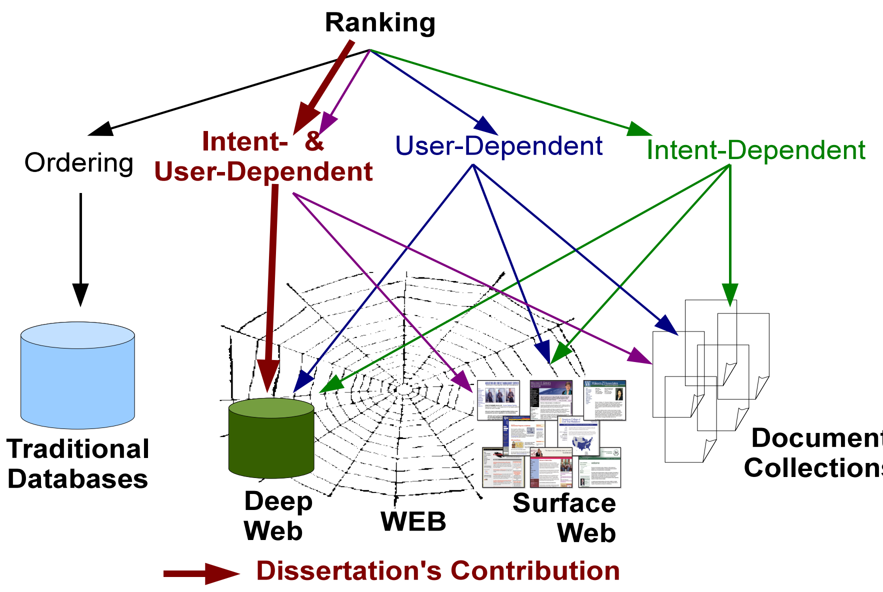


Figure 1.2. Ranking On The Deep Web: Current Scene And Our Contribution.

Similarity and *User Similarity*, and integrate them into a unified framework for performing subsequent ranking. Further, we formalize a novel approach for forming an appropriate *workload* [52] [53] of ranking functions, and present a novel *learning* technique [49] [50] to infer individual ranking functions based on the browsing choices of an user over the results of a query. In addition, we extend the models of *user* and *query similarity* into a generic framework [51] for establishing similarity between any pair of users (new and/or experienced) as well as any pair of arbitrary SPJ queries that can be employed by other applications (besides ranking).

The salient contributions of this dissertation to the areas of intent specification and ranking are further illustrated in Figures 1.1 and 1.2. As discussed in Sections 1.1 and 1.2 respectively, and highlighted by the figures, *querying* the Web for focussed retrieval of information, and ranking in a *user-* and *query-dependent* manner on the deep Web has not received sufficient attention. We hope this dissertation contributes in spawning new threads in these areas of research towards building the next generation of retrieval frameworks on the Web.

1.4 Roadmap

The rest of the dissertation is organized as follows:

- Chapter 2 presents the Query-By-Keywords approach for intent specification over unstructured sources.
- In Chapter 3, we elaborate the details of the *similarity-based* ranking framework, present the details of a *learning* technique for deriving individual ranking functions from implicit user behavior, and discuss the results of the experimental evaluation with respect to the same.
- Chapter 4 describes our approach for establishing an appropriate *workload* of ranking functions. Specifically, we define the proposed metric of *workload goodness*, elaborate on the heuristic solution, discuss the details of the algorithms, and present experimental results in terms of quality and efficiency of this approach.
- In Chapter 5, we extend the notion of *query similarity* into a generic model for establishing similarity between any arbitrary SPJ queries on databases (both traditional and Web), discuss its applicability across multiple applications (besides ranking) and experimentally validate its effectiveness. Likewise, we extend the proposed *user similarity* model that unifies a dynamic model based on users' browsing behavior with a static model based on their profiles.
- Chapter 6 surveys the related work with respect to the motivation of this dissertation, and Section 7 concludes the dissertation with directions for future work.

CHAPTER 2

THE QBK APPROACH FOR QUERYING THE WEB

The staples of information retrieval have been *querying* and *search*, respectively, for structured and unstructured repositories. Processing queries over known, structured repositories (such as traditional and Web databases) has been well-understood, and search has become ubiquitous when it comes to structured as well as unstructured repositories (either traditional document collections or the *surface* Web). However, the challenge of **querying unstructured sources** has not received significant attention.

As part of this dissertation, we advance the proposal that querying unstructured sources is the next step in performing focused retrievals. Specifically, this Chapter proposes a new approach (termed Query-By-Keywords (or QBK) [21] [22]) to generate queries from search-like inputs for unstructured repositories. Instead of burdening the user with schema details, we believe that pre-discovered semantic information in the form of taxonomies, relationship of keywords based on context, and attribute & operator compatibility can be used to generate query skeletons. Furthermore, progressive feedback from users can be used to improve the accuracy of query skeletons generated. The overall focus of this work is to propose an alternative paradigm for the generation of queries using as little information, initially, from the user as possible. Given that the ultimate goal is to make querying over unstructured data repositories as easy as search, we present our current work on semantics-guided query formulation.

2.1 Motivation

The paradigms of *query* and *search* have served well, respectively, as mechanisms for retrieving desired information from structured and unstructured repositories. A query (e.g., SQL) is typically quite precise in expressing *what* information is desired and is usually formed with the prior knowledge of the data sources, the data model, and the operators of the data model. On the other hand, a search request (typically, in the form of keywords) is on text repositories (including HTML and XML) and does not assume the knowledge of sources or the structure of data. A search is likely to generate a large number of results (especially when the search corpus is large and the input, by definition, is imprecise with respect to the intent) and hence ranking or ordering the results to indicate their relevance or usefulness has received considerable attention.

One of the significant differences between querying and searching is that some forms of querying require training – in the query language, the data model on which the query is being expressed, and the syntax of the query language. It also requires an understanding of the sources or schema. As a result, querying is the proclivity of those who are willing to spend time learning the intricacies of correct specification. In contrast, searching is straightforward and easy, and as a result, has become extremely popular with the advent of the Web. The lack of a learning curve associated with the notion of searching makes it useful to a large class of users. The proliferation of search and its popularity has lead researchers to apply it to structured corpus as well (e.g., DBExplorer [6] and BANKS [7]).

Consider *Motivating Example-1* (provided in Chapter 1) where the user wants to – "*Retrieve castles near London that can be reached in 2 hours by train*". Although all the information for answering the above request is available on the web, it is currently **not possible** to frame it as a *single query/search* and get meaningful results. Since this and similar queries (Section 2.2 provides additional examples of such queries) require combining information from multiple sources, search is not likely to be a preferred alternative. Although there are a number of systems (e.g., HAVASU [15], WHIRL [16], MetaQuerier [57], etc.) that combine information from multiple sources belonging to the *same* domain such as literature, airline reservations and so on, they cannot answer the above class of queries.

More recently, there is a body of work that has applied keyword searching to structured repositories so that the data can be retrieved in multiple ways and without the need for specifying a query. However, most of the work, to date, is restricted to querying sources in a single domain (for instance, Web applications such as Yahoo! Autos (`www.autos.yahoo.com` or NetFlix (`www.netflix.com`) or a set of pre-defined domains (as done in frameworks like HAVASU [17], Ariadne [18] or WHIRL [16]). Additionally, techniques proposed for querying unstructured repositories have taken the approach of converting the unstructured data into a structured form and then querying the structured data (e.g., DB-Pedia [20]). However, to the best of our knowledge, there has been no work in trying to formulate or generate queries directly over a combination of structured as well as unstructured corpus. Furthermore, different repositories relevant to the same domain may provide different sets of attribute and their values, and the number of attributes and characteristics of these repositories may vary drastically from each other. Hence, a framework for formulating queries over these repositories would be desirable, although it is a challenge.

The body of work on querying and search as it pertains to structured and unstructured corpus is characterized in Figure 2.1. As the figure indicates, the lower left and upper right quadrants are well understood[1]. It is certainly difficult to query something where the equivalent of source characteristics (source type, data model, and operators) are not known. Hence developing a framework for formulating a complete query, from an intent and a few keywords that correspond to the intent, over unstructured corpus is very much needed.

[1]Structured corpus also includes semi-structured repositories, such as XML databases, for which query languages and processing have been developed.

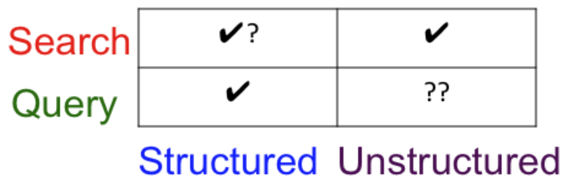

Search
Query
✓?
✓
✓
??
Structured
Unstructured

Figure 2.1. Comparison Of Query & Search Paradigms.

The focus of this Chapter is to address the problems associated with querying unstructured data sources, especially the Web. The *InfoMosaic* project [54] is investigating the integration of results from multiple heterogeneous Web sources to form meaningful answers to queries (such as the one shown above) that span multiple domains. As part of this project, we are investigating a mechanism for formulating a complete query from search-like keywords input (to avoid a learning curve) that can be processed over multiple unstructured domains to retrieve useful answers. Additionally, we are also investigating the generation and optimization of a query plan to answer the formulated structured query over the Web. More recently, this problem has received attention in literature [19] [58].

An intuitive approach to both, searching and querying, would be to use natural language to express the query. This is certainly a preferred alternative as it frees the user from the data model and other considerations needed for expressing a structured query. However, the general capability to accept arbitrary natural language queries and convert them to structured queries is still not mature. We view structured queries being at one end of the spectrum and natural language queries being at the other end. The goal of our proposed approach (that lies somewhere in-between the two and even easier than a natural language) is thus, to provide a mechanism for querying unstructured repositories with intuitive inputs and minimal user interaction.

Consequently we advance the thesis that a query is essential to extract useful and relevant information from the Web – especially when it involves integrating information that spans multiple domains and hence multiple disparate repositories. However, if only search is used or available, the burden of obtaining (through individual searches) and integrating information from multiple sources falls squarely on the user. As most of the users are familiar and comfortable with search, we want to start with a search-like input and expand it to a complete query using different types of semantic information and minimal user feedback/interaction. For instance, in the case of the above query example, a user may provide his input as a set of the following keywords: "`castle, train, London`" in any order. For the same query, another user may input: "`train, 2 hours, London, castle`". These list of words may mean several different things to different users. For example, the user may be looking for a "castle in London" or a "book written by an author named London with its titles containing words Castle and Train". The semantic information (that is assumed to be separately discovered and collected for this purpose) will assist in formulating the correct complete query with minimal interactions with the user.

As alluded to above, our proposed approach uses different types of semantic information such as: taxonomies (for context information such as travel or publishing in the above example), attributes associated with concept nodes in the taxonomy, their types, and whether they can participate in join, spatial or temporal conditions, alternative meanings of words as they appear in multiple taxonomies, compatibility of attributes across concept meanings, dictionary meaning and priority of word semantics from the dictionary to the extent possible, and finally user feedback on past interactions. It is also useful to identify metrics for comparing the approaches used for this purpose.

The remainder of the Chapter will elaborate on the proposed Query-By-Keywords (QBK) approach that uses different types of semantics and workload statistics to disambiguate and generate a partial query to capture the intent of the user.

2.1.1 Contributions And Roadmap

One of the contributions of this chapter is the novelty of the approach proposed to generate a structured query from an input that is characteristic of search. Other important contributions include: the identification of appropriate semantic information (e.g., taxonomies and other associated data) for the transformation of keywords, algorithms for generating alternative query skeletons and ranking them using compatibility and other metrics. Finally, the role and use of feedback, for improving the ranking and generating the complete structured query, is also analyzed.

We would like to clarify that the scope of this work does not include the discovery (or automatic acquisition) of semantic information described above. The thrust of this Chapter is to identify what information is needed, establish the effectiveness of this information, and an approach for transforming input keywords into a complete query. The discovery of this information is an independent problem in itself; however, it is outside the scope of this work and is not covered as part of this dissertation.

The rest of the Chapter is organized as follows. Section 2.2 provides an overview of the proposed approach with motivating examples of user intent, keyword input, and alternative queries that are generated by our approach. Section 2.3 discusses the details of steps for transforming input keywords into a complete query including keyword resolution, ranking, and query template generation. We conclude this work with directions for future research in Section 2.4.

2.2 The Query-By-Keywords Approach: An Overview

User queries that span across multiple domains (such as Travel, Literature, Shopping, Entertainment etc.) and involve different join conditions across sources in these domains can be complex to specify. For example, consider some representative queries that users would like to pose on the web:

Query 1: *Retrieve castles near London that are reachable by train in less than 2 hours*

Query 2: *Retrieve lowest airfare for flights from Dallas to VLDB 2011 conference*

Query 3: *Retrieve French restaurants within 1 mile of a IMAX theaters in Dallas, Texas*

Query 4: *Retrieve a list of 3-bedroom houses in Houston within 2 miles of exemplary schools and within 5 miles of a highway and priced under 250,000$*

Query 5: *Find a place to buy kitchen furniture within walking distance of a metro stop in Washington DC area*

Although all the information for answering the above (and similar) intents is available on the Web, it is currently not possible to pose such queries. Ideally, an appropriate retrieval framework should be capable of accepting minimal input (e.g., in the form of keywords) that characterizes the above queries from the user, and refine it into a *complete* query (such as the one shown below in response to *Query 1*) to reflect the user intent.

```
 /* Using The TRAVEL Domain */
SELECT *
FROM www.castles.org AS castle, www.national-rail.com AS train
WHERE
   train.source = 'London' AND
   train.destination = castle.location AND
   train.start_date = 06/19/2011 AND
   train.return_date = 06/19/2011 AND
   train.duration < 02 hours;
```

The approach we are proposing is to generate the above complete query by accepting a set of keywords (can also be extracted using a natural language specification). It may be possible to derive the above query *completely* from the keywords given for *Query 1* by making some minimal default assumptions about the dates based on the context. However, if a set of keywords are input, the generation of a complete query may not always be possible. Hence, we have introduced the notion of a *query skeleton* in this paper.

Web users are comfortable expressing queries through keywords rather than a query language (as displayed by the popularity of search and meta-search engines). Furthermore, current language processing techniques do not posses the ability to process and translate any arbitrary natural language query into a structured query. Hence, it is preferable for the user to express a query using a set of representative words than in natural language. For instance, some of the possible keyword representations for *Query 1* could be:

Query 1_{K1}: `castle, train, London`

Query 1_{K2}: `train, from, London, to, castle`

...

Query 1_{Kn}: `castle, train, from, London, less than, 2 hours`

The above can also be extended to specify phrases/conditions instead of only keywords. For example, "`less than 2 hours`" can be expressed together rather than separately. The phrase needs to be parsed with respect to a context. Irrespective of how the intent is mapped into keywords (e.g., alternatives *Query* 1_{K1}, *Query* 1_{K2}, ..., *Query* 1_{Kn} shown above), the final query formulated by the system in response to all these different inputs should correspond to *Query 1*. Of course, this may not be possible without some interaction with the user once the domains and potential query skeletons are identified and generated by the system. On the other hand, it is also possible that different user intents may result in the same set of keywords introducing ambiguity that needs to be identified and resolved systematically in order to arrive at the final query intended by the user. As an example, the following query intents can result in the ***same*** set of keywords from a user's perspective.

- *Retrieve Castles near London that are reachable by Train*

- *Retrieve Hotels near London that are Castles and can be reached by a Train*
- *Retrieve Books whose title contain the words 'Castle' or 'Train' written by an author whose name is 'London'*

Thus, for formulating query skeletons that converge to the actual user intent, it is necessary for the underlying system to intelligently correlate the specified keywords and generate alternative query skeletons based on the interpretation of the keywords in different domains. It is also possible that, within the same domain, multiple query skeletons can be generated by using alternative interpretations of keywords.

2.2.1 QBK Versus Other Specification Mechanisms

It is clear that there is a tradeoff between ease of query specification (or learning effort), its precision, and utility of results. Search is easy to specify but inherently vague and the result has to be sifted to obtain useful or meaningful answers (low utility). Although ranking helps quite a bit, as ranking is not always completely customized to an individual user, results need to be pruned by the user. On the other hand, a structured query is precise and the user does not have to interact with the system for obtaining meaningful answers (high utility). Of course, ranking can further help bring more meaningful answers to the top (or even avoid computing others).

Table 2.2.1 shows a back-of-the-envelope comparison of various search/query specifications across the dimensions of learning effort, precision, utility, and knowledge of schema/sources. The ultimate goal is to have a query specification mechanism that has low learning effort, high precision, high utility, and does not require the knowledge of sources. QBK is an attempt in that direction as shown in the bottom row. The purpose of the table is to quickly understand a specification mechanism along a number of metrics and learn how it stacks up against other mechanisms. The table does not include specifications such as faceted search as it is navigational with results being refined at each step. The score of

Table 2.1. Comparison of Current Intent Specification Mechanisms

Specification	**Learning Curve**	**Precision**	**Utility**	**Schema Knowledge**
SQL	High	High	Med-high	High
QBE	Low	High	Med-high	Medium
Templates	Low	High	Medium	Low
Natural Language	Low-Med	Medium	Med-high	Low-Med
Search	Low	Low	Low	Low
QBK	**Low**	**High**	**High**	**Low**

"Medium-high" utility for SQL and QBE is based on whether ranking is used or not. The Natural language (NL) row assumes ambiguities in the specification and hence the utility of results may not be "High". This table can be used as a starting point to to compare different approaches used for search as well as for querying.

2.3 The Query-By-Keywords Framework

The high-level architecture of the QBK approach for completing multi-domain queries is shown in Figure 2.2. The user provides keywords deemed significant (e.g., {`castle, train, London`} for *Query 1*) instead of a full-fledged query. The *Keyword Resolution* phase checks these keywords against the *Knowledge Base* for resolving each keyword to entities, attributes, values, or operators. For a keyword that occurs as a heteronym (i.e., same entity representing multiple meanings/contexts) in the *Knowledge Base*, all possible combinations for the different meanings of this keyword is generated. This occurs when the keyword occurs in different taxonomies corresponding to different domains/context. From these combinations, query skeletons and any other conditions (or attributes on which conditions are possible) are generated.

These query skeletons are ranked by the *Rank Model* in the order of relevancy (to the input and past usage) and shown to the user to choose one that corresponds to his/her intent. Both keyword resolution and ranking is based on the information in the *Knowl-*

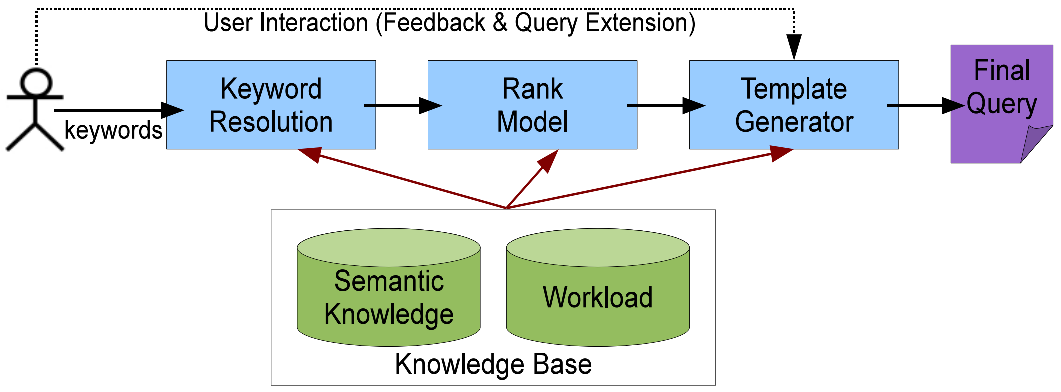


Figure 2.2. The QBK Approach: From Keywords To A Complete Query.

edge Base. Subsequently, a template is generated with all the details which can be further filled with additional conditions. The list of entities that are of interest, the domain and sources to which they map, and the possible list of simple conditions (e.g., $train.startTime < relationaloperator > value$) or attributes as well as join conditions (e.g., $castle.location < traditional/spatial/temporaloperator > train.startLocation$) are shown. Additionally, a list of attributes is displayed for the choice of result attributes. The user fills/modifies the template in a manner similar to filling a Web query interface so that an unambiguous and complete query can be generated for further processing. Although this approach involves a minimum of three to four steps, we believe that such an approach will be useful for a novice user in understanding and using the system.

2.3.1 Knowledge Base

The Knowledge Base consists of a *Semantic Knowledge repository* that contains taxonomies organized by domains/context including meta-information about the entities, sources, operators, attributes and values. This is used in the keyword resolution phase and for constructing query skeletons. The Knowledge Base also consists of a *Workload reposi-*

tory that organizes the query skeletons selected by different users, in the past, with respect to the specified keyword input.

Semantic Knowledge Repository: This repository contains pre-discovered *semantic* information in the form of a collection of taxonomies associated with domains and are populated with appropriate types of meta-data. For instance, the domain of *Travel* can be represented using taxonomies for – *transportation*, *travel lodging*, and *tourist attractions*. Similarly, the domain for *Literature* may contain taxonomies such as *journal*, *book*, etc.. These represent the roots of different taxonomies within the given domain. Nodes in each taxonomy represent entities (e.g., castle, museum, church, etc.) associated with the domain corresponding to a *is-a* relationship[2].

In addition to the *is-a* hierarchy, supplementary meta-data is associated with each node in a taxonomy. For instance, an entity *castle* in the *tourist attractions* domain may have several types of associated meta-data such as –

- *Web sources* (e.g., `www.castles.org`) from which attribute values relevant to this entity can be extracted,
- Common *attributes* (e.g., name, age, country_location) that are associated with the entity, and
- *Semantics* representing linguistic meaning and semantic popularity.

Additionally, each attribute of a given entity could have several types of meta-data associated with it such as – i) the data type of the attribute (e.g., string, numerics, etc.), ii) its category (spatial, temporal, generic), iii) its domain, and iv) associated synonyms. For leaf-level entities in a taxonomy, the *values* for certain categorical attributes are associated. This is needed to resolve keywords that are values themselves (e.g., London) and infer the entity for which it is a value (e.g., city). As this set can be arbitrarily large, a way to

[2]In this paper, we assume the availability of such taxonomies. Simple taxonomies can be generated using a combination of Web directories (e.g., Yahoo Dictionary) and dictionaries (e.g., Webster).

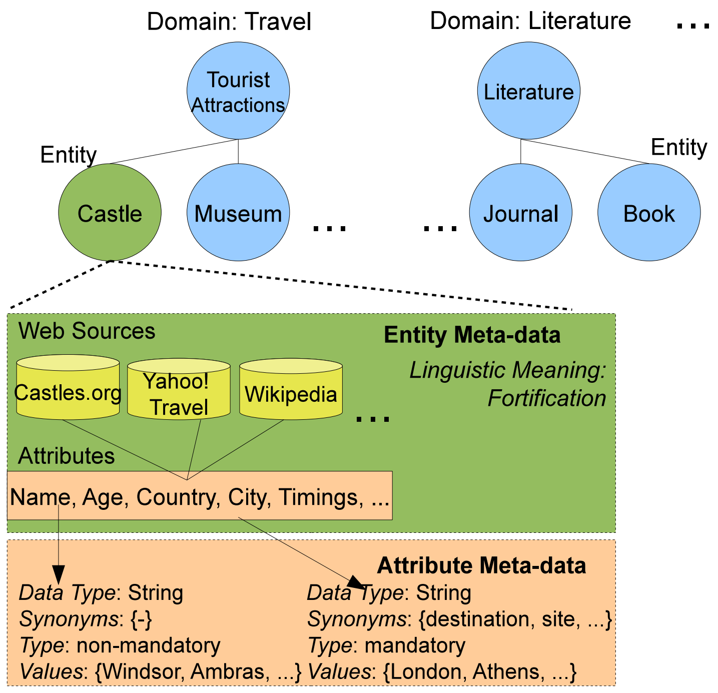


Figure 2.3. Sample Taxonomy For The ***Travel*** Domain.

infer them (using a tool like WordNet [59]) instead of storing all the values is going to be important. The list of relevant Web sources corresponding to an entity can be obtained using search engines and the information associated with Web directories. Figure 2.3 shows a snapshot of such semantic information associated with the entity – *castle* in the taxonomy of – *tourist attractions* within the *Travel* domain.

In addition to meta-data, information about the *compatibility* between entities also needs to be maintained. Two entities are said to be *compatible* if a meaningful *join* condition can be formulated between (one or more) attributes of the participating entities. For instance, the entities *castle* and *train* are compatible since their respective attributes *location* and *source location* can be compared. The join could result in traditional, spatial, or temporal conditions based on the attribute types and the operators associated with them. This compatibility across entities can be identified by matching the respective attribute data types. A compatibility matrix can be used as the number of operators is not that large. Compatibility information of successfully formulated past queries can also be used for this purpose. Another component of this repository is a list of operators that are applicable to attributes. We assume – relational operators ($==$, $!=$, $<$, $<=$, $>$, $>=$), Boolean operators, temporal operators (derived from Allen's Algebra [60]), and a few spatial operators (such as near, within, etc. [61]).

It is evident that building such a comprehensive repository of semantic information is separate problem in itself and is beyond the scope of this dissertation. Given such a repository, its completeness and algorithmic usage in formulating queries is of concern here.

Workload Repository: Past statistics based on user interaction can play an important role in the query formulation process as it indicates significance of user preferences. Hence, we maintain a repository that is a collection of statistics and feedback associated with past queries. Specifically, it comprises of the information associated with users' preference of the query skeletons from those generated by the *Rank Model*. Additionally, statistics of the attributes of entities used for specifying conditions or output are also collected. This information is used for choosing widely preferred attributes for an entity in the generation of skeletons and templates. This repository is constructed using the feedback collected as a by-product of the query formulation process. For every keyword, the following user preferences are collected:

1. *Context* of the individual keyword (e.g., *castle* belonging to the travel domain frequently chosen over others),
2. *Frequency* of the attributes used for specifying conditions or chosen for output, and
3. *Interpretation* of the desired query.

Subsequent statistics associated with selection of attributes while filling the query template are also collected and stored.

Table 2.2. Input Keywords & Their Taxonomical Associations

Keyword	Occurs As	Taxonomy	Domain	Attribute Value Association
CASTLE	Entity	Tourist Attraction	Travel	-
	Entity	Travel Accomodation	Travel	-
	Value	Books	Literature	Book.*title*
TRAIN	Entity	Transport Mode	Travel	-
	Value	Books	Literature	Book.*title*
	Value	Movies	Entertainment	Movie.*title*
LONDON	Value	City	Geography	City.*name*
	Value	Books	Literature	Book.*author*
	Value	Business	Industry	Company.*name*
	Value	Movies	Entertainment	

2.3.2 Keyword Resolution

The purpose of this phase is to map specified keywords into the domains that are stored as part of the Knowledge Base. Coverage of keywords in each domain is important as it indicates the relevance of the domain to the input. This phase performs matching for each of the input keywords with – *entity* names in the taxonomies, *attribute* names associated with each node (entity) in the taxonomies, and *values* associated with leaf-level entity attributes.

Since a domain comprises multiple taxonomies, it is possible (as shown in Table 2.3.2) that the same keyword (*castle*) will belong to multiple taxonomies (*Tourist Attractions* and *Travel Accommodations*) in a single domain (*Travel*). In such cases, determining which intent the user has in mind is not possible. Hence, the resolution phase checks for multiple instances of the same keyword in different taxonomies, each giving rise to a separate entity set (and hence, separate query intents) for each occurrence of the entity.

Additionally, it is also possible for the same keyword to occur in taxonomies in multiple domains (*castle*) occurs in the domains of *Travel* and *Literature* as shown in Table 2.3.2.

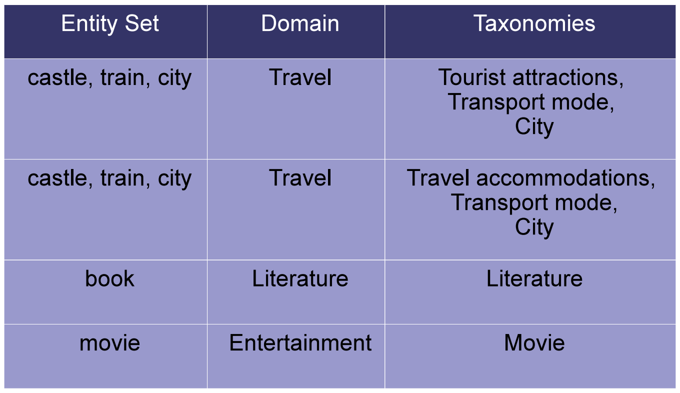
Hence, the resolution phase analyzes all the domains independently to get the list of entity sets within each domain.

Further, it may so happen that the keyword may not always match to an entity in the taxonomy i.e., it may match to an attribute of an entity or the value of an entity's attribute. In such cases, the immediate parent entity (to which the attribute/value belongs) is chosen to represent the input keyword. Further, the keywords not finding a match to any of these categories are compared against a set of operators to determine their occurrences as a spatial, temporal, or generic operators and an operator-list. The keywords that do not find any match to either entities, attributes or operators are ignored.

Thus, for a given set of input keyword, the resolution process generates a list of entity sets belonging to the same or multiple domains where every set comprises of entities that belong to the same domain (but might map to multiple taxonomies within that domain). For instance, the outcome of the resolution process for intent *Query* 1_{K1} (for *Query 1*) (based on the keyword matching results shown in Table 2.3.2) is shown in Figure 2.4. As the figure indicates, it is possible that the given set of input keywords may generate multiple combinations (of entities). This would make the task of separating the relevant intents from the irrelevant ones extremely hard. Hence, in order to establish an order amongst these combinations, the output generated by the *resolution* phase is fed to the *Rank Model*.

2.3.3 Rank Model

Consider an input set of keywords $\{K_1, K_2, ..., K_n\}$, and for the sake of simplicity, let the outcome of the *Keyword Resolution Process* be $\mathcal{L}$: $\{E_1, E_2, ..., E_m\}$, where each $E_j \in \mathcal{L}$ represents a set of "n" entities given by E_j: $\{e_{j_1}, e_{j_2}, ..., e_{j_n}\}$ such that e_{j_i} represents a distinct entity corresponding to input keyword K_i. Now, if the size of set $\mathcal{L}$ is large, determining the most relevant combination (of entities) is important. Correspondingly, we propose a ranking technique to order these combinations (in $\mathcal{L}$) based on three different



Entity Set	Domain	Taxonomies
castle, train, city	Travel	Tourist attractions, Transport mode, City
castle, train, city	Travel	Travel accommodations, Transport mode, City
book	Literature	Literature
movie	Entertainment	Movie

Figure 2.4. Keyword Resolution Outcome: *Query* 1_{K1}.

parameters, namely – i) *linguistics*, ii) *statistics*, and ii) *join compatibility*. In the rest of the section, we elaborate on each of these parameters, and explain their importance in ranking entity sets and discuss the mechanism to fuse them in a single ranking function.

2.3.3.1 Linguistics

It is a common observation [62] that *linguistic meaning* of a keyword plays an important role when users specify their input. For instance, as per WordNet [59], the two (of the many) possible meanings associated with the keyword *castle* are: i) a large and stately mansion, and ii) a piece in the game of chess. However, as pointed out by WordNet, users generally adhere to an established language model [63] to formulate their natural language queries as well as search queries. That is, they choose the meaning which is linguistically more popular than the rest of the meanings for the same keyword (in this case the former meaning of *castle* will be chosen over the latter in most cases). Thus, there is reason to believe that when users express their keyword input to formulate queries over multiple domains, they will use a similar language model of picking linguistically popular meanings for an entity.

Based on this observation, consider an input keyword K_i and let $\{e_{1_i}$, e_{2_i}, ..., $e_{m_i}\}$ represent the distinct meanings associated with this keywords. Each meaning, in turn represents an entity within a specific taxonomy in the Knowledge Base. In order to clearly distinguish and order these different entities (i.e., the meanings), we assign a *linguistic score* to each entity e_{j_i} corresponding to K_i. This score (represented as $score_L(e_{j_i})$) is determined as follows: Given an input K_i, WordNet returns the set of entities $\{e_{1_i}$, e_{2_i}, ..., $e_{m_i}\}$ wherein each entity has a rank associated to it (based on it's linguistic popularity). We normalize these ranks and translate them into *linguistic score* such that every entity associated with any given keyword will be assigned a score in a fixed interval of (0.0, 0.1). The reason for such a normalization is due to the fact that the number of entities (meanings) associated with each input keyword may be different; thus, the ranges (within which the ranks of entities corresponding to a keyword occur) will vary from one keyword to another. Consequently, in order to establish a consistent scoring model for any keyword input, we normalize the resulting ranks provided by WordNet. Formally, the linguistic score of a specific entity e_{j_i} associated with a given keyword K_i is computed as:

$$score_L(e_{j_i}) = 1 - \frac{rank(e_{j_i}) - 1}{m - 1} \tag{2.1}$$

where, $rank(e_{j_i})$ represents the rank assigned by WordNet to this entity, and "m" is the total number of entities corresponding to K_i.

Now, given an entity set E_j: $\{e_{j_1}$, e_{j_2}, ..., $e_{j_n}\}$, each entity e_{j_i} receives a linguistic score based on its association with keyword K_i. Since the overall goal is to order the different combinations of entity sets in $\mathcal{L}$, we need a mechanism to assign a global score to E_j. Toward that, we rely on an **independence assumption** i.e., given that the entities within a set E_j are not associated with each other in a linguistic context, we assign a

Global Linguistic Score to each entity set $E_j \in \mathcal{L}$. This score, represented as $Score_L(E_j)$, is computed as shown by Equation 2.2:

$$Score_L(E_j) = \prod_{i=1}^{n} score_L(e_{j_i}) \tag{2.2}$$

Thence, the set $\mathcal{L}$ can be ordered based on the score, computed by Equation 2.2) and assigned to each of its element.

2.3.3.2 Statistics

The notion of applying a linguistic ordering is *static* in nature i.e., since the linguistic meanings of a word never change, the ordering assigned to $\mathcal{L}$ will always remain the same. However, based on browsing choices of users, it might be the case that certain combinations of entities, that receive a lower rank based on linguistics, may in fact, be more frequently selected by users. In order to accommodate users' preferences towards specific sets of entities, we include another parameter in the ranking model i.e., *statistics*, and rely on the past user queries from the *Workload Repository* in our *Knowledge Base* for the same.

Accordingly, given $\mathcal{L}$: $\{E_1$, E_2, ..., $E_m\}$, we assign a *Statistic Score* to every element of this set (given as $score_S(E_j)$). This score is basically the frequency with which a particular E_j is selected by the user whenever the input to the system is the set $\{K_1$, K_2, ..., $K_n\}$. However, in the context of real Web applications, the probability that a large number of users pose the exact same set of input keywords – $\{K_1$, K_2, ..., $K_n\}$ is very less; thus, affecting the computation of the *Statistic Score* for the elements in $\mathcal{L}$. In contrast, the chances of multiple users asking the same keyword (say K_1) is much higher. Therefore, similar to the linguistic model, we avail of the **independence assumption**, and associate a *statistic score* to every distinct entity e_{j_i} within E_j. Formally, the statistic score of a specific entity e_{j_i} associated with a given keyword K_i is computed as:

$$score_S(e_{j_i}) = \frac{frequency(e_{j_i})}{frequency(K_i)} \tag{2.3}$$

The *Global Statistic Score* for a given $E_j \in \mathcal{L}$ is then given as:

$$Score_S(\mathbf{E_j}) = \prod_{i=1}^{n} score_S(e_{j_i}) \tag{2.4}$$

Using Equation 2.4, the set $\mathcal{L}$ can thus, be ordered based on the notion of frequency.

2.3.3.3 Join Compatibility

Given any entity set E_j: $\{e_{j_1}, e_{j_2}, ..., e_{j_n}\}$, the user would most likely be interested in formulating query conditions that involve joining the different entities based on the commonality of the attributes between these entities. Since, the exact query conditions at this stage cannot be determined, we believe that an entity set that allows the flexibility to join every entity to every other entity on some attribute is definitely desirable than a set that offers very less combinations across entities. In addition, the flexibility to join any two entities on a larger number of attributes (instead of just one or two) is definitely desirable. Hence, for every set $E_j \in \mathcal{L}$, we define a *Join Compatibility Score*.

Consider the entity set $\{castle, train, city\}$ representing the input *Query* 1_{K1}. It is possible to join the entities $castle$ and $train$ based on an attribute (e.g., location), the entities $train$ and $city$ based on an attribute (e.g., location) and the entities $castle$ and $city$ based on an attribute (e.g., name, location). On the other hand, an entity set $\{book, city\}$ representing the input keywords (considering the keywords ‘castle’ and ‘train’ refer to the name of a book i.e., its attribute value), will not allow any joins between the two entities and restrict a number of query conditions the user would like to formulate. Thus, it is clear that the first entity set should be ranked higher than the second.

For a given set of keywords $\{K_1,\ K_2,\ K_3\}$, let $\{E_1,\ E_2,\ E_3\}$ represent the three different entity sets within $\mathcal{L}$. Now, let E_1: $\{e_{1_x},\ e_{1_y},\ e_{1_z}\}$, and likewise for E_2 and E_3. In

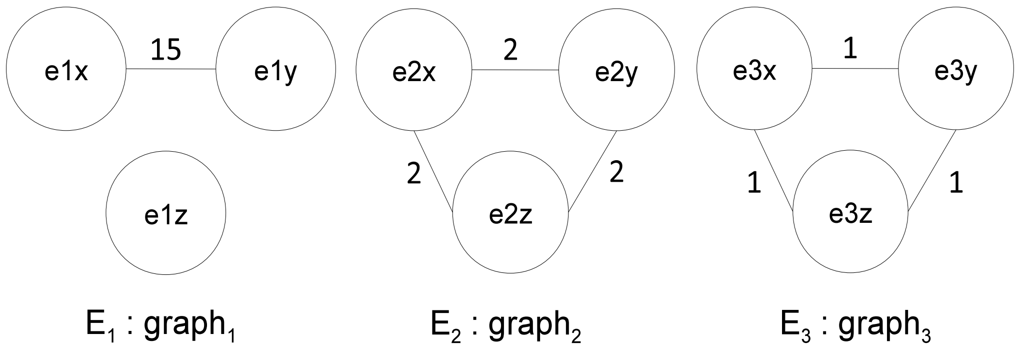


Figure 2.5. The Notion Of Join Compatibility For Entity Sets E_1, E_2, E_3.

order to estimate *join compatibility*, we represent each entity set as a graph (e.g., $graph_1$ representing E_1) whose vertices represent the entities ($\{e_{1_x},\ e_{1_y},\ e_{1_z}\}$ in this case). If any two entities in the set can be joined on an attribute, then an edge exists between the corresponding two vertices (representing the entities). The edge weight is represented by the total number of distinct attributes that can be used for the joining the two entities. A sample instance of such graphs for the entity sets in $\mathcal{L}$ is shown in Figure 2.5.

In the case of $graph_1$ (corresponding to E_1) for instance, the edge weight (of 15) between e_{1_x} and e_{1_y} indicates that these two entities can be joined on 15 different attributes. However, it is also evident that no conditions can be formed between e_{1_x} and e_{1_z} (as well as between e_{1_y} and e_{1_z}); thus, making it less applicable for the given input. In contrast, $graph_2$ reveals that every entity can be joined with every other entity using two distinct attributes; hence, making it a better choice over $graph_1$ (as well as $graph_3$ wherein all entities can be joined with each other based on only a single attribute). Accordingly, we can surmise that an entity set that allows the flexibility to form join conditions, using multiple attributes, between any pair of entities within it would be definitely preferable to the user, and hence, must receive a very high score in the overall ranked list of entity sets.

Formally, given an entity set E_j: $\{e_{j_1}, e_{j_2}, ..., e_{j_n}\}$, we determine a *Join Compatibility Score* for this set of entities by applying the *Maximum Spanning Tree* [64] algorithm to every combination. It can be observed that, larger the number of spanning trees on a combination, more is the flexibility of joining any entity pair. Similarly, greater the sum of the weight produced by each Spanning Tree, the options for the user to join a given entity pair increase. Thus, for a given E_j, if the total number of distinct maximum spanning trees (M.S.T.) is *M* and the weight of each tree T_i is W_i, the *Join Compatibility Score* (Equation 2.5) for E_j is given as:

$$score_J(E_j) = \frac{M}{n} * \sum_{i=1}^{n} W_i \tag{2.5}$$

where "n" is the total number of distinct entities in a given E_j.

Thus, using Equation 2.5, all the elements of set $\mathcal{L}$ can be ordered based on their ability to be involved in queries with sufficiently large and diverse join conditions.

2.3.3.4 Putting it all Together: A Single Ranking Model

We have discussed three distinct parameters that can play an important role in ranking the outcome of the *Keyword Resolution Process*. Since they are derived from linguistics, statistics as well as analyzing the nature of entities within a set (for ensuring query conditions), we believe that when combined together, they form a comprehensive and suitable model to order the set $\mathcal{L}$.

However we need to determine a mechanism to translate these basic intuitions into quantitatively describable ranking functions, and to combine these varied parameters into a single ranking function. Typically, ranking functions are designed using a combination of learning methods and statistical techniques. One of the popular, and in this case applicable technique is the user of *regression analysis*. More specifically, we use the concept of *Logistic Regression* [65] to model our ranking function since it is a well-established and proven model used for predicting the outcome of an event when the parameters influencing the event are diverse and unrelated to each other (as is the case with our parameters).

An important issue with regression analysis is to collect a training set to learn the parameter weights used for ranking the combinations. However, as the issue at hand is to rank query intents instead of results, coming up with an appropriate training set is an independent problem that can be resolved using synthetic query generation and/or user studies. Initially, we set uniform regression parameters i.e., equal weight to every parameter (linguistics, statistics and join compatibility) to rank the elements of $\mathcal{L}$. Accordingly, for each element of this set, the ranking function assigns a score which is a sum of the three scores described in the above sections. As more and more workload is collected, we believe it will act as a major component in generating a significant training data to learn the parameter weights.

Logistic regression is generally defined using a *logistic ranking function* given by:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

The input variable z, in our model represents the parameters – semantics, statistics and join-compatibility, while the output *f(z)* represents the probability or score of every element in $\mathcal{L}$. The *logit* variable z is defined as:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 \tag{2.7}$$

where β_0 is the intercept, $\beta_{1..3}$ represent the regression coefficients, and $x_{1..3}$ represent the predictor variables – linguistics, statistics and compatibility respectively.

The regression coefficients are weights that depict the importance of each parameter and are learned through statistical learning methods (in this case, Maximum Likelihood Estimation [66]). Once the scores for each parameter is generated, the total score for every combination in the space is generated by the ranking function which establishes a ranking order for the keyword combinations.

2.3.4 Query Completion

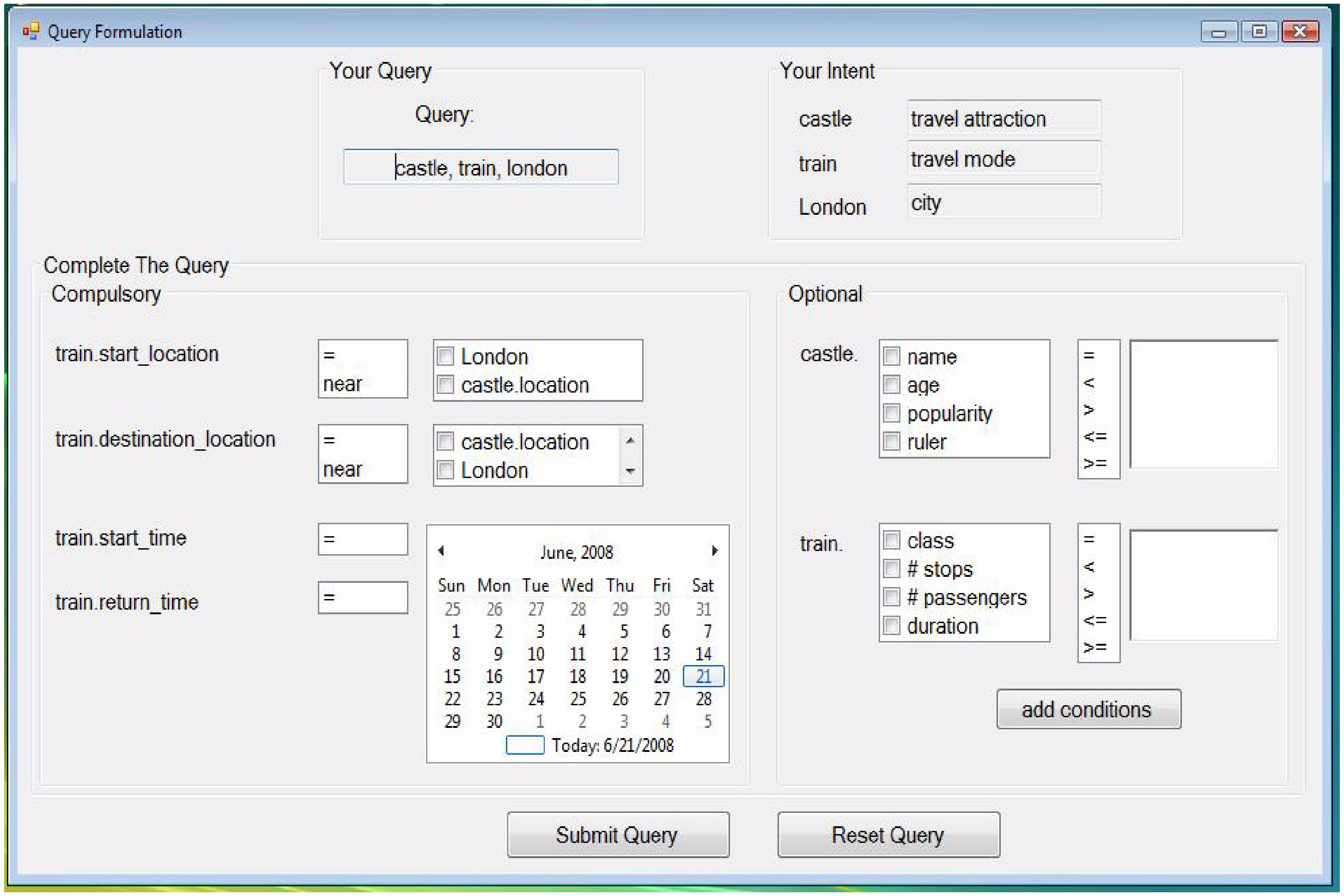
Based on the previous phases, our approach generates a template with a partially-filled query. Each keyword in the input is accounted for. For an entity, the template is populated by its corresponding domain and the underlying Web source to which it belongs. For instance, for *Query* 1_{K1} if the user selected *castle* to represent a *tourist attraction*, *train* to represent the *transportation mode*, and *London* is identified as a *city*, then the domains (tourist attractions, transportation) and sources (www.castles.org, www.london-rail.com) are obtained from the Knowledge Base and the template is appropriately populated.

For every entity, the list of mandatory attributes are obtained and the corresponding partial conditions are formulated. A *mandatory attribute* represents the attribute for which a data value *must* be provided by the user for querying the underlying Web source. For instance, the mandatory attributes associated with *train* as a *transportation mode* are – $sourcelocation$, $destinationlocation$, $starttime$, and $returntime$. Additionally, the attributes associated for the entity which are popular and occur in a large number of similar past queries are also used to formulate possible partial query conditions of interest to the user. For instance, the *workload repository* may indicate that *age* is a popular attribute for the *castle* entity and has featured in a large number of queries. In such a scenario, a generic condition of the form – $castle.age$ $\{relationalOperator\}$ $\{value\}$ could be generated. Furthermore, popular attributes desired as output by users for the corresponding entities may also be generated to fill the `SELECT` clause of the query template. If there exist multiple entities in the user intent, the possible integration conditions possible between them are generated. For instance, $castle.location$ $\{spatialOperator\}$ $\{train.startLocation$, $train.endLocation$, $city.Location\}$ would be generated.

For an attribute, if the attribute has not been listed in the query template in the first step, it is analyzed for its type (spatial, temporal, generic) and the entity associated with it is obtained to formulate query conditions of the type: $entity.attribute\ \{operator\}\ \{value\}$. If this attribute can be formulated as an integration condition, then the corresponding conditions are formulated. Similarly, if the attribute is a popular choice for the output, then the `SELECT` clause is populated with it. For a value (e.g., London), the corresponding attribute and its parent entity are derived and condition of the type $city.name = London$ is formed. For operators, if they are not listed in the template in the above steps, the possible conditions between the entities for which the operators are applicable are analyzed and modified accordingly. For instance, if an *operator* "near" is specified for the above intent, then the integration condition can be modified as: $castle.location\ near\ \{train.startLocation$, $train.endLocation$, $city.Location\}$. As the last stage of user interaction, the template is filled/modified by the user based on his/her preferences and the complete query is formulated that captures the exact user intent in terms of constraints and conditions across multiple domains. A sample query template, generated in response to the input *Query* 1_{K1} (i.e., "`castle, train, London`") is shown in Figure 2.6.

2.4 Conclusion

In this Chapter, we have presented a novel semantic-based approach termed Query-By-Keywords (or QBK) for query specification, and elaborated on it's usefulness for Web users who are familiar with the paradigm of *keyword* search. We have also demonstrated how this approach can be carried out with the help of a Knowledge Base comprising of various kinds of semantic and statistic information. We have detailed the steps involved in the generation of a query skeleton, its ranking, and how a complete query can be generated with meaningful user interaction.



Query Formulation

Your Query

Query: castle, train, london

Your Intent

castle	travel attraction
train	travel mode
London	city

Complete The Query

Compulsory

train.start_location	= / near	London / castle.location
train.destination_location	= / near	castle.location / London
train.start_time	=	
train.return_time	=	

June, 2008

Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

Today: 6/21/2008

Optional

castle.	name / age / popularity / ruler	= / < / > / <= / >=
train.	class / # stops / # passengers / duration	= / < / > / <= / >=

Figure 2.6. Sample Template For QBK Framework

CHAPTER 3

A SIMILARITY-BASED RANKING FRAMEWORK

With the emergence of the deep Web, searching Web databases in domains such as vehicles, real estate, flights and so on, has become a routine task. One of the problems in this context is ranking the results of a user query. Earlier approaches for addressing this problem have used frequencies of database values, query logs, and user profiles. However, a common thread in most of these approaches is that ranking is done in either a user-independent fashion and/or a query-independent manner.

In this Chapter, we elaborate on a novel query- and user-*dependent* approach for ranking query results in Web databases. We present a ranking model, based on two complementary notions of *user* and *query* similarity, to derive a ranking function for a given user query. The model is based on the intuition that similar users display comparable ranking preferences over the results of similar queries. The requisite ranking function is thus, acquired from a sparse workload comprising of several such functions derived for various user-query pairs. We define these similarities formally in alternative ways and discuss their effectiveness analytically and experimentally over two distinct Web databases. We also advance a learning technique to derive individual ranking functions (for the workload) using implicit browsing behavior of users, and experimentally validate its effectiveness.

3.1 Introduction

The emergence of the deep Web [11] [12] has led to the proliferation of a large number of Web databases for a variety of applications (e.g., airline reservations, vehicle search, real estate scouting). These databases are typically searched by formulating query conditions on their schema attributes. When the number of results returned is large, it is time-consuming to browse and choose the most useful answer(s) for further investigation. Currently, Web databases simplify this task by displaying query results sorted on the values of a single attribute (e.g., Price, Mileage, etc.). However, as alluded to by *Example-2* (in Chapter 1) most Web users would prefer an ordering derived using multiple attribute values, which would be closer to their expectation. For the sake of clarity and to better understand the problem at hand, we reproduce *Example-2* for further analysis.

Motivating Example-2: *Consider Google Base's* Vehicle *database that comprises of a table with attributes Make, Price, Mileage, Location, Color, and so on, where each tuple represents a vehicle for sale.*

*Two users – a business executive (U_1) and a student (U_2), seek answers to the same query (Q_1): "***Make = Honda AND Location = Dallas, TX***". In response to this query, more than 18,000 tuples are returned. Intuitively, U_1 would typically search for* ***new*** *vehicles with specific* ***color*** *choices (e.g., only* ***red*** *colored vehicles), and hence would prefer vehicles with "****Condition = New AND Color = Red****" to be ranked and displayed higher than the others. In contrast, U_2 would most likely search for* ***used*** *vehicles* ***priced*** *under a specific amount (e.g., "Price $<$ \$ 5,000"); hence, for U_2, the results should be ordered such that vehicles with "****Condition = Used AND Price $<$ \$ 5,000****" are displayed before the rest.*

*The same student user (U_2) now moves to Mountain View, CA as an intern with Google and asks a different query: "***Make = Pontiac AND Location = Mountain View, CA***". We can presume (since he has procured an internship) that he may be willing to pay a slightly higher* ***price*** *for a lesser* ***mileage*** *vehicle (e.g., "Mileage $<$ 100,000"), although he may still be searching for* ***used*** *vehicles. His preferences for this query would then require vehicles with "****Condition = Used AND Price $<$ \$ 10,000\$ AND Mileage $<$ 100,000****" to be ranked higher than others.*

The above example illustrates that different Web users may have contrasting ranking preferences towards the results of the same query. Furthermore, it emphasizes that the same user may display different ranking preferences for the results of different queries. Thus, it is evident that in the context of Web databases, where a large set of queries given by varied classes of users is involved, the corresponding results are likely to be better received when ranked in a *user*- and *query*-dependent manner.

The current sorting-based mechanisms used by Web databases do not perform such sophisticated ranking. While some extensions to SQL allow *manual* specification of ranking functions/preferences [38] [39] [40] [41], this approach is cumbersome for **most** Web users. *Automated* ranking of database results has been studied in the context of relational databases, and although a number of techniques [67] [43][44] [45] perform *query-dependent ranking*, they do not differentiate between users and hence, provide a single ranking order for a given query across *all users*. In contrast, techniques for building extensive user profiles [36] as well as requiring users to order data tuples [48], proposed for *user-dependent* ranking, do not distinguish between queries and provide a single ranking order for any query given by the same user. Recommender (i.e., collaborative [28] [29] [30] and content filtering [32] [33] [34]) systems use the notions of user- and object/item-similarity for recommending objects to users. Although our work is inspired by this idea, there are differences that prevent its direct applicability to database ranking.

In this Chapter, we present a *user*- and *query-dependent* approach [49] [50] for ranking the results of Web database queries. For a query Q_j given by a user U_i, if the corresponding ranking function (represented as $\mathcal{F}_{U_i,Q_j}$) is not available, then a relevant ranking function (say $\mathcal{F}_{U_x,Q_y}$) corresponding to U_x's preferences towards the results of Q_y (where U_x and Q_y are similar to U_i and Q_j respectively), is identified from a workload of ranking functions (inferred for a number of user-query pairs). This function is then used to rank Q_j's results for U_i. This choice of an appropriate function is based on a novel *similarity-*

based ranking model proposed as part of this dissertation, and elaborated in this Chapter. The intuition behind this approach is:

- For a given query's results, *similar* users display *comparable* ranking preferences.
- A user displays *similar* ranking preferences over results of *analogous* queries.

Therefore, we decompose the notion of similarity into – 1) ***query similarity***, and 2) ***user similarity***. The former is estimated using either of the two proposed metrics namely, i) *query-condition similarity*, determined by comparing the conditions in these queries, or ii) *query-result similarity* based on a comparison between actual query results. On the other hand, the latter is calculated by comparing individual ranking functions over a set of common queries between users. Although each model can be applied independently, we also propose a holistic model to determine an improved ranking order. The ranking function (defined formally in Section 3.2.2) in our framework is a *linear weighted-sum* function comprising of: i) *attribute-weights* denoting the significance of individual attributes and ii) *value-weights* representing the importance of attribute values.

In order to make our approach practically useful, a minimal workload is important. Although we propose a heuristic approach for establishing such workloads later in this dissertation (i.e., in Chapter 4), the focus of the current Chapter is on the usage, instead of the acquisition of such workloads; therefore, for the rest of this Chapter, we assume the availability of a workload of suitable ranking functions collected from several user-query pairs. However, we do elaborate on a novel learning method that derives these individual ranking functions for the workload.

Contributions: The primary contributions of this work are:

- Formulation of a *user*- and *query-dependent* approach for ranking query results of Web databases.

- Proposal for a novel ranking model, based on two complementary measures of *query similarity* and *user similarity*, to derive functions from a workload containing ranking functions for several user-query pairs.
- Results of an extensive experimental evaluation using Amazon Mechanical Turk over two Web databases supported by Google Base to validate our approach in terms of efficiency as well as quality for real-world use.
- Elaboration of learning method for deriving ranking functions from implicit user behavior and the results for its experimental evaluation.

Roadmap: Section 3.2 formally defines the ranking problem. In Sections 3.3 and 3.4, we respectively elaborate the proposed models for query and user similarity. Section 3.5 explains the combined model of similarity, and Section 3.6 respectively details the experimental results. In Section 3.7, we highlight the proposed *learning method*, along with its experimental evaluation, for deriving a ranking function, and conclude in Section 3.8.

3.2 Problem Definition And Architecture

In this Section, we formally define the ranking problem for Web databases and present a general architecture of our solution.

3.2.1 Problem Definition

Consider a Web database table *D* over a set of *p* attributes, *A* = $\{A_1, A_2, ..., A_p\}$. A user U_i asks a query Q_j of the form[1]: "SELECT * FROM *D* WHERE A_1 = a_1 AND $\cdots$ AND A_k = a_k", where each $A_i \in$ *A* and a_i is a value in the domain of A_i. Let N_j =

[1]The specified query Q_j is **not** a SPJ query since it does not contain projection and join predicates. Further, it comprises of conjunctive point conditions in the selection predicates expressed over a single relation. However, later in Chapter 5 of this dissertation, we generalize our framework to support arbitrary SPJ queries comprising of range, IN, disjunction and negation conditions over multiple relations.

$\{t_1, t_2, ..., t_n\}$ be the set of result tuples for Q_j. Further, let $\mathcal{U}$ = $\{U_1$, U_2, ..., $U_M\}$ and $\mathcal{Q}$ = $\{Q_1$, Q_2, ..., $Q_N\}$ respectively represent a large set of users and queries over *D*, and let **W** represent the workload (refer to Tables 3.1 and 3.2 for an example) comprising of distinct ranking functions collected across several user-query pairs (from $\mathcal{U}$ and $\mathcal{Q}$).

The ranking problem can be stated as: "*For the query Q_j given by the user U_i, determine a* ranking function $\mathcal{F}_{U_i,Q_j}$ *from* **W**". A ranking order can then be obtained by applying $\mathcal{F}_{U_i,Q_j}$ to N_j. Given the scale of Web users and the large number of queries that can be posed on *D*, **W** will not possess a function for every user-query pair; hence the need for a *similarity-based* method to find a **suitable** function ($\mathcal{F}_{U_x,Q_y}$) in place of the missing $\mathcal{F}_{U_i,Q_j}$. The ranking problem, thus, can be decomposed into:

Sub-problem 1 – Identify a ranking function using the similarity model: Determine a user U_x similar to U_i and a query Q_y similar to Q_j such that $\mathcal{F}_{U_x,Q_y} \in \mathbf{W}$.

Sub-problem 2 – Establish a workload of ranking functions: Determine a subset of users (in $\mathcal{U}$) and a subset of queries (in $\mathcal{Q}$) to represent a workload **W**, such that for *any* query Q_i ($\in$ $\mathcal{Q}$) asked by *any* user U_j ($\in$ $\mathcal{U}$), there exists at least one pair (e.g., $(Q_x, U_y)) \in$ **W** such that the similarity between the queries Q_x and Q_i and/or the similarity between users U_y and U_j, is very high.

Sub-problem 3 – Derive individual ranking functions for the pairs in W: For a user U_x and a query Q_y belonging to **W**, based on U_x's preferences towards Q_y's results, determine, explicitly or implicitly, a ranking function $\mathcal{F}_{U_x Q_y}$.

In this Chapter, we propose a solution to Sub-problems 1 and 3 listed above. The solution to Sub-problem 2 is elaborated in Chapter 4. In addition, given that the similarity-based ranking framework is proposed in the context of Web databases, the solution presented in this Chapter is addressed to cover **only** point queries with conjunctive conditions. However, we propose a refined formal model to support generic SPJ queries comprising of projections, joins, and selections (specifically IN/range condition, and different Boolean

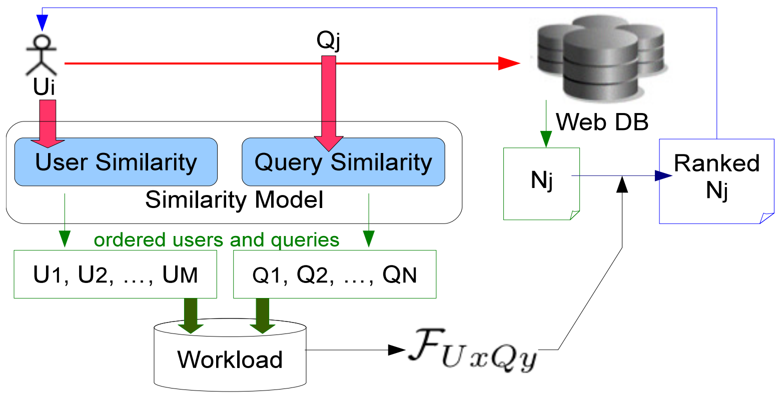


Figure 3.1. Architecture Of Similarity-based Ranking Framework.

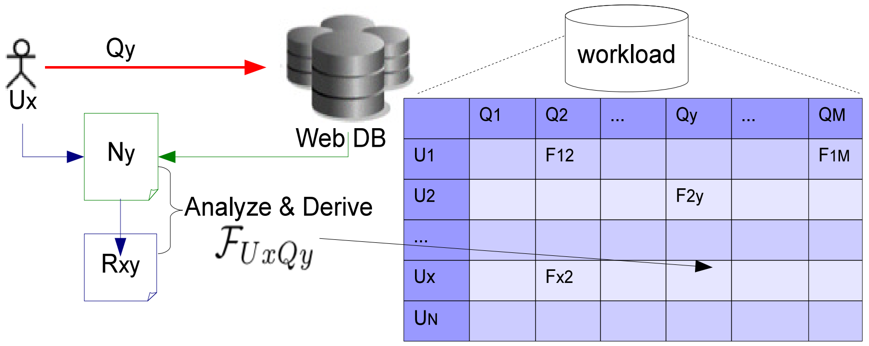


Figure 3.2. High-level View For Establishing A Workload.

operators like AND, NOT and OR) as well as to incorporate functional dependencies and attribute correlations in Chapter 5.

3.2.2 Ranking Architecture

The similarity model (whose architecture is shown in Figure 3.1) forms the basis of our ranking framework. When the user U_i poses the query Q_j, the *query-similarity* model orders the queries in $\mathcal{Q}$ based on their individual similarity with respect to Q_j. Likewise, the *user-similarity* model orders[2] all users in $\mathcal{U}$ according to their similarity with U_i. Using

[2]For the sake of simplicity, we assume the orderings with respect to Q_j and U_i to be $\{Q_j, Q_1, Q_2, ..., Q_N\}$ and $\{U_i, U_1, U_2, ...U_M\}$ respectively; however, in practice, the individual queries and users may occur at different positions in the ordered lists.

these two ordered sets of similar queries and users, it searches the workload to identify the function $\mathcal{F}_{U_x}, Q_y$ such that the combination of U_x and Q_y is most similar to U_i and Q_j. $\mathcal{F}_{U_x}, Q_y$ is then used to rank Q_j's results for U_i.

The workload used in our framework comprises of ranking functions for several user-query pairs. Figure 3.2 shows the high level view of deriving an individual ranking function for a user-query pair (U_x, Q_y). By analyzing U_x's preferences (in terms of a selected set of tuples $R_{x,y}$) over the results (N_y), an approximate ranking function ($\mathcal{F}_{U_x}, Q_y$) can be derived. As our ranking function is a linear weighted-sum function, it is important that the mechanism used for deriving this function captures

1. the significance associated by the user to each attribute i.e., an *attribute-weight*, and
2. an user's emphasis on individual values of an attribute i.e., a *value-weight*.

These weights can then be integrated into a ranking function $\mathcal{F}_{x,y}$ to assign a *score* to every tuple t in N_y using Equation 3.1:

$$score(t) = \sum_{i=1}^{p} w_i * v_i \tag{3.1}$$

where w_i: *attribute-weight* of A_i, and v_i: *value-weight* for A_i's value in tuple t.

The workload $\mathbf{W}$ is populated using such ranking functions. Tables 3.1 and 3.2 show two instances of the workload[3] (represented in the form of a matrix of users and queries). Cell [x,y] in the workload, if defined, consists of the ranking function $\mathcal{F}_{x,y}$ for the user-query pair U_x and Q_y. We now elaborate on the proposed individual models of query and user similarity, and then present the holisitic similarity model for ranking.

[3]For Web databases, although the workload matrix can be extremely large, it is very sparse as obtaining preferences for large number of user-query pairs is practically difficult. We have purposely shown a dense matrix to make our model easily understandable.

Table 3.1. Sample *Workload-A*

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8
U_1	**??**	$\mathcal{F}_{1,2}$	–	–	$\mathcal{F}_{1,5}$	–	$\mathcal{F}_{1,7}$	–
U_2	$\mathcal{F}_{2,1}$	$\mathcal{F}_{2,2}$	–	$\mathcal{F}_{2,4}$	–	$\mathcal{F}_{2,6}$	$\mathcal{F}_{2,7}$	–
U_3	$\mathcal{F}_{3,1}$	$\mathcal{F}_{3,2}$	$\mathcal{F}_{3,3}$	$\mathcal{F}_{34}$	–	–	$\mathcal{F}_{3,7}$	–

Table 3.2. Sample *Workload-B*

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8
U_1	**??**	–	$\mathcal{F}_{1,3}$	–	$\mathcal{F}_{1,5}$	$\mathcal{F}_{1,6}$	$\mathcal{F}_{1,7}$	$\mathcal{F}_{1,8}$
U_2	$\mathcal{F}_{2,1}$	$\mathcal{F}_{2,2}$	–	$\mathcal{F}_{2,4}$	–	–	–	$\mathcal{F}_{2,8}$
U_3	$\mathcal{F}_{3,1}$	$\mathcal{F}_{3,2}$	–	$\mathcal{F}_{3,4}$	–	–	$\mathcal{F}_{3,7}$	$\mathcal{F}_{3,8}$

3.3 Query Similarity

For the sample *Workload-A* shown in Table 3.1, a ranking function does not exist for user U_1 asking query Q_1. However, it can be observed that ranking functions over queries Q_2, Q_5, Q_7 (the actual queries are shown in Table 3.3) have been derived; thus, forming U_1's workload. It would be useful to analyze if any of $\mathcal{F}_{1,2}$, $\mathcal{F}_{1,5}$, or $\mathcal{F}_{1,7}$ can be used for ranking Q_1's results for U_1. However, we know (from *Example-2*) that a user is likely to have displayed different ranking preferences for different query results. Consequently, a randomly selected function from U_1's workload is not likely to give a desirable ranking order over N_1. On the other hand, the ranking functions are likely to be comparable for queries similar to each other.

We advance the hypothesis that if Q_1 is most similar to a query Q_y (in U_1's workload), U_1 would display similar ranking preferences over the results of both these queries; thus, the ranking function ($\mathcal{F}_{1,y}$) derived for Q_y can be used to rank N_1. Similar to recommendation systems, our framework can utilize an aggregate function, composed from the functions corresponding to the top-K most similar queries to Q_1, to rank N_1. Although the results of our experiments showed that an aggregate function works well for certain

Table 3.3. Input query (Q_1) and U_1's Workload

QUERY	Make	Location	Price	Mileage	Color
Q_1	Honda	Dallas	any	any	any
Q_2	Toyota	Atlanta	any	any	any
Q_5	Lexus	Basin	any	any	any
Q_7	any	Little Rock	any	any	Grey

individual instances of users asking particular queries, on average, across all users asking a number of queries, using an individual function proved better than an aggregate function. Hence, for the reminder of the section, we only consider the most similar query (to Q_1). We translate this proposal of *query similarity* into a principled approach via two alternative models: i) *query-condition* similarity, and ii) *query-result* similarity.

3.3.1 Query-Condition Similarity

In this model, the *similarity* between two queries is determined by comparing the attribute values in the query conditions. Consider *Example-2* and the queries from Table 3.3. Intuitively, "Honda" and "Toyota" are vehicles with similar characteristics i.e., they have similar prices, mileage ranges, and so on. In contrast, "Honda" is a very different vehicle from "Lexus". Similarly, "Dallas" and "Atlanta", both being large metropolitan cities, are more similar to each other than "Basin", a small town in the state of Montana.

From the above analysis, Q_1 appears more similar to Q_2 than Q_5. In order to validate this intuitive similarity, we examine the relationship between the different values for each attribute in the query conditions. For this, we assume independence of schema attributes, since, availability of appropriate knowledge of functional dependencies and/or attribute correlations is not assumed[4].

[4]However, as elaborated later in Chapter 5, we do avail of the knowledge of functional dependencies and attribute correlations to further strengthen the similarity computations.

Definition Given two queries Q and Q', each with the conjunctive selection conditions, respectively of the form “WHERE A_1=a_1 AND $\cdots$ AND A_p=a_p” and “WHERE A_1=a'_1 AND $\cdots$ AND A_p=a'_p” (where a_i or a'_i is ‘any’[5] if A_i is not specified), the *query-condition* similarity between Q and Q' is given as the conjunctive similarities between the values a_i and a'_i for every attribute A_i (Equation 3.2).

$$similarity(Q, Q') = \prod_{i=1}^{p} sim(Q[A_i = a_i], Q'[A_i = a'_i]) \tag{3.2}$$

In order to determine the right-hand-side (RHS) for the above equation, it is necessary to translate the intuitive similarity between values (e.g., “Honda” is more similar to “Toyota” than it is with “Lexus”) to a formal model. This is achieved by determining the similarity between databases tuples corresponding to point queries with these attribute values. For instance, consider the values “Honda”, “Toyota” and “Lexus” for the attribute “Make”. The model generates three distinct queries (Q_H, Q_T and Q_L) with the conditions: “Make = Honda”, “Make = Toyota” and “Make = Lexus” respectively, and obtains the individual sets of results N_H, N_T and N_L (shown[6] in Tables 3.4, 3.5, and 3.6). It can be observed that the tuples for “Toyota” and “Honda” display a high degree of similarity over multiple attributes as compared to the tuples for “Lexus” indicating that the former two attribute values are more similar to each other than the latter. The similarity between each pair of query results (i.e., [N_H, N_T], [N_H, N_L], [N_T, N_L]) is then translated as the similarity between the respective pairs of attribute values[7].

[5]The value ‘any’ represents a union of all values for the domain of the particular attribute. For example, a value of ‘any’ for the Transmission attribute retrieves cars with ‘manual’ as well as ‘auto’ transmission.

[6]For the sake of readability, we have displayed only the top-3 query results returned by Google Base. Further, only five out of the eleven attributes from the Vehicle database schema are shown.

[7]The similarity between an attribute value (e.g., Honda) and the value ‘any’ is estimated as the average similarity between Honda and every other value in the domain of the corresponding attribute.

Table 3.4. Sample Results (N_H) For Query "Make = Honda"

TupleID	Make	Location	Price (in $)	Mileage	Color
t1	Honda	Dallas	20-25K	10-25K	red
t2	Honda	Atlanta	20-25K	25-50K	red
t3	Honda	Boston	25-30K	0-10K	green
...	...	...	...	...	...

Table 3.5. Sample Results (N_T) For Query "Make = Toyota"

TupleID	Make	Location	Price (in $)	Mileage	Color
t1	Toyota	Little Rock	5-10K	125-150K	red
t2	Toyota	Raleigh	15-20K	25-50K	green
t3	Toyota	Atlanta	20-25K	10-25K	silver
...	...	...	...	...	...

Table 3.6. Sample Results (N_L) For Query "Make = Lexus"

TupleID	Make	Location	Price (in $)	Mileage	Color
t1	Lexus	Boston	35-40K	0	silver
t2	Lexus	Detroit	35-40K	0	black
t3	Lexus	Urbana	30-35K	0-10K	red
...	...	...	...	...	...

Formally, we define the similarity between any two values a_1 and a_2 for an attribute A_i as follows. Two queries Q_{a_1} and Q_{a_2} with the respective selection conditions: "WHERE $A_i = a_1$" and "WHERE $A_i = a_2$" are generated. Let N_{a_1} and N_{a_2} be the set of results obtained from the database for these two queries. The similarity between a_1 and a_2 is then given as the similarity between N_{a_1} and N_{a_2}, and is determined using the variant of the *cosine-similarity* model proposed in [42]. Formally, given two tuples $T =< t_1, t_2, ..., t_m >$ in N_{a1} and $T' =< t'_1, t'_2, ..., t'_m >$ in N_{a2}, the similarity between T and T' is:

$$sim(T, T') = \sum_{i=1}^{p} sim(t_i, t'_i) \tag{3.3}$$

where,

$$sim(t_i, t'_i) = \begin{cases} 1 & \text{if } t_i = t'_i, \\ 0 & \text{if } t_i \neq t'_i. \end{cases} \tag{3.4}$$

It is obvious that Equation 3.4 will work improperly for numerical attributes where exact matches are difficult to find across tuple comparisons. In this work, we assume that numerical data has been discretized in the form of histograms (as done for query processing) or other meaningful schemes (as done by Google Base; shown for the values of 'price' and 'mileage' in Tables 3.4, 3.5 and 3.6). The existence of a (reasonable) discretization is needed for our model (instead of its justification).

Using Equation 3.3, the similarity between the two sets N_{a1} and N_{a2} (which in turn, corresponds to the similarity between the values a_1 and a_2) is estimated as the average pair-wise similarity between the tuples in N_{a1} and N_{a2} (Equation 3.5).

$$sim(N_{a1}, N_{a2}) = \frac{\sum_{i=1}^{|N_{a1}|} \sum_{j=1}^{|N_{a2}|} sim(T_i, T'_j)}{|N_{a1}| \cdot |N_{a2}|} \tag{3.5}$$

These similarities between attribute values can then be substituted into Equation 3.2 to estimate *query-condition* similarity.

3.3.2 Query-Result Similarity

In this model, *similarity* between a pair of queries is evaluated as the similarity between the tuples in the respective query results. The intuition behind this model is that if two queries are similar, the results are likely to exhibit greater similarity.

For the queries in Table 3.3, let the results shown in Tables 3.7, 3.8, and 3.9, respectively, correspond to a sample set (again top-3 results and five attributes displayed) for Q_1, Q_2 and Q_5. We observe that there exists certain similarity between the results of Q_1 and Q_2 for attributes such as 'price' and 'mileage' (and even 'color' to a certain extent). In

Table 3.7. Sample Results Of Q_1: “Make=Honda AND Location = Dallas,TX”

TupleID	Make	Location	Price (in \$)	Mileage	Color
t1	Honda	Dallas	15-20K	25-50K	red
t2	Honda	Dallas	15-20K	25-50K	red
t3	Honda	Dallas	20-25K	0-10K	green
...	...	...	...	...	...

Table 3.8. Sample Results Of Q_2: “Make=Toyota AND Location = Atlanta,GA”

TupleID	Make	Location	Price (in \$)	Mileage	Color
t1	Toyota	Atlanta	15-20K	25-50K	red
t2	Toyota	Atlanta	15-20K	25-50K	green
t3	Toyota	Atlanta	20-25K	10-25K	silver
...	...	...	...	...	...

Table 3.9. Sample Results Of Q_5: “Make=Lexus AND Location = Basin,MT”

TupleID	Make	Location	Price (in \$)	Mileage	Color
t1	Lexus	Basin	35-40K	new	silver
t2	Lexus	Basin	35-40K	new	black
t3	Lexus	Basin	30-35K	0-10K	red
...	...	...	...	...	...

contrast, the results of Q_5 are substantially different; thus, allowing us to infer that Q_1 is more similar to Q_2 than Q_5. Formally, we define *query-result similarity* below.

Definition Given two queries Q and Q', let N and N' be their query results. The *query-result* similarity between Q and Q' is then computed as the similarity between the result sets N and N', given by Equation 3.6.

$$similarity(Q, Q') = sim(N, N') \tag{3.6}$$

The similarity between the pair of results (N and N') is estimated using the Equations 3.3, 3.4 and 3.5 defined earlier.

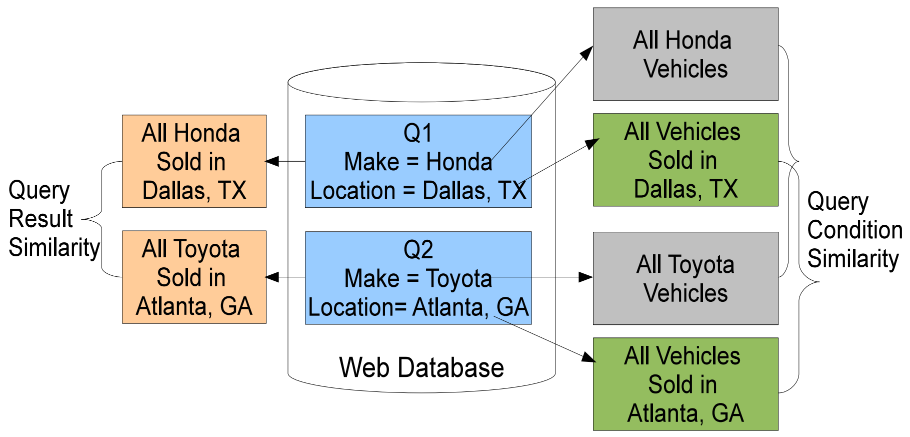

All Honda Vehicles
All Vehicles Sold in Dallas, TX
All Toyota Vehicles
All Vehicles Sold in Atlanta, GA
Query Condition Similarity
All Honda Sold in Dallas, TX
All Toyota Sold in Atlanta, GA
Query Result Similarity
Q1 Make = Honda Location = Dallas, TX
Q2 Make = Toyota Location= Atlanta, GA
Web Database

Figure 3.3. Summarized View Of The *Query Similarity* Models.

3.3.3 Analysis Of The Proposed Query Similarity Models

The computation of similarity for the two models discussed above is summarized in Figure 3.3. While the *query-condition similarity* uses the conjunctive equivalence between individual attribute values, the *query-result similarity* performs a comparison between the actual sets of query results. Below, we discuss the accuracy and computational efficiency of these models.

Accuracy: Intuitively, similarity between queries depends on the proximity between their respective query conditions. For instance, “Honda” and “Toyota” are cars with a lot of common features which reflects in the tuples representing these values in the database, and hence, queries that search for these vehicles are more similar than queries asking for “Lexus”. In contrast, two queries may return very similar results although their conditions could be quite dissimilar. For example, the following two queries on Google Base – “Make = Mercedes AND Color = Lilac”, and “Location = Anaheim, CA AND Price > $ 35,000”, end up returning exactly the same set of results. Although these are very similar from the *query-result similarity* definition (a corresponding value of 1.000 is obtained by applying Equation 3.6 over these two queries), the queries, in fact, are intuitively not similar (which is vindicated by a value of 0.0031 yielded by applying Equation 3.2 i.e., the query-condition similarity model over the same two queries).

In general, similar queries are likely to generate similar results; however, the converse is not necessarily true. Hence the *query-condition similarity* model is expected to be more accurate and consistent than the *query-result similarity* model which is also borne out by the experiments.

Computational Efficiency: Both query similarities can be computed by applying queries over the Web database i.e., direct access to the data is not needed, which can be difficult for Web database such as Google Base (a collection of multiple databases). One alternative toward efficiently implementing these similarities would be to pre-compute them and use the respective values at query time[8].

In order to distinguish the two models, consider the workload $\mathbf{W}$ and assume a schema of p attributes. For the sake of simplicity, let the domain of each attribute have n values. The *query-condition* model relies purely on the pairwise similarities between attribute values, and hence can be independently pre-computed (generating $p * n$ queries, one for each value of every attribute, and performing $p * n^2$ computations). At query-time, the similarity between input (Q_i) and every query in $\mathbf{W}$ can be estimated by a lookup of the attribute-value similarities. In contrast, the *query-result* model requires a comparison between the query results. Since an input query cannot be predicted, pre-computation is not possible (unless every possible query on the database can be generated and compared – a combinatorial explosion scenario). Even if we assume that a large set of potential queries (instead of every possible query) can be generated, the *query-result* model requires that the entire set of results for all these queries be stored, in order to establish similarity with the

[8]Pre-computation assumes that the underlying data does not change significantly over time. Considering the size of these databases, small percent of changes are unlikely to affect the similarity computations; however it is possible to re-compute the similarities periodically.

results of an incoming query; thus, leading to a extremely large overhead in terms of storage. In contrast, the *query-condition* model simply requires a singular value to be stored for every pair of attribute values, and hence, incurs a significantly less cost of storage as compared to its counterpart model.

Thus, intuitively, the *query-condition* model seems superior to the *query-result* model. Also, computation of similarity by the *query-condition* model is tractable as compared to the *query-result* model. Furthermore, both these observations are substantiated by our experiments as well.

3.4 User Similarity

For the sample *Workload-A* shown in Table 3.1, a function ($\mathcal{F}_{1,1}$) is needed to rank the results of Q_1 for U_1. Observing this workload, functions $\mathcal{F}_{2,1}$ and $\mathcal{F}_{3,1}$ have been derived for users U_2 and U_3 with respect to Q_1. It would be useful to determine if either of these functions can be used in place of $\mathcal{F}_{1,1}$ to yield an acceptable ranking order of N_1 for U_1. However, we know from *Example-2* that different users may display different ranking preferences towards the same query. Thus, we advance the notion of *user similarity* to determine an appropriate ranking function; instead of randomly picking one from $\mathbf{W}$.

We put forward the hypothesis that if U_1 is similar to an existing user U_x, then, for the results of a given query (say Q_1), both users will show similar ranking preferences; therefore, U_x's ranking function ($\mathcal{F}_{x,1}$) can be used to rank Q_1's results for U_1 as well. Again, as alluded to in Section 3.3 for query similarity, instead of using a *single* most similar user (to U_1), our framework can be extended to determine the top-K set of most similar users to establish *user-similarity* as well. However, like *query-similarity*, an aggregate ranking function did not provide significant improvement in the ranking quality; hence, we only consider the most similar user (to U_1) in our discussion.

In order to translate the hypothesis of *user-similarity* into a model, we need to understand how to compute similarity between a given pair of users. In this work, we propose a novel methodology for determining this similarity by *only* considering the set of respective ranking functions obtained over the **common** queries asked by these users.

We would also like to point out that since this framework is based on the notion of similarity, we could have used the same notion of similarity employed in recommender systems i.e., based on user profiles. However, our premise was that the profile-based systems provide *static* or *context-independent* similarity (i.e., a users behavior does not change across all items/objects) whereas we believe that ranking of results (unlike recommending) varies substantially for the same user based on the query type (i.e., is context-dependent) and attributes specified (as elaborated by *Example-2* in Section 3.1). This work can thus, be treated as a generalization that uses information beyond profiles and seems appropriate for the Web database context. Furthermore, in Chapter 5, we present a combined model that combines the dynamic component presented here, with a static component based on user profiles, to facilitate a comprehensive computation of user similarity between any pair of arbitrary users. Formally,

Definition Given two users U_i and U_j having specified the same set of common[9] queries: $\{Q_1, Q_2, ..., Q_r\}$ over the Web database, for which ranking functions ($\{\mathcal{F}_{i,1}, \mathcal{F}_{i,2}, ..., \mathcal{F}_{i,r}\}$ and $\{\mathcal{F}_{j,1}, \mathcal{F}_{j,2}, ..., \mathcal{F}_{j,r}\}$) exist in $\mathbf{W}$, the *user similarity* between U_i and U_j is computed as the average similarity between their individual ranking functions for each common query (and is shown in Equation 3.7):

$$similarity(U_i, U_j) = \frac{\sum_{p=1}^{r} sim(\mathcal{F}_{i,p}, \mathcal{F}_{j,p})}{r} \tag{3.7}$$

[9]Without loss of generality, we assume $\{Q_1, Q_2, ..., Q_r\}$ are the common queries for U_i and U_j, although they can be any queries.

In order to determine the right-hand-side of the Equation 3.7, it is necessary to quantify a measure that establishes the similarity between a given pair of ranking functions. We use the *Spearman's rank correlation coefficient (ρ)* [68] to compute similarity between the sets obtained by applying these ranking functions on the query results. We choose the *Spearman coefficient* based on the observations of the survey conducted by [69] regarding its usefulness, with respect to other metrics, in comparing ranked lists.

Accordingly, using the Spearman coefficient, we establish similarity between a pair of ranking functions as follows: Consider two functions $\mathcal{F}_{i,1}$ and $\mathcal{F}_{j,1}$ respectively derived for users U_i and U_j, for the same query Q_1. We apply these two functions individually on N_1 (i.e., the results of Q_1) to obtain two ranked sets of results – $N_{R_{i1}}$ and $N_{R_{j1}}$. If the number of tuples in the result sets is $\mathbf{N}$, and d_i is the difference between the ranks of the same tuple (t_i) in $N_{R_{i1}}$ and $N_{R_{j1}}$, then we express the similarity between $\mathcal{F}_{i1}$ and $\mathcal{F}_{j1}$ as the *Spearman's rank correlation coefficient* given by Equation 3.8:

$$sim(F_{i1}, F_{j1}) = 1 - \frac{6 * \sum_{i=1}^{\mathbf{N}} d_i^2}{\mathbf{N} * (\mathbf{N}^2 - 1)} \tag{3.8}$$

The right hand side of Equation 3.8 can then be plugged into Equation 3.7 to estimate the overall similarity between the given pair of users.

So far in our method for estimating *user similarity*, we have considered **all** the queries that are common to a given pair of users. This assumption forms one of our models for user similarity termed ***query-independent*** *user similarity*. However, it might be useful to estimate user similarity based on only those queries that are similar to the input query Q_1. In other words, in this hypothesis, two users who may not be very similar to each other over the *entire workload* comprising of similar and dissimilar queries, may in fact, be very similar to each other over a smaller set of *similar queries*. We formalize this hypothesis

Table 3.10. Drawbacks Of Query-Independent User Similarity

For Query	User Similarity
Q_2	sim(U_1, U_2) $>$ sim(U_1, U_3)
Q_7	sim(U_1, U_2) $<$ sim(U_1, U_3)
Q_2, Q_7	sim(U_1, U_2) $<$ sim(U_1, U_3)

using two different models – i) ***clustered***, and ii) ***top-K*** – for determining *user similarity*. We now elaborate these proposed models.

3.4.1 Query-Independent User Similarity

This model follows the simplest paradigm and estimates the similarity between a pair of users based on ***all*** the queries common to them. For instance, given *Workload-A* in Table 3.1, this model determines the similarity between U_1 and U_2 using the ranking functions of Q_2 and Q_7. From the queries in Table 3.3, let the *query-similarity* model indicate that Q_2 is *most* similar to Q_1 whereas Q_7 is *least* similar to Q_1, and let us consider the *user-similarity* results be as shown in Table 3.10.

This model will pick U_3 as the most similar user to U_1. However, if only Q_2 (which is most similar to Q_1) is used, U_2 is more similar to U_1. Based on our premise that similar users display similar ranking preferences over the results of similar queries, it is reasonable to assume that employing $\mathcal{F}_{2,1}$ to rank Q_1's results would lead to a better ranking order (from U_1's viewpoint) than the one obtained using $\mathcal{F}_{3,1}$. The failure to distinguish between similar and dissimilar queries is thus a potential drawback of this model, which the following models aim to overcome.

3.4.2 Cluster-Based User Similarity

In order to meaningfully restrict the number of queries that are similar to each other, one alternative is to *cluster* queries in the workload based on query similarity. This can be done using a simple *K-means* clustering method [70]. Given the set $\mathcal{Q}$: $\{Q_1$, Q_2, ... $Q_N\}$, each query (Q_j) is represented as a N-dimensional vector of the form $<s_{j,1}, ..., s_{j,N}>$ where $s_{j,p}$ represents the *query-condition similarity* score between the queries Q_j and Q_p (by Equation 3.2). Using *K-means*, we cluster these "N" queries into K clusters based on a pre-defined K and a pre-determined number of iterations.

Consider *Example-1* and the queries in Table 3.3. Assuming the similarities specified in Section 3.4.1 (i.e., Q_2 and Q_7 are most and least similar to Q_1 respectively), for a value of $K = 2$, the *simple K-means* algorithm will generate two clusters – C_1 containing Q_1 and Q_2 (along with other similar queries), and C_2 containing Q_7 (in addition to other queries not similar to Q_1). We then estimate the similarity between U_1 and every other user only for the cluster C_1 (since it contains queries most similar to the input query Q_1). Using the scenario from Table 3.10, U_2 would be chosen as the most similar user and $\mathcal{F}_{2,1}$ would be used to rank the corresponding query results.

The above model assumes that ranking functions are available for reasonable number of queries in each cluster. However, as the workload is likely to be sparse for most Web databases, it is possible that no ranking functions are available in the cluster most similar to the incoming query. For example, considering the *Workload-B* in Table 3.2 and assuming a cluster C_1 of queries Q_1, Q_2, Q_3 and Q_4, due to the lack of ranking functions, no similarity can be established between U_1 and other users. Consequently, the similarities would then be estimated in other clusters, thus hampering the quality of the ranking achieved due to dissimilarity between the input query and the queries in the corresponding clusters.

A well-established drawback of using a cluster-based alternative is the choice of K. In a Web database, a small value of K would lead to a large number of queries in every cluster, some of which may not be very similar to the rest, thus, affecting the overall user similarity. In contrast, a large value of K would generate clusters with few queries, and in such cases, the probability that there exist no users with any function in the cluster increases significantly. However, to use this model, approaches developed for finding an appropriate K may be used (as this issue is common to most clustering approaches and has been studied extensively in the literature).

3.4.3 Top-K User Similarity

Instead of finding a reasonable K for clustering, we propose a refinement, termed *top-K user similarity*. We propose three measures to determine *top-K* queries that are most similar to an input query (say Q_1 from *Example-1*), and estimates the similarity between the user (U_1) and every other user.

Strict top-K user similarity: Given an input query Q_1 by U_1, *only* the *top-K* most similar queries to Q_1 are selected. However, the model does not check the presence (or absence) of ranking functions in the workload for these K queries. For instance, based on assumption in Section 3.4.2, and using $K = 3$, the three queries most similar to Q_1 are Q_2, Q_3 and Q_4, and hence, would be selected by this model.

In the case of *Workload-A*, similarity between U_1 and U_2 as well as between U_1 and U_3 will be estimated using Q_2. However, in the case of *Workload-B*, similar to the problem in the *clustering* alternative, there is no query common between U_1 and U_2 (as well as U_3). Consequently, similarity cannot be established and hence, no ranking is possible.

User-based top-K user similarity: In this model, we calculate *user similarity* for a given query Q_1 by U_1, by selecting *top-K* most similar queries to Q_1, each of which has a ranking function for U_1. Consequently for *Workload-A*, using $K = 3$, the queries Q_2, Q_5 and Q_7 would be selected. Likewise, in the case of *Workload-B*, this measure would select Q_3, Q_5 and Q_6 using the 'top-3' selection. However, since there exist no function for users U_2 and

U_3 (in *Workload-B*) given these queries, no similarity can be determined, and consequently, no ranking would be possible.

Workload-Based top-K user similarity: In order to address the problems in previous two models, we propose a *Workload-Based top-K* model that provides the stability of the *query-independent model* (in terms of ensuring that ranking is *always* possible, assuming there is at least one non-empty cell in the workload for that user) and ensures that similarity between users can be computed in a *query-dependent* manner.

Given a Q_1 by U_1, the *top-K* most similar queries to Q_1 are selected such that for each of these queries, there exists: i) a ranking function for U_1 in the workload, and ii) a ranking function for *at least one* other user (U_i) in the workload. Considering $K = 3$, this model will select Q_2, Q_5 and Q_7 in the case of *Workload-A* and the queries Q_7 and Q_8 for *Workload-B*, and ensure a ranking of results in every case.

3.4.4 Summary Of User Similarity Models

Intuitively, *query-dependent* estimation of user similarity is likely to yield better ranking as compared to the *query-independent* model. Also, barring the *workload-based top-K* model, ranking may not *always* be possible in other *query-dependent* models. In order to overcome this, the framework can always dynamically choose the top-K queries at runtime in the following order: *strict*, *user-based*, and *workload-based*. The *strict top-k* gives the best results as can be understood intuitively and has been ascertained experimentally. The alternatives extend the availability of ranking functions with reduced accuracy. Additionally, as our experiments show, the choice of *K* in *top-K* is not as critical as the selection of *K* in *K-means* algorithm. Thus, the *top-K* model seems to be the best alternative in a query-dependent environment.

Further, we would like to point out that either of the proposed models cannot be applied in situations when the pair of users being compared have absolutely no queries common between them – a realistic scenario in the context of Web database applications involving new and/or infrequent users. Consequently, no ranking is possible in such cases. In order to support a consistent model of *user-dependent* ranking uniformly across all types of users (new, infrequent as well as frequent), we propose an extension that unifies the proposed model, based on similarity of ranking functions, with a static model (that determines similarity based on user profiles i.e., the ones adopted by recommender and other database systems). We elaborate the details of such an unified model in Chapter 5.

3.5 The Holistic Similarity Model

In order to derive a user's (U_i) ranking function for a query (Q_j), we have proposed two independent approaches based on *user* and *query* similarity. However, given the scale of Web users and queries, and the sparseness of the workload, applying only one model may not be the best choice at all times.

Considering *Example-1* and *Workload-B* (Table 3.2), we want to identify a ranking function to rank Q_1's results for U_1. Using only the *query-similarity* model, $\mathcal{F}_{1,3}$ will be selected since Q_3 is most similar to Q_1. In contrast, applying only *user-similarity* model will yield $\mathcal{F}_{2,1}$ as U_2 is most similar to U_1. It would be meaningful to rank these functions ($\mathcal{F}_{1,3}$ and $\mathcal{F}_{2,1}$) to choose the most appropriate one. Furthermore, in a more practical setting, the workload is likely to have a ranking function for a similar query (to Q_1) derived for a similar user (to U_1). For instance, the likelihood of $\mathcal{F}_{2,2}$ existing in the workload would be higher than the occurrence of either $\mathcal{F}_{1,3}$ or $\mathcal{F}_{2,1}$. Hence, it would be meaningful to combine the two measures into a single *Similarity Model.*

The goal of this composite model is to determine a ranking function ($\mathcal{F}_{x,y}$) derived for the most similar query (Q_y) to Q_j given by the most similar user (U_x) to U_i to rank Q_j's results. The process is performed as shown in Algorithm 1.

```
   Input: $U_i$, $Q_j$, $\mathcal{Q}, \mathcal{U}$, W
   Output: $\mathcal{F}_{x,y} \in$ W
 1 foreach $Q_a \in \mathcal{Q}$ do
 2     Calculate $similarity(Q_j, Q_a)$
 3 end
 4 $\mathcal{Q}_S$: sort($\mathcal{Q}$) // based on similarity with $Q_j$
 5 foreach $U_b \in \mathcal{U}$ do
 6     Calculate $similarity(U_i, U_b)$
 7 end
 8 $\mathcal{U}_S$: sort($\mathcal{U}$) // based on similarity with $U_i$
 9 foreach $Q_p \in \mathcal{Q}_S$ do
10     foreach $U_q \in \mathcal{U}_S$ do
11         Rank($U_q$,$Q_p$) = Rank($U_q$) + Rank($Q_p$)
12     end
13 end
14 $\mathcal{F}_{x,y}$ = Get-RankingFunction( )
```

Algorithm 1: Deriving Ranking Functions From **W**

The input to the algorithm is a user (U_i) and a query (Q_j) along with the set of queries ($\mathcal{Q}$), users ($\mathcal{U}$) and the workload (**U**) comprising of ranking functions. The algorithm begins by determining the *query-condition similarity* (using Equation 3.2) between Q_j and every query in $\mathcal{Q}$ (**Steps 1–3**). Likewise, it determines the *top-K user similarity* (using Equation 3.7) between U_i and every user in $\mathcal{U}$ (**Steps 5–7**). The queries in $\mathcal{Q}$ and the users in $\mathcal{U}$ are then sorted based on their similarities with respect to Q_j and U_i respectively (**Steps 4 & 8**). Correspondingly, each user-query pair (obtained from the two sorted lists) receives a rank which is an aggregate of their respective ranks in the sorted lists (**Steps 9–13**). For instance, if U_q and Q_p occur as the q^{th} and p^{th} elements in the respective ordered lists, the pair (U_q, Q_p) are assigned an aggregate rank (in this case, a rank of "$q + p$" will be assigned). The "Get-RankingFunction" method (**Step 14**) then selects a ranking function $\mathcal{F}_{x,y}$ such that the following two conditions are satisfied:

1. *Condition-1*: $\mathcal{F}_{x,y} \in \mathbf{W}$, and
2. *Condition-2*: rank(U_x) + rank(Q_y) is **minimum**

The first condition determines that a ranking function does exist for a pair to be selected, whereas the second condition ensures that the selected function belongs to the most similar query (to Q_j) and the most similar user (to U_i).

Algorithm 1 only displays a higher-level view of finding a suitable function using the *Similarity Model*. However, the process of estimating *query similarities*, *user similarities* as well as traversing the workload to select an appropriate ranking function can be computationally costly. Applying appropriate indexing techniques where estimating *query similarity* can be reduced to a simple lookup of similarities between attribute value-pairs, pre-computation of *user similarity* to maintain an indexed list of similar users for every user, and maintaining appropriate data structures to model the matrix traversal as a mere lookup operation are some of the preliminary approaches that we adopted to establish a workable ranking framework. We discuss some of the preliminary results of efficiently using our framework in Section 3.6. Although there is great scope for devising techniques to make the system scalable and efficient, the focus of this work was to propose techniques to establish good ranking quality.

3.6 Experimental Evaluation

We have evaluated each proposed model (*query-similarity* and *user-similarity*) in isolation, and then compared both these models with the *combined* model for *quality/accuracy*. We also evaluated the *efficiency* of our ranking framework.

Ideally, we would have preferred to compare our approach against existing ranking schemes in databases. However, what has been addressed in literature is the use of exclusive profiles for user-based ranking (the techniques for the same do not distinguish between queries) or the analysis of the database in terms of frequencies of attribute values for query-dependent ranking (which does not differentiate between users). In the context of Web databases like Google Base, the data is obtained on-the-fly from a collection of data sources; thus, obtaining the entire database for determining the individual ranking functions, for comparing with query-dependent ranking techniques, is difficult. Even if we obtain ranking functions for different queries, all users will see the same ranking order for a given query. Thus, comparing such static ordering of tuples against our approach (that determines distinct ranking of tuples for each user and query separately) would not be a meaningful/fair comparison. Likewise, we felt that the comparing only static user profiles (that ignore the different preferences of the same user for different queries) to our proposed definition of user similarity, for user-dependent ranking will not be fair. Hence we have tried to compare the proposed user, query, and combined similarities to indicate the effectiveness of each model with respect to the other two models.

Further, the context of recommendation systems is different from the one considered in this paper; hence, the direct application of these techniques for comparing against our framework was not feasible. We would also like to point out that unlike information retrieval, there are no standard benchmarks available and hence, we had to rely on controlled user studies for evaluating our framework.

3.6.1 Setup

We used two real Web databases provided by Google Base. The first is a *vehicle* database comprising of 8 categorical/discretized attributes (Make, Vehicle-Type, Mileage, Price, Color, etc.). Although Price and Mileage are numeric, they are discretized a priori by Google into meaningful ranges. In addition, for every attribute, the domain of values (e.g., ‘Chevrolet’, ‘Honda’, ‘Toyota’, ‘Volkswagen’, ... for the Make attribute) is provided. The second is a *real estate* database with 12 categorical/discretized attributes (Location, Price, House Area, Bedrooms, Bathrooms, etc.). Google provides APIs for querying its databases, and returns a maximum of 5000 results for every API query. Our experiments were performed on a Windows XP Machine with a 2.6 GHz Pentium 4 processor and 4 GB RAM. All algorithms were implemented in Java.

3.6.2 Workload Generation And User Studies

Our framework utilizes a workload comprising of users, queries and ranking functions (derived over a reasonable set of user queries). Currently, user and query statistics are not publicly available for any existing Web databases. Furthermore, establishing a real workload of users and queries for a Web database would require significant support from portals supporting Web databases such as Yahoo or Google – a task beyond the scope of this dissertation. Hence, to experimentally validate the quality of our framework, we had to rely on controlled user studies for generating the workload. For each database, we initially generated a pool of 60 random queries (comprising of conditions based on randomly selected attributes and their values), and manually selected 20 representative queries that are likely to be formulated by real users. Tables 3.11 and 3.12 show three such queries over each database. The users in these experiments comprised of a combination of – real Web users from Amazon Mechanical Turk (`www.mturk.com`), and graduate students and faculty members from the University of Texas at Arlington.

Table 3.11. Sample Experimental Queries: ***Vehicle*** Database

Q_1	"Make = Honda AND Location = Dallas,TX AND Price $<$ \$ 10,000"
Q_2	"Make = Toyota AND Location = Miami,FL AND Price $<$ \$ 8,000"
...	...
Q_{10}	"Location = Chicago,IL AND Mileage $<$ 100,500 AND Year $>$ 2004"
...	...

Table 3.12. Sample Experimental Queries: ***Real Estate*** Database

Q_1	"Location = Dallas,TX AND Beds = 3 AND To = Buy"
Q_2	"Location = Houston, TX AND Beds = 2 AND Bath = 2 AND To = Buy"
...	...
Q_{16}	"Location = Boston,MA AND Type = Townhouse AND To = Rent"
...	...

We then conducted two separate surveys (one for each database) where every user was shown 20 queries (one-at-a-time) and asked to input, for each query, a ranking function by assigning weights to the schema attributes (on a scale of 1 to 10). For aiding the user in expressing these preferences, we also displayed the set of results returned for each query. In reality, collecting functions explicitly from users for forming the workload is far from ideal; however, the focus of this work was on using, instead of establishing, a workload for performing similarity-based ranking. Although generating a larger set of queries would have been ideal for testing the framework, asking users to interact with more than 20 queries would have been difficult. We would also like to indicate that as is the case with most user studies in the domain of databases, obtaining a large number of users and queries to participate in the corresponding survey is difficult and hence, these numbers are typically very small (as seen by the user surveys conducted for database ranking in [43] [44] [36]).

Each *explicit* ranking provided by a user for a particular query was then stored in the associated workload $\mathbf{W}$. The *vehicle* database survey was taken by 505 users (i.e., 450

Mechanical Turk users and 55 student/faculty members) whereas the *real estate* database survey was taken by 525 users who provided ranking functions for all the queries displayed to them. Thus, we generated a workload of 10,100 ranking functions for the *vehicle* database and 10,500 functions for the *real estate* database.

It is evident that more the number of functions in the workload, better will be the quality of ranking achieved (since more similar queries and users would be found for whom functions exist in the workload). This hypothesis was further validated when we achieved a better ranking quality when 50% of the workload was filled (i.e., 50% of the functions were masked out) as compared to the one achieved when the workload contains 25% and 10% of the total ranking function. However, for most Web databases, the workloads would generally be very sparse since obtaining ranking functions across a large number of user-query pairs would be practically difficult. In order to show the effectiveness of our model for such scenarios, we present the results when ranking functions exist for only 5% workload (i.e., 95% of the functions are masked out). Hence, for the rest of the section, we consider the workload for the *vehicle* database consisting only 505 (5% of 10,100) ranking functions, and the *real estate* database comprising 525 ranking functions.

3.6.3 Quality Evaluation

We first present the results for individual similarity models before explaining the results for the combined model.

3.6.3.1 Evaluation Of Query Similarity

Based on the two proposed models of *query similarity* (Section 3.3.1 and 3.3.2), in the absence of a function $\mathcal{F}_{i,j}$ for a user-query pair (U_i, Q_j), the most similar query (Q_c and Q_r using the query-condition and the query-result model respectively) asked by U_i, for which a function ($\mathcal{F}_{i,c}$ and $\mathcal{F}_{i,r}$ respectively) exists in the workload, is selected and the corresponding function is used to rank Q_j's results.

We test the quality of both query similarity models as follows: We rank Q_j's results (N_j) using $\mathcal{F}_{i,c}$ (obtained from the *query-condition* model) and $\mathcal{F}_{i,r}$ (derived using the *query-result* model) respectively, and obtain two sets of ranked results (R' and R''). We then use the original (masked) function $\mathcal{F}_{i,j}$ to rank N_j and obtain the set (R). Since R represents the true ranking order provided by U_i for Q_j, we determine the quality of these models by computing the *Spearman rank correlation coefficient* (Equation 3.8) between R and R', and between R and R''. If the coefficients obtained are high (nearing 1.0), it validates our hypothesis (that for similar queries, the same user displays similar ranking preferences). Furthermore, if the coefficient between R and R' is greater than the one between R and R'', our understanding that *query-condition* model performs better than the *query-result* model is validated.

We performed the above process for each user asking every query. Figure 3.4 shows, for both databases, the *average query-condition* similarity (as well as the *average query-result* similarity) obtained across every query. The horizontal axis represents the queries; whereas the vertical axis represents the average value of the resulting *Spearman coefficient*. As the graph shows, over both the domains, the *query-condition* model outperforms the *query-result* model.

The graphs indicate that the comparative loss of quality (highest value of Spearman coefficient being 0.95 for query 5) is due to the restricted number of queries in the workload. Although finding a similar query (for which a ranking function is available) for a workload comprising of 20 queries and only 5% of ranking functions is difficult, the results are very encouraging. Based on the results of these experiments, we believe that the *query similarity* model would perform at an acceptable level of quality even for large, sparse workloads.

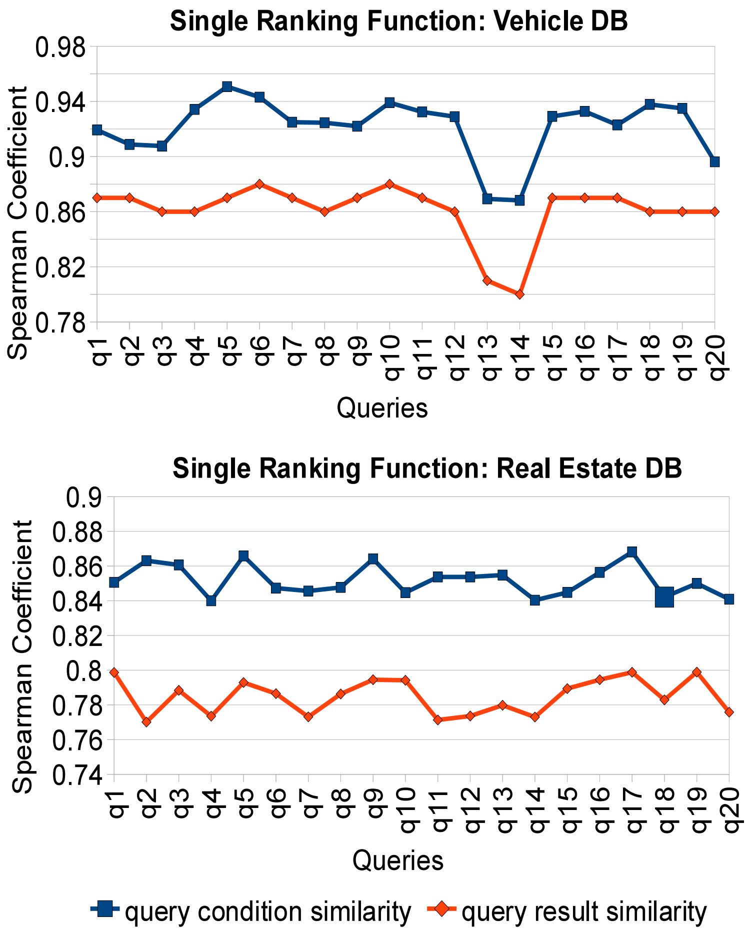


Figure 3.4. Quality Of Query Similarity: Single Ranking Function.

We further tested this model for comparing the quality produced by applying an *aggregate* function (i.e., selecting the top-K similar queries and combining their respective functions) instead of using the function of the most similar query. Since the ranking function in our framework is a linear-weighted sum function, the aggregate function is represented using a combination of *average* attribute-weights and *average* value-weights (wherein the *average* values are obtained by aggregating the respective weights over the top-K most similar queries provided by either of the two models, and then, dividing each of these values by K). In this set of experiments, we varied the values of K from 2, 4, 5

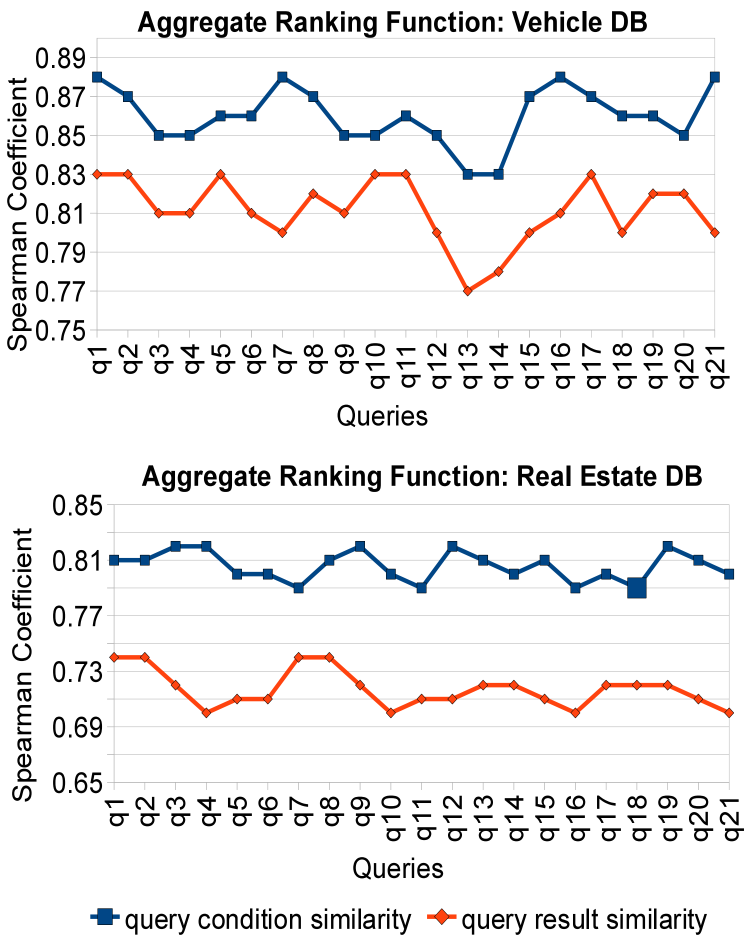


Figure 3.5. Quality Of Query Similarity: Aggregate Ranking Function.

and 10. For a value of $K = 5$, the *query-condition-similarity* model produced an overall average value of 0.86 for the Spearman coefficient (versus the 0.91 obtained if a single function of the most similar query is used) for the *vehicle* database. Similarly, a value of 0.83 was obtained for the *query-result-similarity* model (versus the 0.86 obtained for a single function). A similar trend was observed for the *real estate* database as well. Figure 3.5 shows the results of applying such an aggregate function (over top-5 queries) for all users asking every query over both databases.

3.6.3.2 Evaluation Of User Similarity

We validate the quality of all versions of the *User Similarity* models, described in Section 3.4, as follows: Using the original function given by a user U_i for Q_j, we obtain a ranked set of results R. Then we determine, for all the proposed models, the corresponding user U_l most similar to U_i having function $\mathcal{F}_{lj}$ in the workload. Using the corresponding function for each case, we get the ranked set of results (i.e., R_i for *query-independent*, R_c for *clustered*, R_s for *strict top-K*, R_u for *user-based top-K*, and R_w for *workload-based top-K*) and compute the Spearman coefficient between each of these ranked sets and R. In our experiments, we set a value of $K = 5$ for the *K-means* algorithm, and a value of $K = 2$ was chosen for the *top-K* models as our workload (in the number of queries) is small.

Figure 3.6 shows the average ranking quality individually achieved for the *vehicle* as well as *real estate* database, across all queries for all users taking the survey. Our results clearly show that the *strict top-K* model performs consistently better than the rest.

However, as Figure 3.7 shows, the *strict top-K* (as well as the *clustered* and *user top-K*) fail to find functions[10] for several queries (shown for a randomly selected user U_{27}). Figure 3.8 further confirms this fact by comparing the different models, in terms of their ability to determine ranking functions, across the entire workload. In spite of this, the accuracy of the *strict top-K model* is superior to all other models when a ranking function can be identified. Only when a ranking function cannot be found, using the *strict top-K* model does not make sense. We have proposed a suit of models (such as *clustered*, *user top-k*, *workload-based top-k* as well as *query-independent*) precisely for this reason and together will cover all the scenarios encountered in a sparse workload to provide a meaningful ranking function.

[10]In the graph shown in Figure 3.7, the *user-based* and *workload-based* top-K models produce identical results since the number of queries in the workload is very small. Hence, the plots for these two models overlap and hence, cannot be distinguished in the graph.

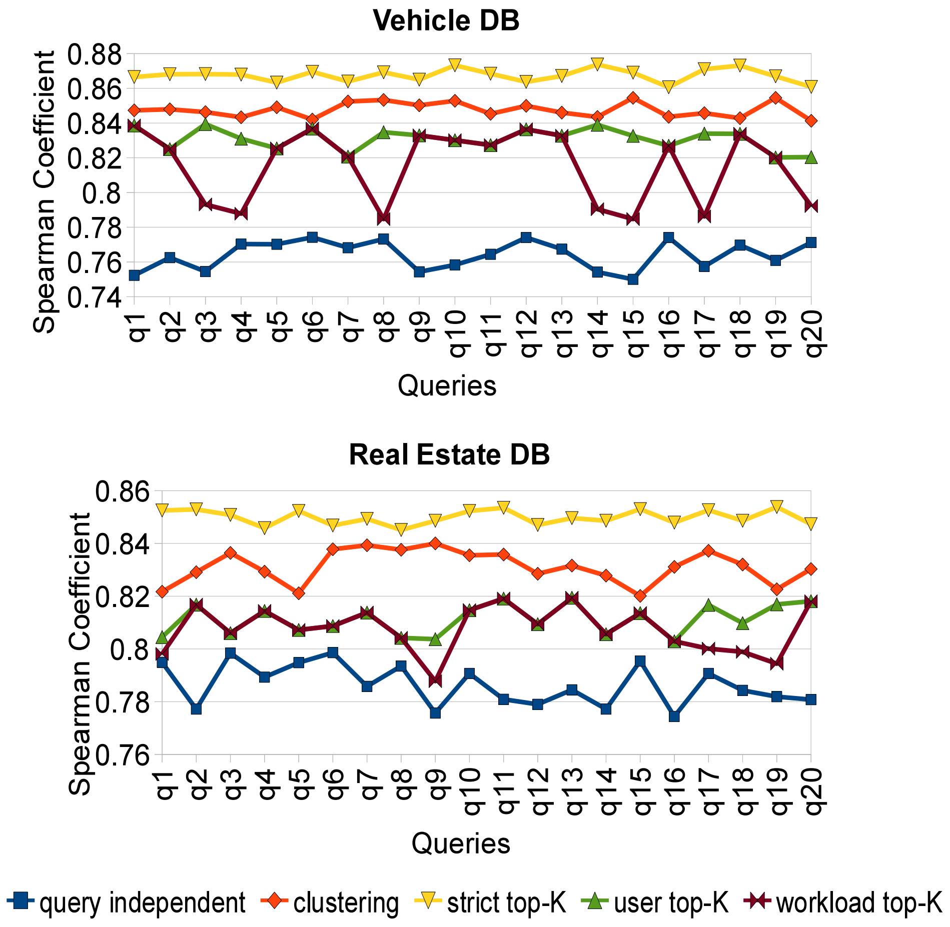


Figure 3.6. Quality Of User Similarity.

Similar to the *Query Similarity* model, using an aggregate function (i.e., derived by combining functions of top-K most similar users) did not provide any improvement in the quality of ranking than the one achieved by using a single function of the most similar user; hence, we do not include the details of these experiments.

3.6.3.3 Evaluation Of The Holistic Similarity Model

Finally, we evaluated the quality of the *holistic similarity model* using the steps described in Algorithm 1. Figures 3.9 and 3.10 show the average quality of this model for

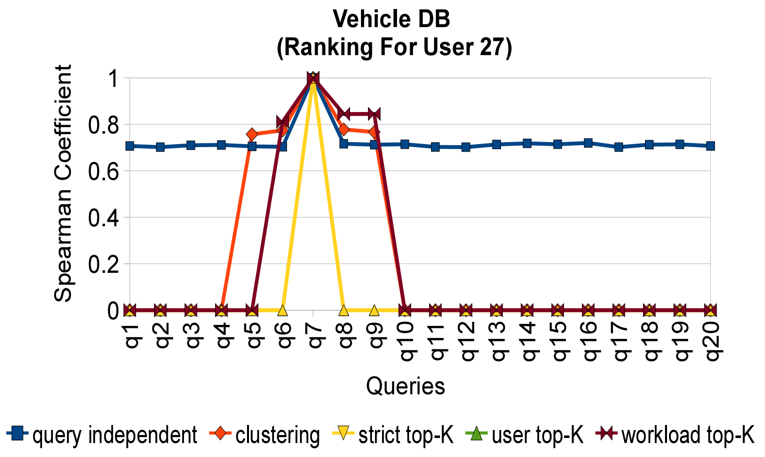


Figure 3.7. Quality Of User Similarity For An Individual User.

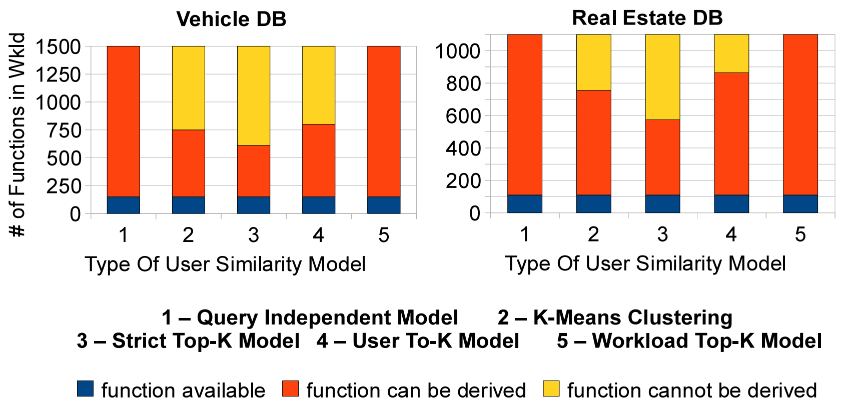


Figure 3.8. Number Of Functions Available For User Similarity Models.

the *vehicle* and *real estate* database respectively. We observe that the effect of this model is enhanced, compared to the individual models, when the sparseness of the workload increases (i.e., the number of ranking functions decreases roughly from 500 to 100 for both databases). However, the overall average value of the *spearman coefficient*, and hence, the average ranking quality decreases with increasing workload sparseness. This matches the intuition that with more ranking functions in the workload, one is likely to find a good ranking function for a pair with better similarity value.

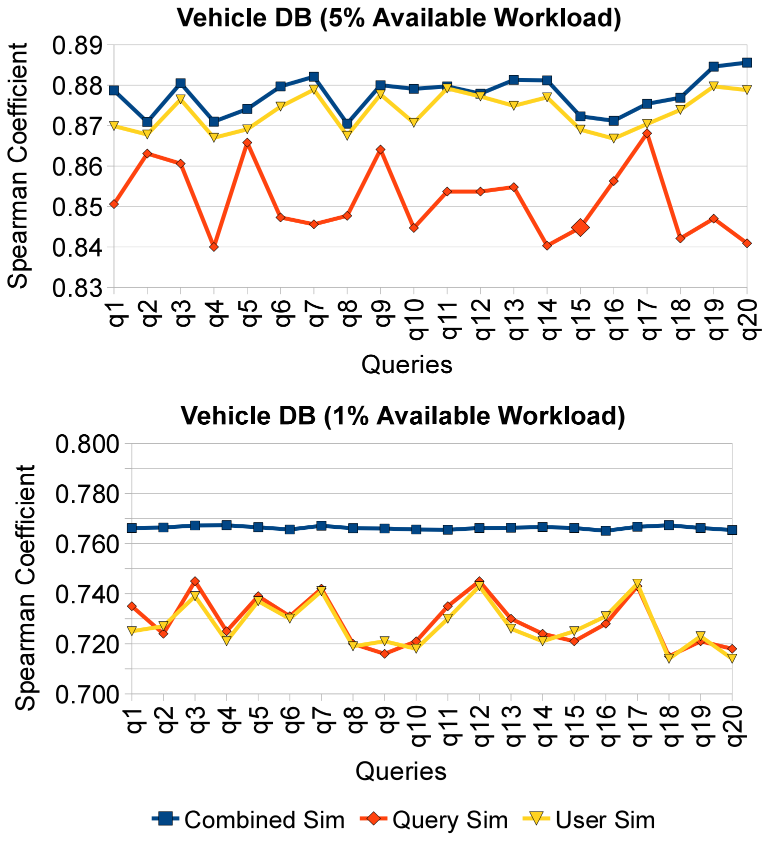


Figure 3.9. Ranking Quality of Holistic Similarity Model: ***Vehicle*** Database.

For instance, in the *vehicle* database, the composite model achieved an average Spearman coefficient (across all users asking all queries) of 0.89 versus the 0.85 achieved by the user similarity model and 0.82 achieved by the query similarity model when 95% of the functions were masked (i.e., out of the 5000 results for the query, the combined similarity model correctly ranked 4450 tuples versus the user and query similarity model that correctly ranked 4250 and 41000 tuples respectively). However, when 99% of the functions were masked, the composite model achieved a 0.77 Spearman coefficient versus the 0.72 and 0.71 achieved by the user and query similarity models respectively (i.e., the combined model correctly ranked 3850 tuples whereas the user and query similarity models correctly ranked 3600 and 3550 tuples respectively). A similar trend is seen for the *real*

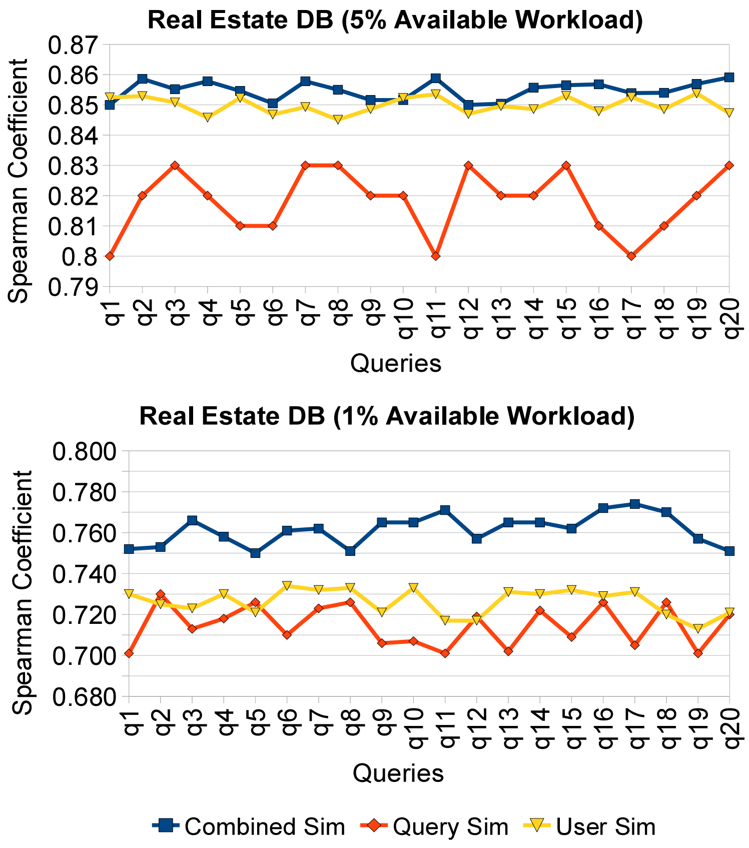


Figure 3.10. Ranking Quality of Holistic Similarity Model: ***Real Estate*** Database.

estate database as well. Furthermore, user similarity shows a better quality than the *query similarity*. Since the averages are presented, one reason for the *user similarity* to be better is that in most cases a similar user is found due to the larger number of users than queries in our experimental setting.

3.6.4 Efficiency Evaluation

The goal of this study was to determine whether our framework can be incorporated into a real-world application. Rather than explore new indexing algorithms and/or data structures, our focus was to use known indexing and hashing techniques to efficiently store our workload, perform search and compute ranking. We believe this can be improved further. We generated a workload comprising of 1 million queries and 1 million users, and randomly masked out ranking functions such that only 0.001% of the workload was filled. We then generated 20 additional queries and selected 50 random users from the workload. We measure the efficiency of our system in terms of the average time, taken across all users, to perform ranking over the results of these queries.

If we use main memory for storing the workload and not use any pre-computation and indexing for retrieval, determining similarities are computational bottlenecks. In order to reduce the time for estimating query similarities, we can pre-compute pairwise similarities between all values of every attribute in the schema. Since most Web database have 10-50 attributes, pre-computing these pairwise similarity values once, and storing them is not a problem. Furthermore, in order to reduce the time to lookup every query in the workload and then evaluate its similarity with the input query, we use a value-based hashing technique [43] to store all the queries in the workload. Likewise, all users are stored using a similar technique where the values corresponding to a user refer to various properties of the user profile.

Figure 3.11 show the performance of our framework using naive (where all steps are done in main memory at query time) and pre-computed/hash-based approaches (where similarities are pre-computed and the values are stored using hashing) for both databases. As is expected the naive approach is an order of magnitude slower than the other approach. For the *vehicle* database, Google Base provides unranked results in approximately 1.5 seconds. Our hash-based pre-computed system takes an average of 2.84 seconds (including Google Base response time) to rank and return the results. Our system adds, on the average, 1.3 seconds to return results using query- and user-dependent ranking. Likewise, for the *real estate* database, our system takes an average of 2.91 seconds (as compared to 2 seconds for Google Base) adding less than a second for the improved ranking.

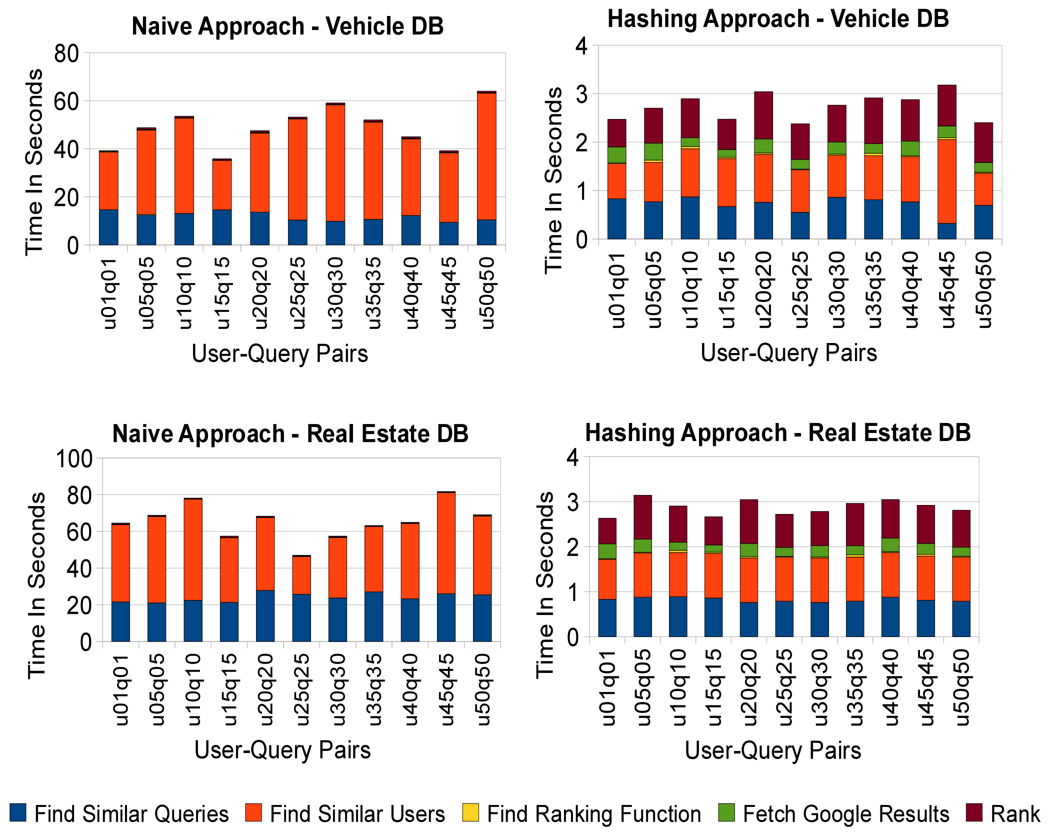


Figure 3.11. Ranking Efficiency Of Combined Similarity Model.

Although our system adds approximately a second over Google to display the results, the user is getting a customized, ranked result as compared to a *one-size-fits-all* ranking. We believe that if, the ranking produced is desirable to the users, the extra overhead added would still be tolerable as compared to displaying unranked results. The fact is that over two distinct databases having different schema, with a workload of 1 Million users and queries each, the results show that this framework can be incorporated into a real-life application. Finally, as we indicated earlier, the performance can be improved further.

3.7 Deriving Individual Ranking Functions For The Workload

Our proposed framework uses a workload of ranking functions derived across several user-query pairs. Since a ranking function symbolizes an user's specific preferences towards individual query results, obtaining such a function (especially for queries returning large number of results) is not a trivial task in the context of Web databases. Furthermore, since obtaining ranking functions from users on the Web is difficult (given the effort and time the user needs to spend in providing his/her preferences) determining the exact set of ranking functions to be derived for establishing the workload is important. Although we discuss the latter problem in the next Chapter, we present a learning technique for the former in this Chapter. Prior to explaining the proposed technique, we briefly discuss the possible alternatives in deriving individual ranking functions.

3.7.1 Alternatives For Obtaining A Ranking Function

Given a user U, a query Q and its results N, the ranking function ($\mathcal{F}_{\mathcal{U},\mathcal{Q}}$) corresponds to the preference associated to each tuple (in N) by U. A straightforward alternative for deriving $\mathcal{F}_{\mathcal{U},\mathcal{Q}}$ would be to ask U to manually specify the preferences as part of Q; a technique adapted in relational databases [37] [38] [39] [40] [41]. However, Web users are typically unaware of the query language, the data model as well as the workings of a ranking mechanism; thus, rendering this solution unsuitable for Web databases.

Since Web applications allow users to select results of their choice by an interaction (clicking, cursor placement, etc.) with the Web page, a feasible alternative for obtaining $\mathcal{F}_{\mathcal{U},\mathcal{Q}}$ would be to analyze U's interaction over N; an approach similar to *relevance feedback* [71] in document retrieval systems. However, mandatorily asking U to iteratively mark a set of tuples as relevant (and non-relevant) i.e., obtaining *explicit* feedback, may be difficult since Web users are typically reluctant to indulge in a lengthy interactions. Although providing various incentives to users (as done by routine surveys conducted by portals such as Yahoo, Amazon, etc.) is possible, an explicit feedback approach appears impractical in the domain of most Web databases.

In contrast, since most users voluntarily choose a few tuples for further investigation (typically by interacting with the database application via operations such as clicking, cursor placement, etc.), applying *implicit* feedback (where these selected tuples can be analyzed without U's knowledge) to obtain $\mathcal{F}_{\mathcal{U},\mathcal{Q}}$ seems a more practical alternative. However, determining the number of tuples to be selected for generating an accurate function is difficult. This problem is further compounded by the fact that most users typically select very few tuples from the displayed results. An alternative to this problem would be to display an appropriately chosen sample, instead of the entire set of tuples, and determine $\mathcal{F}_{\mathcal{U},\mathcal{Q}}$ based on the tuples selected from this sample (and extrapolate it over the entire set of results). Although determining an appropriate sample and validating the corresponding function obtained is difficult, we believe that there is scope to further investigate this problem and derive relevant solutions. In this dissertation, we present a preliminary solution, in the form of a learning model, to obtain the requisite function $\mathcal{F}_{\mathcal{U},\mathcal{Q}}$ using one such *implicit* technique.

3.7.2 A Probabilistic Learning Model

For a query Q given by user U, we have at our disposal the set of query results N generated by the database. Let R ($\subset N$) be the set of tuples selected *implicitly* by U. For an attribute A_i, the relationship between its values in N and R can capture the significance (or weight) associated by U for A_i. As discussed in Section 3.2, our ranking function is a linear combination of attribute-weights and value-weights. We propose a learning technique called the **Probabilistic Data Distribution Difference** method for estimating these weights.

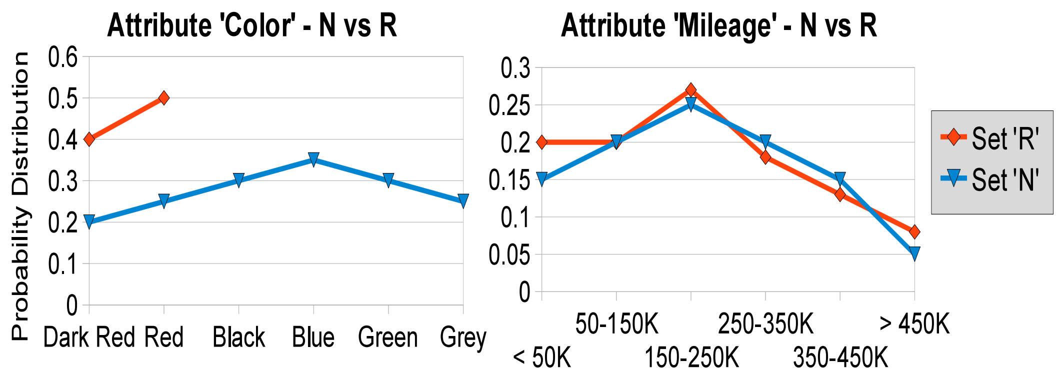


Figure 3.12. Probability Distribution Difference: Retrieved v/s Selected Results.

3.7.2.1 Learning Attribute-Weights

From *Example-2*, we know that U_1 is interested in ‘red’ colored ‘Honda’ vehicles. Correspondingly, let us consider U_1’s preferences towards the “Color” and “Mileage” attributes, and consider the individual probability distributions (shown in Figure 3.12) of their respective values in sets N and R. Since the set R will contain only ‘red’ colored vehicles, there is a significant difference between the distributions (for “Color”) in R and N. In contrast, assuming that U_1 is not interested in any particular mileage, the difference in the distributions for “Mileage” over sets R (which will contain cars with different mileages) and N is small.

Based on this observation, we can hypothesize that – for an attribute A_i, if the difference in the probability distributions between N and R is *large*, it indicates that A_i is important to the user, and hence, will be assigned a *higher* weight, and vice versa. The attribute weights are estimated formally, using the popular *Bhattacharya distance* (D_B) measure [72]. If the probability distributions for the values of attribute A_i in sets N and R are N_p and R_p respectively, the *attribute-weight* (W_{A_i}) of A_i is given as:

$$W_{A_i} = D_B\left(N_p, R_p\right) \ = \ -ln\left(BC(N_p, R_p)\right) \tag{3.9}$$

where *BC* (Bhattacharya coefficient) for categorical and numerical attributes is given in Equations 3.10 and 3.11 as:

$$BC(N_p, R_p) = \sum_{x \in X} \sqrt{N_x, R_x} \tag{3.10}$$

$$BC(N_p, R_p) = \int \sqrt{N_x, R_x}\, dx \tag{3.11}$$

3.7.2.2 Establishing Value-Weights

In order to rank the results of a query (using Equation 3.1), it is necessary that the score associated with every tuple is on a canonical scale. Although the attribute-scores estimated using the *Bhattacharya Distance* are between [0.0,1.0], the values in the tuples may have different types and ranges. Hence, we need to normalize them to obtain the necessary value-weights.

Since we have assumed that numerical attribute values are discretized using an appropriate scheme (Section 3.3.1), we normalize only categorical attributes (e.g., "Make", "Color", etc.) using a *frequency-based* approach. For the query Q by U, let a be the value of a categorical attribute A_i. The *value-weight* (a_{weight}) is given (by Equation 3.12) as the ratio of the frequency of a in R (a_r) with its frequency in N (a_n).

$$a_{weight} = \frac{a_r/|R|}{a_n/|N|} \tag{3.12}$$

3.7.3 Learning Model Evaluation

We test the quality of our proposed learning method (*Probabilistic Data Distribution Difference*) in deriving *attribute-weights* for a user query. In an ideal scenario, using the feedback (i.e., tuples selected) provided by an user over a query's results, a ranking function would be deduced using the learning model. The quality of this function, however, can only be evaluated the next time the same user asks the same query, and would be estimated in terms of the percentage of the top-k results generated (by this function) that match the user's preferences. In an experimental setting, asking a user to select tuples (from a large set) once, and then validate the function by asking the same query again would be difficult.

Consequently, we test the quality of the proposed learning model as follows: From the workload in Section 3.6.2, we have at our disposal ranking functions provided by users over 20 distinct queries on two databases. Consider query Q_1 (with results N_1) for which a user U_1 has provided a ranking function ($\mathcal{F}_{1,1}$). Using this function, we rank N_1 to obtain N_{r1}, and select a set of top-K tuples (from this ranked set) as the set (R_1) chosen by U_1. Using our learning model, we derive a ranking function (${\mathcal{F}'}_{1,1}$) comparing N_1 and R_1. We then use ${\mathcal{F}'}_{1,1}$ to rank Q_1's results and obtain N'_{r1}. The quality of the learning model is then estimated as the *Spearman rank correlation coefficient* (Equation 3.8) between N_{r1} and N'_{r1}. Higher the value of this coefficient, better is the quality of the ranking function (and the model), and vice-versa.

In our experiments, we chose an initial value of $K = 25$ i.e., we choose the *top-25* tuples generated by the user's original ranking function as the set R since a Web user, in general, selects a very small percentage of the top-K tuples shown to him/her. The model was also tested with R varying from 10 to 50 tuples. Below, we present the evaluation when the set R contains 25 and 10 tuples respectively. We validate the ranking quality of our proposed learning model to the quality achieved by using two established and widely-used learning models – *Linear Regression* and *Naive Bayesian classifier*. Figures 3.13 and 3.14 compares the average ranking quality achieved by the models, for R = 25 and R = 10 respectively, in deriving ranking functions for each individual query for all the users in both databases. As the figure illustrates, for the *vehicle* database, our proposed model attains an average value of 0.88 for the Spearman coefficient, as compared to the average

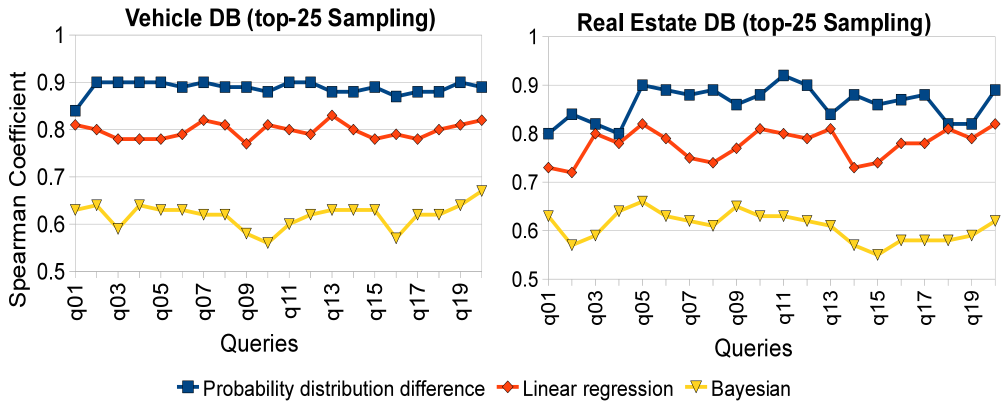


Figure 3.13. Ranking Quality Evaluation Of Learning Models: Top-25 Selection.

values obtained by Linear Regression (0.81) and Naive Bayesian classifier (0.66). These values indicate that the proposed learning model derives a ranking function that yields a ranking order that is very close to the order desired by the user, than the ones generated by the other two learning models. Furthermore, it can be observed that the learning models perform significantly better when the size of R increases (as seen when an average value of 0.82 is obtained when $R = 10$ as compared to 0.88 produced when $R = 25$); thus, indicating that greater the number of interactions of the user with the query results, better will be the quality of the eventual ranking function inferred from these interactions. A similar trend can be observed for the *real estate* database as well.

In order to prove the generic effectiveness of our model, the R is now chosen using different sampling techniques (instead of top-25). It is natural that based on users' preferences, higher ranked tuples (from N_{r1}) should have a higher probability of being sampled than the lower ranked ones. Hence, the sampling technique we choose selects the required $R = 25$ tuples using following Power-Law [73] distributions: *Zipf*, *Zeta*, and *Pareto*. Figure 3.15 compares the quality of our learning model (shown by vertical bar in the graph) with the other models across both databases using different sampling schemes. The results clearly indicates that our proposed method performs significantly better than the popular

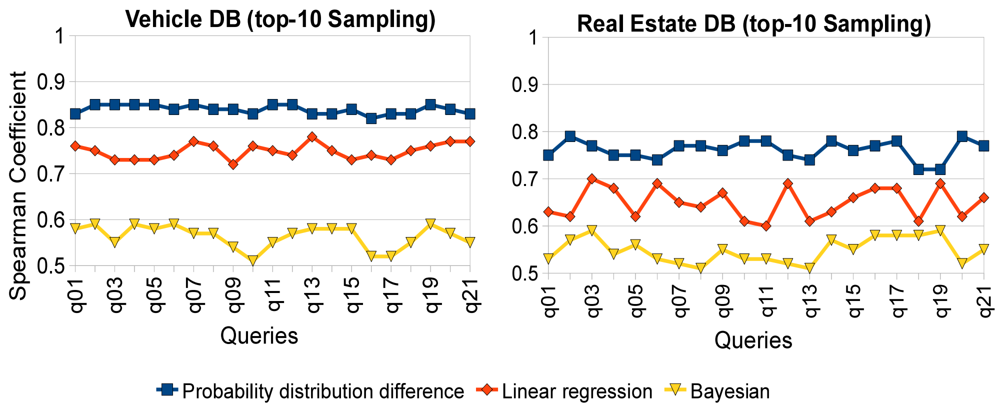


Figure 3.14. Ranking Quality Evaluation Of Learning Models: Top-10 Selection.

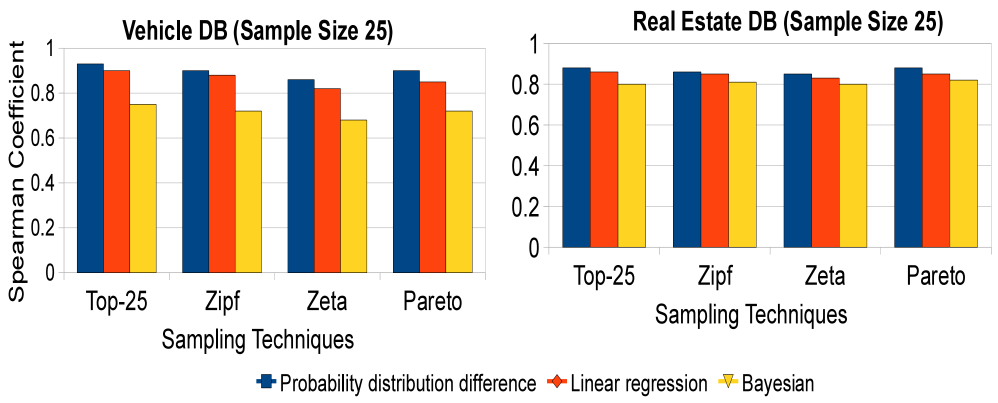


Figure 3.15. Ranking Quality Evaluation Using Power-Law Sampling Schemes.

learning models. These results also validate our claim of *learning* using *feedback* as a suitable alternative to obtaining ranking functions for generating workloads.

3.8 Conclusion

In this Chapter, we proposed a *user*- and *query*-dependent solution for ranking query results for Web databases. We formally defined the similarity models (*user*, *query* and *holistic*) and presented experimental results over two Web databases to corroborate our analysis. We demonstrated the practicality of our implementation for real-life databases. Further, we presented a learning method for inferring individual ranking functions.

In the context of Web databases, an important challenge is the design and maintenance of an appropriate workload that satisfies properties of similarity-based ranking. We address this challenge and present a solution for the same in the next Chapter.

CHAPTER 4

ESTABLISHING A WORKLOAD FOR SIMILARITY RANKING

An important component of the similarity-based ranking framework [49] [50] (discussed in Chapter 3) is a workload of ranking functions, where each function represents an individual user's preferences towards the results of a specific query. At the time of answering a query for which no prior ranking function exists, the similarity model can ensure a good quality of ranking only if a function for a very similar user-query pair exists in this workload.

Accordingly, in this Chapter, we address the problem of determining an appropriate set of user-query pairs to form a workload of ranking functions for supporting user- and query-dependent ranking in Web databases. We propose a novel metric, termed *workload goodness*, that quantifies the notion of a "good" workload into an absolute value. The process of finding such a workload of optimal goodness is a combinatorially explosive problem; therefore, we propose a heuristic solution, and advance three approaches for determining a workload with acceptable goodness, in a static as well as a dynamic environment. We discuss the effectiveness of our proposal analytically as well as experimentally over two Web databases.

4.1 Introduction

In order to support a holistic scheme for *user*- and *query*-dependent ranking in Web databases, we proposed a novel similarity-based framework in Chapter 3. This framework selects a suitable ranking function from a *workload* comprising of a number of such functions collected across several user-query pairs. Each ranking function in the work-

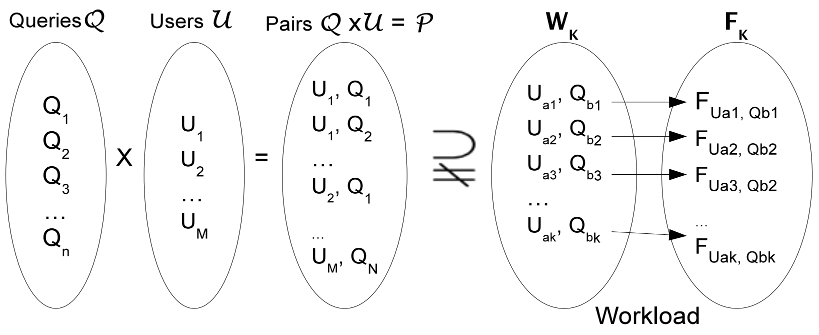

Queries $\mathcal{Q}$
Users $\mathcal{U}$
Pairs $\mathcal{Q}$ x $\mathcal{U}$ = $\mathcal{P}$
$\mathbf{W}_K$
$\mathbf{F}_K$
Q_1 Q_2 Q_3 … Q_n
X
U_1 U_2 … U_M
=
U_1, Q_1 U_1, Q_2 … U_2, Q_1 … U_M, Q_N
$\supsetneq$
U_{a1}, Q_{b1} U_{a2}, Q_{b2} U_{a3}, Q_{b3} … U_{ak}, Q_{bk}
$F_{Ua1, Qb1}$ $F_{Ua2, Qb2}$ $F_{Ua3, Qb2}$ … $F_{Uak, Qbk}$
Workload

Figure 4.1. Sample Queries $\mathcal{Q}$, Users $\mathcal{U}$, Pairs $\mathcal{P}$ & Workload.

load represents the preferences of a distinct user over the results of a specific query, and is selected on the assumptions that: i) for the results of a given query, similar users display comparable ranking preferences, and ii) a user displays analogous ranking preferences over results of similar queries. It is important to note that the concept of a *workload* used in the context of similarity-based ranking is significantly different from the one used in traditional databases. In the former, it represents a collection of user-query pairs along with individual ranking functions associated with each of these pairs; in contrast, it pertains to a simple log of queries for the latter's context.

The notion of such a workload is further illustrated by Figure 4.1. Let $\mathcal{Q}$ = $\{Q_1$, Q_2, ..., $Q_N\}$ and $\mathcal{U}$ = $\{U_1$, U_2, ..., $U_M\}$, respectively, represent a large set of queries and users over a Web database. The set $\mathcal{P}$ = $\{(U_1, Q_1)$, (U_2, Q_1), ..., (U_1, Q_1), ..., $(U_M, Q_N)\}$ (of size $M * N$) is the Cartesian product of the sets $\mathcal{Q}$ and $\mathcal{U}$. Accordingly, the *workload* is defined as a onto mapping between two sets – $\mathbf{W}_K$ and $\mathbf{F}_K$, of size[1] K, such that:

- $\mathbf{W}_K$ = $\{(U_{a_1}, Q_{b_1})$, (U_{a_2}, Q_{b_2}), ..., $(U_{a_K}, Q_{b_K})\}$ where $\mathbf{W}_K \subsetneq \mathcal{P}$, and
- $\mathbf{F}_K$ = $\{\mathcal{F}_{U_{a_1}, Q_{b_1}}$, $\mathcal{F}_{U_{a_2}, Q_{b_2}}$, ..., $\mathcal{F}_{U_{a_K}, Q_{b_K}}\}$ where each $\mathcal{F}_{U_{a_i}, Q_{b_i}}$ represents the ranking function for the user-query pair (U_{a_i}, Q_{b_i}).

[1]The size of K can vary between 0 and $M * N$

At the time of answering a query Q_j ($\in \mathcal{Q}$) asked by a user U_i ($\in \mathcal{U}$), if (U_i, Q_j) $\notin \mathbf{W}_K$, the similarity model employs Algorithm 1 (detailed in Section 3.5 of Chapter 3) and orders all users in $\mathcal{U}$ based on their similarity with U_i. Likewise, based on their similarity with Q_j, it orders all queries in $\mathcal{Q}$. Each user-query pair in $\mathcal{P}$, then, receives a rank with respect to the pair (U_i, Q_j), that is computed using the individual ranking positions of the query and user (within the pair) in the respective ordered lists of queries and users.

Given that every pair $\in \mathcal{P}$ receives a rank with respect to every other pair ($\in \mathcal{P}$), these ranks can be represented in the form of a *rank matrix* $\mathbf{R}$ (of size $M * N$ X $M * N$) where the rows as well as columns represent each distinct pair from $\mathcal{P}$. Table 4.1 shows an example of such a matrix formed over the user-query pairs in $\mathcal{P}$ from Figure 4.1. Each cell (e.g., $\mathbf{R}$[2][1]) of this matrix represents the rank of the pair representing the **column** (in this case, (U_1, Q_1)) with respect to the pair representing the **row** (i.e., (U_1, Q_2)) and is represented as $Rank[(U_1, Q_2), (U_1, Q_1)]$. Formally, given two pairs, (U_x, Q_y) and (U_i, Q_j), the rank $Rank[(U_i, Q_j), (U_x, Q_y)]$ is computed as shown in Equation 4.1.

$$Rank[(U_i, Q_j), (U_x, Q_y)] = \mathbf{R}_{U_i}^{U_x} + \mathbf{R}_{Q_j}^{Q_y} \tag{4.1}$$

where, $\mathbf{R}_{U_i}^{U_x}$ is the rank of U_x with respect to U_i in the ordered list of queries from $\mathcal{U}$. Likewise, $\mathbf{R}_{Q_j}^{Q_y}$ is the rank of Q_y with respect to Q_j in the ordered list of queries from $\mathcal{Q}$.

Thus, for a given pair (U_i, Q_j) the similarity model selects, amongst the cells in the row corresponding to (U_i, Q_j), the cell (e.g., (U_x, Q_y)) with the highest rank (with respect to (U_i, Q_j)) such that $(U_x, Q_y) \in \mathbf{W}_K$.

Based on the definition of the similarity model, if U_x and Q_y have a very high similarity with respect to U_i and Q_j, the corresponding pair (U_x, Q_y) will receive a very high rank, and therefore, a *good* quality of ranking will be obtained after applying $\mathcal{F}_{U_x,Q_y}$ to the results of U_i. In contrast, if the similarity between U_x and U_i (and/or between Q_y and

Table 4.1. Sample Rank Matrix For Set $\mathcal{P}$

	(U_1, Q_1)	(U_1, Q_2)	...	(U_2, Q_1)	...	(U_n, Q_m)
(U_1, Q_1)	0	7	...	21	...	966
(U_1, Q_2)	81	0	...	467	...	45
...	...	...	...	...	...	...
(U_2, Q_1)	515	171	...	0	...	710
...	...	...	...	...	...	...
(U_n, Q_m)	981	771	...	481	...	0

Q_j) is very low, the subsequent quality of ranking will be poor. Thus, we can informally hypothesize that the workload is a ***good*** workload if, for *any* query Q_j ($\in \mathcal{Q}$) asked by *any* user U_i ($\in \mathcal{U}$), there exists at least one pair (e.g., (U_x, Q_y)) $\in \mathbf{W}_K$ such that the similarity between the queries U_x and U_i, and the similarity between users Q_y and Q_j, is very high.

The task of establishing such a *good* workload involves several challenges. For instance, determining the appropriate $\mathbf{W}_K$ i.e., the set of K user-query pairs from $\mathcal{P}$ is an important problem. Likewise, we need to obtain K suitable ranking functions, one for each distinct pair in this set $\mathbf{W}_K$, to form the set $\mathbf{F}_K$. Currently, the similarity model acquires each ranking function via a learning model [49] (discussed in Chapter 3) that analyzes the user's interactions with the query results. Given the time and effort a user needs to spend in selecting results of his/her choice, obtaining a large number of functions for various user-query pairs is difficult. Hence, determining a pragmatic value for K i.e., the number of functions that can be actually obtained, is vital. In addition, each ranking function is derived based on the relationship between a sample of selected tuples and the entire set of query results. Given that the similarity between users is computed based on these collected ranking functions, evaluation of these functions becomes crucial. Furthermore, the above discussion assumes a static collection of queries and users over the Web database. However, in a real setting, the number of users and queries change over time; hence, incorporating these changes while establishing the workload is important.

In this work, we concentrate on the problem of forming $\mathbf{W}_K$ in a *static* as well as *dynamic* environment i.e., given $\mathcal{P}$, our task is to determine an appropriate set of K user-query pairs such that whenever the corresponding functions are obtained for these pairs, the result will be a *good* workload. We term this as the ***Workload Filling Problem***. The choice of an appropriate K (i.e., the size of the sets $\mathbf{W}_K$, and hence, $\mathbf{F}_K$) may be determined based on the database and other pragmatic issues such as incentive mechanisms used for that purpose. Further, we assume that for a given $\mathbf{W}_K$, the resulting $\mathbf{F}_K$ can be subsequently established (using the learning model given in Section 3.7.2); thereby yielding the necessary workload[2].

In order to formalize the *Workload Filling* problem, we need to translate the above hypothesis, of determining a *good* set of K pairs, into a standard model. Toward that, we propose a novel metric, termed the ***workload goodness*** metric, that assigns a value of goodness to a given $\mathbf{W}_K$. As elaborated later in Section 4.2, lower the estimated value of this metric (represented as $\mathcal{G}(\mathcal{Q}, \mathcal{U}, \mathbf{W}_K)$), better is the quality of the corresponding workload and vice-versa. Thus, our goal is to determine $\mathbf{W}_K$ such that for this set of pairs, the value of $\mathcal{G}(\mathcal{Q}, \mathcal{U}, \mathbf{W}_K)$ is as low as possible. However, the process of determining a workload that yields an optimal goodness value leads to a combinatorial explosion (elaborated in Section 4.2.1) when the sizes of $\mathcal{Q}$ and $\mathcal{U}$ are very large[3]. Hence, we propose a heuristic solution and advance three distinct approaches:

1. Independent Rank-based Selection,
2. Independent High Rank-based Selection, and
3. Cumulative Selection.

[2]Since the focus is on establishing $\mathbf{W}_K$, the terms *workload* and $\mathbf{W}_K$ are used interchangeably.

[3]Typically, the number of users and queries on most real Web databases are extremely large, whereas the value of K is typically much smaller than the number of users and/or queries.

Further, given that the queries and users on a Web database change frequently, the proposed approaches can be adapted such that the set of K pairs can be determined in a static as well as dynamic (or incremental) fashion.

Contributions: The contributions of this work are -

- Formulation of a *workload* model, as a set of user-query pairs (coupled with their respective ranking functions), to support user- and query-dependent ranking using the similarity model.
- A novel metric of *Workload Goodness* that translates the hypothesis behind a "good" workload into an absolute value.
- Given the intractability in determining a $\mathbf{W}_K$ of optimal goodness for a given K, a heuristic solution to the *Workload Filling Problem* is proposed. Three approaches based on different heuristics for determining $\mathbf{W}_K$ in a *static* and *dynamic* environment are presented.
- An elaborate set of experimental results that establishes the quality and efficiency of our proposal.

Roadmap: Section 4.2 formally defines the *Workload Filling* problem, and introduces the metric of *workload goodness*. Section 4.3 elaborates on the proposed heuristic solution and the details of the algorithms. The results of experimental evaluation in terms of quality and efficiency are discussed in Section 4.4 and we conclude in Section 4.5.

4.2 Problem Statement

We begin by defining the *workload goodness* metric, and then provide a formal definition for the *Workload Filling Problem*. Further, we explain the difficulty in determining the set of user-query pairs that yield an optimal goodness, and motivate the need for heuristic solutions to the problem. We also provide a brief discussion on the applicability of our proposal for establishing a workload in static as well as dynamic environments.

4.2.1 The Workload Goodness Metric

Based on the explanation of the similarity model in Chapter 3, for a user U_i asking a query Q_j, the user-query pair (e.g., (U_x, Q_y)) having the highest rank amongst all pairs in $\mathbf{W}_K$ is chosen. If the individual ranks of U_x (w.r.t. U_i) and Q_y (w.r.t Q_j) within the ordered lists of queries and users in $\mathcal{Q}$ and $\mathcal{U}$ are very high, the resulting $Rank[(U_i, Q_j), (U_x, Q_y)]$ will be high; hence, applying the corresponding $\mathcal{F}_{U_x,Q_y}$ to U_i's results will yield a **good** quality of ranking order. In contrast, if the ranks of U_x and/or Q_y are very low, the subsequent quality of ranking will be **poor**.

Accordingly, we can assign a measure of goodness, in terms of the rank of the selected user-query pair ($(U_x, Q_y) \in \mathbf{W}_K$), to the overall quality of ranking achieved. Higher the rank of the selected pair, better is the goodness of the ranking order, and vice versa. Thus, over the entire set $\mathcal{P}$, we can associate a goodness to the $\mathbf{W}_K$ based on the contribution of the pairs within $\mathbf{W}_K$ toward the subsequent quality of ranking. We term this as the ***workload goodness*** metric, represented as $\mathcal{G}(\mathcal{Q},\mathcal{U}, \mathbf{W}_K)$. Without loss of generality and for the sake of simplicity, if we represent $\mathbf{W}_K$ as – $\{(U_{x_1}, Q_{y_1}), (U_{x_2}, Q_{y_2}), ..., (U_{x_k}, Q_{y_k})\}$, then $\mathcal{G}(\mathcal{Q},\mathcal{U}, \mathbf{W}_K)$ is computed as shown in Equation 4.2.

$$\mathcal{G}(\mathcal{Q},\mathcal{U},\mathbf{W}_K) = \frac{\sum_{a=1}^{|\mathcal{U}|} \sum_{b=1}^{|\mathcal{Q}|} Rank[(U_a, Q_b), \mathbf{W}_K]}{|\mathcal{U}| * |\mathcal{Q}|} \tag{4.2}$$

where, using Equation 4.1,

$$Rank[(U_a, Q_b), \mathbf{W}_K] = MIN(Rank[(U_a, Q_b), (U_{x_p}, Q_{y_p})] \mid (U_{x_p}, Q_{y_p}) \in \mathbf{W}_K) \tag{4.3}$$

Intuitively, the *workload goodness* metric assigns a *goodness* value to each user-query pair (e.g., (U_a, Q_b) in $\mathcal{P}$), that in turn, is the *rank* of user-query pair (e.g., (U_{x_i}, Q_{y_i}) $\in \mathbf{W}_K$) that attains the highest rank, amongst all pairs in $\mathbf{W}_K$, with respect to (U_a, Q_b). Lower the value obtained for $\mathcal{G}(\mathcal{Q},\mathcal{U}, \mathbf{W}_K)$, better is the goodness of the resulting workload, and vice-versa.

Bounds For $\mathcal{G}(\mathcal{Q},\mathcal{U}, \mathbf{W}_K)$: Consider any arbitrary user-query pair (say U_i, Q_j) in $\mathcal{P}$. The **optimal** value of the rank of this pair i.e., $Rank[(U_i, Q_j), \mathbf{W}_K]$ is derived from Equation 4.3, and is computed as:

$$\begin{aligned} Rank[(U_i, Q_j), \mathbf{W}_K] &= 0 \;\; if\, (U_i, Q_j) \in \mathbf{W}_K \\ &= 1 \;\; otherwise^4 \end{aligned} \tag{4.4}$$

Thus, over the entire set $\mathcal{P}$ (of size $M * N$), if each pair (e.g., $U_i, Q_j \notin \mathbf{W}_K$) receives an optimal value (from Equation 4.4) of 1 for $Rank[(U_i, Q_j), \mathbf{W}]$, the resulting optimal value (from Equation 4.2) for $\mathcal{G}(\mathcal{Q},\mathcal{U}, \mathbf{W}_K)$ will be:

$$\mathcal{G}(\mathcal{Q},\mathcal{U}, \mathbf{W}_K) = \frac{M * N - |\mathbf{W}_K|}{M * N} \tag{4.5}$$

Consequently, Table 4.2 shows the resulting goodness values for different sizes of $\mathbf{W}_K$ computed from Equation 4.5. Based on this, it can be conclusively stated that the bounds i.e., the best- and worst-case values for $\mathcal{G}(\mathcal{Q},\mathcal{U}, \mathbf{W}_K)$ are 0 and (approximately) $M * N$ respectively. It can be further observed that the (optimal value of) goodness decreases in a strictly monotonic manner. This further validates the hypothesis that the choice of establishing an appropriate K is a pragmatic problem, instead of a theoretic problem.

[4]The highest rank of 0 is assigned to the pair itself, the next highest possible rank, computed by Equation 4.1, of a user-query pair with respect to a given pair is 1.

Table 4.2. Impact Of $|\mathbf{W}_K|$ On Optimal Goodness

$\lvert\mathbf{W}_K\rvert$	Optimal $\mathcal{G}(\mathcal{Q},\mathcal{U}, \mathbf{W}_K)$
1	(M * N-1)/M * N
K	(M * N - K)/M * N
K+1	(M * N- K-1)/M * N
M * N	0

4.2.2 The Workload Filling Problem

Consider $\mathcal{Q}$ =$\{Q_1, Q_2, ..., Q_N\}$ and $\mathcal{U}$ = $\{U_1, U_2, ..., U_M\}$ respectively representing a large set of queries and users over a Web database $\mathbf{D}$, and let K be the size of $\mathbf{W}_K$ to be established for assisting the similarity-based ranking framework.

Definition Given $\mathcal{Q}$, $\mathcal{U}$, $\mathcal{P}$, and K, the *Workload Filling Problem* can be defined as the task of determining, from the total of $M * N$ user-query pairs, a set of K pairs to form $\mathbf{W}_K$ such that given any other $\mathbf{W}'_K$ of size K, $\mathcal{G}(\mathcal{Q},\mathcal{U}, \mathbf{W}_K) \leq \mathcal{G}(\mathcal{Q},\mathcal{U}, \mathbf{W}'_K)$.

The above definition is based on the simplest case of the filling problem i.e., it considers a pre-defined size for sets $\mathcal{Q}$ and $\mathcal{U}$. Further, it presumes that once established, the similarities between individual pairs of queries and users do not change. However, in the case of real Web databases, users and queries keep changing. Further, as new queries are added (to set $\mathcal{Q}$), the orderings of queries with respect to each other are likely to change. Similarly, as ranking functions are subsequently obtained for the pairs corresponding to $\mathbf{W}_K$, similarities between users and hence, orderings of users are likely to change. Thus, a solution to the *Workload Filling* problem should be able to determine the requisite set of user-query pairs in a –

1. Static environment i.e., when users, queries and similarities do not change, and a,
2. Dynamic environment i.e., when users, queries and/or similarities change over time.

Brute-force solution: In order to determine a $\mathbf{W}_K$ of size K that yields the lowest possible value for $\mathcal{G}(\mathcal{Q},\mathcal{U}, \mathbf{W}_K)$, we need to generate a total of $\binom{M*N}{K}$ sets of user-query pairs. The best $\mathbf{W}_K$ will then be the one that yields the optimal value for $\mathcal{G}(\mathcal{Q},\mathcal{U}, \mathbf{W}_K)$ amongst all these configurations. However, in the case of most Web databases, the total number of queries and users are typically very large in order of tens of thousands; hence, the computation for finding the optimal $\mathbf{W}_K$ will lead to a combinatorial explosion (since it is exponential on K).

Thus, finding an optimal set $\mathbf{W}_K$ for real Web databases is infeasible in practice.

Heuristic solution: In this paper, we propose a heuristic solution to the *Workload Filling Problem*, and present three distinct approaches using different heuristics. Each of the three solutions are capable of determining the set of K pairs in a static as well as dynamic environment. As we shall observe in Section 4.3, the algorithmic model that determines the requisite pairs is identical for both, the static as well as dynamic environment. Prior to elaborating the solution, we provide a brief discussion on generating the initial set of queries and users, and establishing the subsequent query and user similarities from which an initial set of K pairs can be formed in a static environment.

4.2.3 Generating $\mathcal{Q}, \mathcal{U}$, and Subsequent Similarities

The schema used for querying Web databases typically represents a single table with a small set of attributes. Barring a few (e.g., Location), the domain of each attribute is typically small in size and does not change frequently. In addition, it is intuitive that the queries asked by most Web users tend to be formed on a small subset of popular domain attributes (e.g., price, mileage, model, etc. for a Vehicle database). Correspondingly, the set $\mathcal{Q}$ can be generated such that it contains queries with varying number of attributes that involve different combinations of these popular attributes and their corresponding values. Further, using Equation 3.2, initial similarities between each pair of the generated queries can be computed. From these similarities, we can determine, for each query, an ordering of the remaining queries.

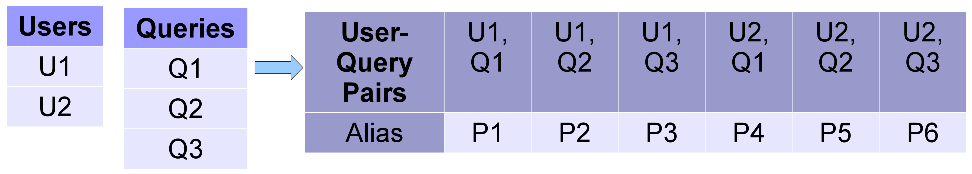
In contrast to query similarities, establishing pairwise similarities between users is not straightforward. From Equation 3.7, pairwise similarities are calculated based on the ranking functions associated with the corresponding users over the same set of queries. Similar to the cold-start problem encountered in recommender systems [74], if there exist no common queries between a given pair of users, computing similarity between them is not possible. In the context of Web databases, this is a common problem since not all users will have asked the same set of queries; thus, obtaining for any given user, an ordering of all users in $\mathcal{U}$ may not be possible. We overcome this problem by assigning an equal and a very small value of similarity to those users whose similarities with a given user cannot be computed. Thus, these users will be assigned an equal but a very low rank in the ordered list of users with respect to the given user.

4.3 Heuristic Solution To The Workload Filling Problem

In order to illustrate our proposed solution, we use the following scenario as a running example for this Section.

Example-3: *Consider a sample set $\mathcal{Q}$ = $\{Q_1, Q_2, Q_3\}$ and a sample set $\mathcal{U}$ = $\{U_1, U_2\}$. The resulting set, $\mathcal{P}$ (i.e., $\mathcal{Q}$ X $\mathcal{U}$), of user-query pairs[5] is shown in Figure 4.2. Table 4.3 shows for each user-query pair, the ranks of the remaining pairs computed using Equation 4.1 (from Section 4.1). Our goal is to determine a set of K=2 user-query pairs to form the set $\mathbf{W}_K$. Using a brute-force solution, Equation 4.2 will generate two configurations – $\mathbf{W}_K^1$ = $\{P_1, P_5\}$, and $\mathbf{W}_K^2$ = $\{P_1, P_6\}$ that both yield an optimal value of* 0.83 *for the* Workload Goodness *metric. The objective of the heuristic solution thus, is to obtain a configuration of pairs that returns a goodness value close to this optimal value.*

[5]For the sake of simplicity, we assign an alias to each user-query pair (e.g., P_2 for the pair U_1, Q_2).



Users
U1
U2

Queries
Q1
Q2
Q3

User-Query Pairs	U1, Q1	U1, Q2	U1, Q3	U2, Q1	U2, Q2	U2, Q3
Alias	P1	P2	P3	P4	P5	P6

Figure 4.2. Example-3: Sample Workload.

Table 4.3. Example-3: Ranked Ordering Of User Query Pairs

Ranks	**0**	**1**	**2**	**3**	**4**	**5**
P_1	P_1	P_4	P_2	P_3	P_6	P_5
P_2	P_2	P_1	P_3	P_4	P_5	P_6
P_3	P_3	P_1	P_2	P_4	P_6	P_5
P_4	P_4	P_3	P_1	P_2	P_5	P_6
P_5	P_5	P_6	P_1	P_2	P_3	P_4
P_6	P_6	P_5	P_4	P_3	P_2	P_1

4.3.1 Independent Rank-Based Selection

This approach avails of the fact that every user-query pair has a rank with respect to every other pair in the set of $\mathcal{P}$ pairs. Further, it follows the underlying assumption of the similarity model i.e., if a pair (say P_1) is very similar, and thus, has a very high rank with respect to a given user-query pair (say P_2) applying the ranking function collected for P_1, to the query in P_2, will yield a good quality of ranking.

Correspondingly, this approach uses the notion of an *average* rank to determine the set of K pairs for representing $\mathbf{W}_K$. Given that a pair (e.g., $P_i \in \mathcal{P}$) has a rank with respect to every other pair (as shown in Equation 4.1), the *average* rank of pair P_i is computed as shown in Equation 4.6.

$$\mathbf{R}_{avg}(P_i) = \frac{\sum_{r=1}^{M*N} Rank[P_r, P_i]}{M * N} \tag{4.6}$$

Thus, the *average* rank for every user-query pair in $\mathcal{P}$ will be computed, and the top-K highest ranked pairs will be selected to represent the set $\mathbf{W}_K$. Since this method

Table 4.4. Example-3: Independent Rank-based Selection Results

User-query Pairs	P_1	P_2	P_3	P_4	P_5	P_6
Average Rank-based Score	**1.83**	2.33	**2.16**	2.33	3.16	3.16

Table 4.5. Example-3: Individual & Overall Goodness of $\mathbf{W} = \{P_1, P_3\}$

Configurations	P_1	P_2	P_3	P_4	P_5	P_6	Overall
Resulting *Goodness*	0	1	0	1	2	3	**1.16**

chooses the K pairs in an independent manner, this is termed as the *Independent Rank-based Selection* approach.

Considering the ranks for the pairs, shown in Table 4.3, for the scenario in *Example-3*, the *average* ranks for each pair is computed using Equation 4.6 and is shown in Table 4.4. The pairs P_1 (average rank = 1.83) and P_3 (average rank = 2.16) attain the highest average ranks. Selecting these pairs to represent $\mathbf{W}_K = \{P_1, P_3\}$, the overall goodness can be estimated across each individual pair $\{P_1$, P_2, ..., $P_6\}$ (as the highest rank of a pair in $\mathbf{W}_K$ with respect to a given pair) and is shown in Table 4.5. The overall goodness, derived from Equation 4.2, will be 1.16 (computed as (0+1+0+1+2+3)/6). In contrast, choosing an arbitrary set of pairs (say, P_2 and P_4) will generate a goodness of 1.33 which is farther from the optimal value of 0.83. Thus, this approach yields an acceptable solution to the *Workload Filling* problem.

Time Complexity: Consider a *static* environment wherein the users, queries and similarities are established once (at the start of the *Workload Filling* process). We then have at our disposal, via the similarity model, an ordering of users (with respect to every user in $\mathcal{U}$), queries (with respect to every query in $\mathcal{Q}$) and user-query pairs (with respect to every pair in $\mathcal{P}$). Even in a naive setting, given that the rank of pair with respect to every other pair is known, the *average* rank of a single pair can be pre-computed at the time of ordering pairs with respect to each other[6]. The next step is to select the top-K pairs with the highest *average* ranks. Using an implementation of the *median-of-medians* algorithm [75], this step can be performed in linear time (i.e., O($|\mathcal{P}|$)). Thus, the worst-case time complexity to determine $\mathbf{W}_K$ will be O($|\mathcal{P}|$).

In a *dynamic* environment, users, queries or similarities may undergo changes. Whenever a new user and/or query is added to $\mathcal{U}$ and $\mathcal{Q}$ respectively, the similarity model will perform the necessary re-computations and modify the necessary entries in the matrix $\mathbf{R}$. Based on these changes, the *average* ranks of individual user-query pairs are likely to change; hence, whenever, the similarity model perform re-ordering of users and/or queries, re-computing the *average* ranks and obtaining a new workload of user-query pairs becomes mandatory. As is the case in the *static* environment, determining the K pairs with the highest average ranks incurs the same cost[7] in the *dynamic* environment (i.e., O($|\mathcal{P}_{new}|$)).

4.3.2 Independent High Rank-Based Selection

Although the *Independent Rank-based Selection* approach is simplistic in nature, due to the notion of *average* taken over the entire set $\mathcal{P}$, those pairs which receive good ranks with respect to certain pairs but poor ranks with remaining pairs may get omitted due to their overall *average* ranks being very low. For instance, P_5 from *Example-3* does not get selected since it's *average* rank is low. However, based on the brute-force approach, it is evident that P_5 coupled with P_1 produces a workload of optimal goodness. Hence, instead of imposing the *average* rank estimated over the entire set $\mathcal{P}$ as the sole heuristic,

[6]This can be achieved by maintaining the average across every column in $\mathbf{R}$ to represent the *average* rank for the pair corresponding to the respective column

[7]Where $\mathcal{P}_{new}$ is the new set obtained following the cartesian product of the changed sets $\mathcal{Q}$ and/or $\mathcal{U}$. Therefore, there is an additional cost of re-ordering the queries and/or users apart from adjusting the ranks of the pairs with respect to each other; however, once this is done, the cost of of determining $\mathbf{W}_K$ remains the same.

we propose a variant heuristic that determines the *average* rank of a pair over a **subset** of $\mathcal{P}$. The corresponding *average* rank computed for a pair, over the selected subset of pairs, is termed as the *high average* rank of the pair.

The intuition behind this approach is that if a given pair (e.g., P_1) occurs within a certain threshold in the ordered list of certain pairs (i.e., it obtains a rank above a specified threshold within the ordered list of each of these pairs), then a $\mathbf{W}_K$ containing P_1 is likely to generate a desirable goodness across these pairs. For instance, considering the ordered lists of pairs in Table 4.3 for *Example-3*, it can be observed that P_1 occurs in the ordered lists of most pairs within a threshold of size 3 i.e., P_1 obtains a rank within the top-3 ranked pairs for most pairs (barring P_6 for which it receives a rank of '5'). Therefore, selecting P_1 as one of the pairs for $\mathbf{W}_K$ will yield a desirable goodness.

Correspondingly, this approach determines the set of K pairs whose *average* rank within a specified threshold, termed as the *high average rank*, is the highest amongst all pairs in $\mathcal{P}$. Formally, given the set of $M * N$ pairs and T represents the threshold, the *high average* rank of a pair P_i is computed as shown in Equation 4.7.

$$\mathbf{R}_{high}(P_i) = \frac{\sum_{r=1}^{M*N} Rank[P_r, P_i]}{M * N} \tag{4.7}$$

where, given T as the threshold, we have

$$Rank[P_r, P_i] = \begin{cases} T + 1 & \text{if } Rank[P_r, P_i] > T, \\ Rank[P_r, P_i] & \text{otherwise.} \end{cases} \tag{4.8}$$

As shown in Equation 4.8, the *high average* rank is determined only within the specified threshold. If the rank of the pair P_i with respect to a given pair P_r is beyond the specified threshold, we assign $Rank[P_r, P_i]$ as the value $T + 1$. In contrast, $Rank[P_r, P_i]$ will be assigned the appropriate rank of P_i if it occurs within the threshold.

Table 4.6. Example-3: Independent High Rank-based Selection

	P_1	P_2	P_3	P_4	P_5	P_6
P_1	0	2	3+1=4	1	3+1=4	3+1=4
P_2	1	0	2	3+1=4	3+1=4	3+1=4
P_3	1	2	0	3+1=4	3+1=4	3+1=4
P_4	2	3+1=4	1	0	3+1=4	3+1=4
P_5	2	3+1=4	3+1=4	3+1=4	0	1
P_6	3+1=4	3+1=4	3+1=4	2	1	0
High Average	**1.66**	**2.66**	**2.5**	**2.5**	**2.83**	**2.83**

Considering the pairs in Table 4.3 for instance, let the threshold be set to $T = 3$. Table 4.6 shows, for each pairs $P_i \in \{P_1, ..., P_6\}$, the computed *high average* rank. Considering pair P_1, it occurs beyond the top-3 ranked pairs in the ordered list of pairs for P_6; thus, from Equation 4.8, $Rank[P_6, P_1]$ is assigned as 4 (i.e., threshold+1). In contrast, since P_1 occurs within the top-3 ranked pairs of P_2, $Rank[P_2, P_1]$ is assigned the actual rank (i.e., 1) of P_1 with respect to P_2. The *high average* rank for P_1 is then computed using Equation 4.7, and is shown in Table 4.6. The computed *high average* ranks for the remaining pairs are also shown in Table 4.6.

It can be observed that based on the computed ranks, two workload configurations are possible – $\{P_1, P_3\}$, and $\{P_1, P_4\}$ (since P_1 has the highest *high average* rank, and P_3 and P_4 both attain the next highest rank). From Equation 4.2, $\{P_1, P_3\}$ will yield a goodness of 1.16; in contrast, $\{P_1, P_4\}$ generates a goodness of 1. Therefore, this approach will select $\{P_1, P_4\}$ as the desired set $\mathbf{W}_K$.

It can be observed that the threshold specified in Equation 4.7 is T. Determining the exact threshold, within which the *high average* ranks of all pairs can be computed, is not straightforward. For instance, when the sizes of $\mathcal{Q}$ and $\mathcal{U}$ are large, setting a threshold of a small size may not be productive since there may not exist sufficient number of pairs which occur more frequently within this threshold. Consequently, most pairs will attain a *high average rank* of $T + 1$, and selecting an appropriate $\mathbf{W}_K$ from these equal ranked pairs will be difficult. In contrast, setting a threshold of very large sizes may produce the same difficulties as seen in the *Independent Rank-based Selection approach* i.e., pairs which receive good ranks for certain pairs but poor ranks for the remaining pairs will yield a lower value for the corresponding *high average* ranks; thus, making it difficult to determine an appropriate $\mathbf{W}_K$.

Therefore in this approach, we set the threshold T as the size of the set $\mathbf{W}_K$. As pointed out in Section 4.2.1, K is chosen based on pragmatic constraints. Further, since K is neither too small nor too large, we believe setting this value as the threshold can yield an acceptable solution. This is further validated by our experiments.

Time Complexity: Like the previous approach, this technique also determines the requisite ranks using the orderings provided by the similarity model; hence, the worst-case time complexity to determine $\mathbf{W}_K$ will be O($|\mathcal{P}|$) for both – *static* and *dynamic* environments.

4.3.3 Cumulative Selection

Based on the ranks of user-query pairs as shown in Table 4.3, it is evident that selecting either $\{P_1, P_5\}$ or $\{P_1, P_6\}$ will give a workload with optimal goodness. However, based on workloads using the *average* as well as the *high average* ranks from the *Rank-based Selection* as well as the *High Rank-based Selection* approaches, neither P_5 nor P_6 is selected due to their corresponding low ranks. However, considering pairs P_1 and P_5, it can be observed that when the rank of P_1 is poor (in the ranked list of pairs for P_5 and P_6), the rank of P_5 is good, and vice versa. In contrast, considering P_1 and P_3, it can be observed that when the rank of one of them (say P_1) is poor, the other pair (P_3) does not receive a particularly high rank; therefore the workload comprising of P_1 and P_3 does not produce a goodness as good as the one produced when P_1 and P_5 are considered.

```
Input: $\mathcal{Q}, \mathcal{U}, \mathcal{P}, \mathbf{W}_K = \oslash$
Output: $\mathbf{W}_K$
1 $\mathcal{G}(\mathcal{Q} * \mathcal{U}, \mathbf{W}) = |\mathcal{Q}| * |\mathcal{U}| - 1$
2 for $k$ = 1 to $K$ do
3     for $i$ = 1 to $|\mathcal{P}|$ do
4         Determine $\mathcal{G}(\mathcal{Q}, \mathcal{U}, \mathbf{W}_K \cup P_i)$
5     end
6     Select $P_x \in \mathcal{P} - \mathbf{W}_K$ such that –
7     $\forall_{P_y \in \mathcal{P} - \mathbf{W}_K} (\mathcal{G}(\mathcal{Q}, \mathcal{U}, \mathbf{W}_K \cup P_x) \leq \mathcal{G}(\mathcal{Q}, \mathcal{U}, \mathbf{W}_K \cup P_y)$
8     Add $P_x$ to $\mathbf{W}_K$
9 end
10 Return $\mathbf{W}_K$
```

Algorithm 2: Cumulative Selection Algorithm

The above discussion follows the notion that for determining $\mathbf{W}_K$ of optimal goodness, we considered the pairs P_1 and P_5 together, instead of considering them individually and measuring their effect on the goodness. This intuition forms the basis of our final approach; termed as the *Cumulative Selection* approach. This approach too avails of the ranked lists of pairs, and determines the necessary $\mathbf{W}_K$ over K iterations. Starting with an empty $\mathbf{W}$, in each iteration, it will select a pair P_i and add it to $\mathbf{W}$ such that for any other pair P_j, given that P_i, $P_j \in \mathcal{P}$ and P_i, $P_j \notin \mathbf{W}_K$, the following condition hold: $\mathcal{G}(\mathcal{Q},\mathcal{U}, \mathbf{W}_K \cup P_i) \leq \mathcal{G}(\mathcal{Q},\mathcal{U}, \mathbf{W}_K \cup P_j)$. The details of this approach are shown in Algorithm 2.

Consider the pairs from *Example-3* shown in Table 4.3. Table 4.7 shows the results of Algorithm 2 for 2 iterations. In the first iteration, P_1 that yields the best goodness value amongst all pairs will be selected and added to $\mathbf{W}_K$. In the next iteration, the algorithm will pick a pair such that when combined with P_1, the best goodness value can be obtained.

Table 4.7. Example-3: Cumulative Selection Process

Iteration 1						
Current $\mathbf{W}_K$	P_1	P_2	P_3	P_4	P_5	P_6
Goodness	**1.83**	2.33	2.16	2.33	3.16	3.16
Iteration 2						
Current $\mathbf{W}_K$	P_1, P_2	P_1, P_3	P_1, P_4	P_1, P_5	P_1, P_6	
Goodness	1.66	1.16	1	**0.83**	**0.83**	

Given that the combination of both – P_1, P_5, and P_1, P_6 yield the best (and in this case, the optimal) goodness, the algorithm will select one of these configurations to represent $\mathbf{W}_K$.

Time Complexity: Unlike the previous approaches, where the ranks of individual pairs could be computed by the similarity model (while establishing the pairwise similarities between user-query pairs), the *cumulative selection* approach cannot determine $\mathbf{W}_K$ while establishing the similarities. This is due to the fact that, based on Equation 4.2, the goodness of $\mathbf{W}_K$ can only be estimated once the pairwise rankings between all pairs are available. Further, this technique determines $\mathbf{W}_K$, unlike its independent counterparts, in an incremental fashion wherein the choice of a pair in each iteration depends on the previously chosen pairs that form the currently established $\mathbf{W}_K$.

Given that the time to compute the goodness of $\mathbf{W}_K$, comprising of a single user-query pair, is O($|\mathcal{P}|$), the worst-case time complexity to determine a workload of size K (from a total of $|\mathcal{P}|$ pairs) will be O($K * |\mathcal{P}|^2$) – in a static as well as dynamic environment.

Heuristic Analysis: Since the *cumulative selection* approach considers combinations of user-query pairs (instead of picking them individually) to assess their impact on the goodness of the $\mathbf{W}_K$, it is expected to perform better than the previous two *independent* approaches.

However, since the cumulative process involves multiple iterations and hence is computationally expensive than its independent counterparts (which are relatively straightforward as far as computations are concerned), there is a definite trade-off in terms of the computation time versus the quality of goodness yielded by these approaches. One possibility would be to initially establish $\mathbf{W}_K$ (in a static environment) using the *cumulative* approach, and shift to the *independent high rank-based* approach when additional users, queries and functions are added to the system (i.e., in a dynamic environment).

4.4 Experimental Evaluation

We have evaluated the *quality/accuracy* of each of our heuristic algorithm for determining $\mathbf{W}_K$ with appropriate goodness in a static as well as dynamic setting. Additionally, we also evaluated the *efficiency* of of each of these algorithms in terms of the time to generate the resulting configuration of $\mathbf{W}_K$.

4.4.1 Setup

Selecting the set of K pairs to represent the workload requires the availability of sets $\mathcal{Q}$ and $\mathcal{U}$. As alluded to in Chapter 3 (Section 3.6), no Web database provides user data and query logs that can be used for experimental evaluation. Hence, in order to experimentally validate the quality of our proposal, we had to rely on generating a synthetic workload of users and queries. For this, we relied on the two databases provided by Google Base – a *vehicle* database comprising of 8 distinct attributes (Make, Model, Vehicle-Type, Mileage, Price, Color, Location, and Transmission), and a *real estate* database with 12 attributes such as Location, Price, House Area, Bedrooms, Bathrooms, and so on.

On each database, we established the set $\mathcal{Q}$ by generating ten thousand queries with varying number of attributes, and involving different combinations of popular attributes and values within the respective domains. We then employed the *query similarity* component of the Similarity model (see Figure 3.1) to determine the individual pairwise similarities between these queries. Since obtaining data of real users is difficult, we generated a total of ten thousand synthetic users to establish the set $\mathcal{U}$. In order to estimate an initial *user similarity* between each pair of users, we randomly filled a small percentage (1%, 10%, 20%, etc.) of this workload with ranking functions. These functions were designed (by varying the attribute-weights) from an existing collection of ranking functions obtained via a survey conducted on real users over these two Web databases (in Section 3.6). Based on these computed similarities, we obtained an ordering of all users with respect to each other based on the method described in Section 4.2.3.

Our experiments were performed on a 2.6 GHz AMD Opteron Duo Core Processor machine with 4GB RAM running on a 32-bit Redhat Linux installation. All algorithms were implemented in Java.

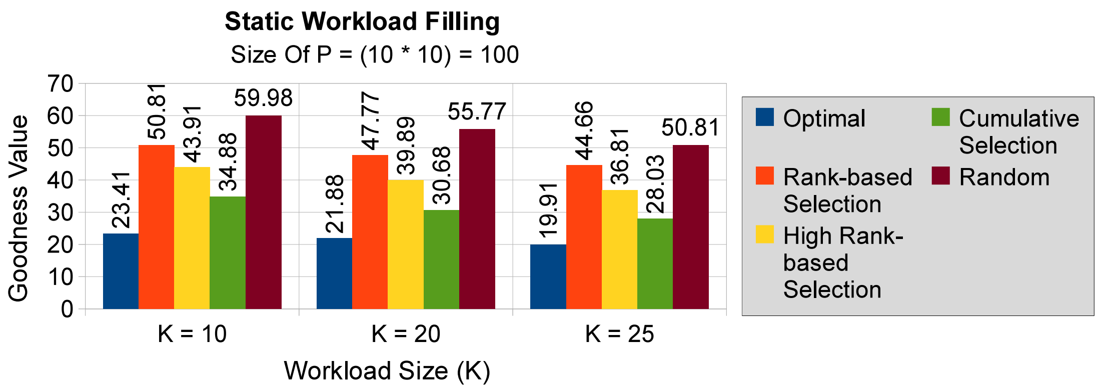
4.4.2 Quality Evaluation

Static Workload Filling: Our initial goal was to compare each of our heuristic approach against a brute-force solution in order to determine the difference in the goodness value estimated by each approach. However, for a $10K * 10K$ matrix, applying a brute-force approach to form $\mathbf{W}_K$ of a significant size (e.g., 1000 pairs) was computationally infeasible.

Hence, for the sole purpose of comparing with the optimal solution, we randomly selected a subset $\mathcal{Q}'$ X $\mathcal{U}'$ of size 10*10. We then randomly assigned ranking functions to 10% of these pairs (obtained from the survey results in [49]) to determine a preliminary set of user similarities. We then set a value of K = 10, and determined an optimal workload $\mathbf{W}_{opt}$ using a brute-force approach. Likewise, we determined $\mathbf{W}_{avg}$, $\mathbf{W}_{high}$, and $\mathbf{W}_{cum}$ (each of size 10) using our three proposed heuristics i.e., *independent rank-based*, *independent high rank-based* and *cumulative* respectively. For the *high rank-based* approach, we associated a threshold T of size 10. We also generated a random workload $\mathbf{W}_{rand}$ of 10 pairs to validate the need of an algorithmic approach to the *Workload Filling Problem*. We repeated this process for determining workloads of sizes K = 20, and K = 25 respectively to further illustrate the effectiveness of our approach.

Figures 4.3 and 4.4 display the goodness obtained for workloads generated using different strategies for the *vehicle* and *real estate* database respectively. It can be observed that even over a 10*10 set of pairs, the difference in the goodness values between the three proposed approaches is significant (over different sizes of workloads). Further, the results validate our intuition that the *cumulative* process outperforms the independent selection mechanism giving an average percentage improvement of 23.71% and 17.61% over the *high rank-based* approach for the goodness value of the *Vehicle* and *Real Estate* database respectively. Similarly, the *high rank-based* selection appears better than a *rank-based* selection (i.e., an average percentage improvement of 15.21% and 14.77% in the goodness values over the two respective databases). The significant point to note from these results is that our proposed approach produces goodness values that are comparable to the optimal value achieved by the brute force approach across workloads of different sizes. In addition, a random selection of pairs does not seem to be ideal while forming a workload for ranking in Web databases.

In order to further bring out the advantage of the *cumulative selection* approach over the *independent selection* variants, we employed the entire 10K*10K set of user-query pairs. Again, we randomly filled 1% of this matrix to determine the initial set of user similarities. We generated corresponding workloads, of size K = 10,000, for all three algorithms and determined their corresponding goodness values. Since obtaining the optimal solution for such a large matrix is intractable, we compared our approach to the *K-Means* clustering algorithm [70]. Given that for each user-query pair we can rank every user-query pair in the matrix, we clustered all the pairs in $\mathcal{Q} * \mathcal{U}$, as done in the *Cluster-Based User Similarity* model in Section 3.4.2. We generated a total of 10,000 clusters and selected the centroid of each cluster as the user-query pair to be selected for the $\mathbf{W}_{Clus}$ workload.



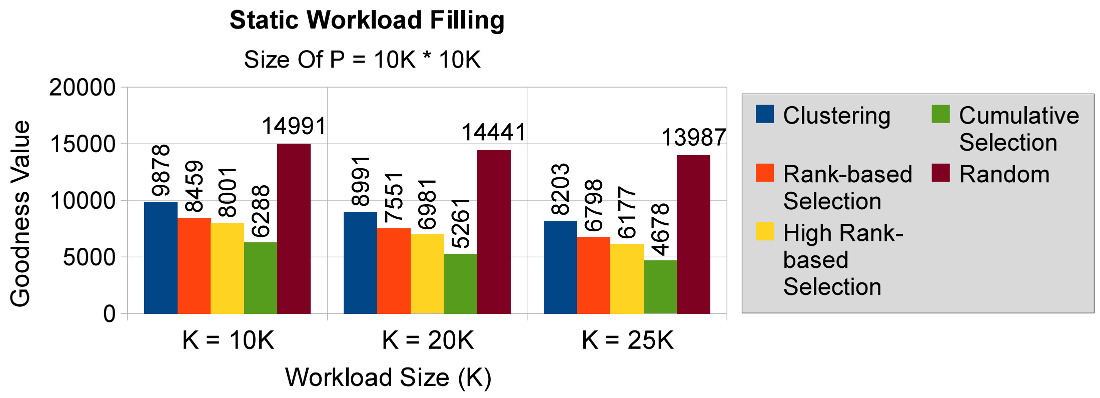
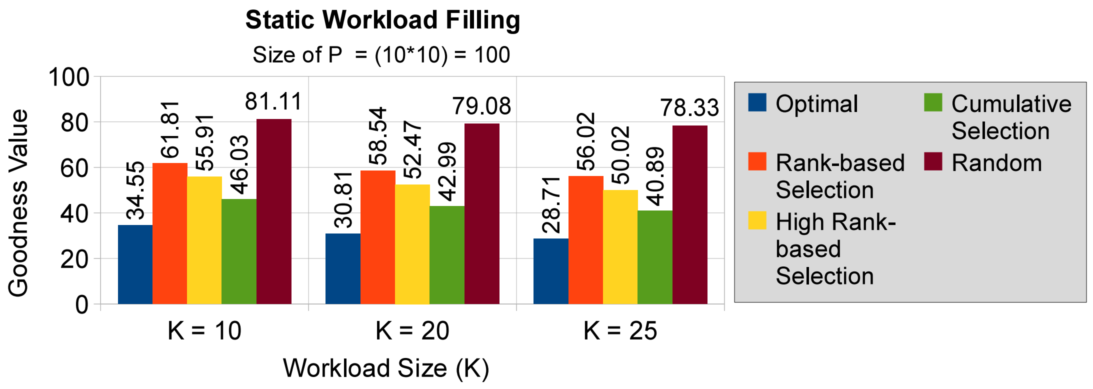


Figure 4.3. Evaluation For Static Filling: ***Vehicle*** Database.

We repeated this experiment for determining workloads of sizes 20,000 and 25,000 respectively. In each experiment, we also generated a random workload for further elucidating the need of a proper mechanism of establishing a workload.

Figures 4.3 and 4.4 show, for the *vehicle* and the *real estate* database respectively, the resulting goodness by applying the above techniques for workloads of different sizes. It is evident from the graphs that the *cumulative* process significantly outperforms the *independent* selection algorithms across workloads of all sizes (again an average improvement of 21.22% in the goodness over the *Vehicle* database); thus, validating our hypothesis. Further, it can be observed that the *high rank-based* approach performs better than the *average* rank approach. This is due to the fact that, for most cases, the highest *average* rank obtained has a value lesser than the one set for K (which is used by the *high rank-based* technique)



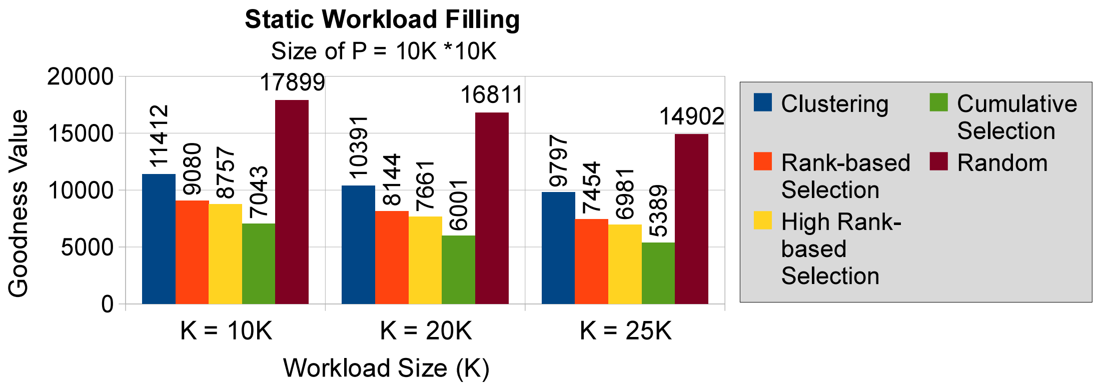


Figure 4.4. Evaluation For Static Filling: ***Real Estate*** Database.

thus, affecting the resulting goodness. The results also show that the heuristic algorithms distinctly outperform the clustering approach (by providing an average improvement of 20.22% in the goodness over the *Real Estate* database); thus, indicating that the computed centroids need not necessarily achieve good ranks in the ordered lists causing a low goodness value to be generated. Further, the random approach is significantly outperformed by our proposed solution.

Dynamic Workload Filling: The goal of this set of experiments was to validate the process of establishing an incremental workload i.e., in a dynamic fashion. Instead of varying the number of users and queries, we decided to incrementally add ranking functions, and hence, subsequently add these functions along with the corresponding user-query pairs to $\mathbf{W}_K$ for determining its effect on the workload filling process. Similar to the previous setting, we wanted to test this approach against an optimal workload obtained via a brute-force technique. Hence, we used the same 10*10 set of pairs, generated above, and assigned to 10% of these pairs, actual functions for the initial user similarity calculation. We performed three sets of experiments for determining a workload of size K = 10, 20, and 25 respectively. Each of the three heuristic algorithms used an increment K' of size 1 i.e., whenever a *single* ranking function is added to $\mathcal{Q}' * \mathcal{U}'$, user-similarities and the user-query pairs are reordered (based on the corresponding changes if any), and a fresh set of $K - K'$ pairs is computed. This process continues till the desired size of K is reached.

Figures 4.5 and 4.6 show the goodness obtained for workloads generated using different strategies in a dynamic environment for the *vehicle* and *real estate* database respectively. Similar to the static process, the difference in the goodness between the heuristic algorithms is distinctly evident. Further, all the three approaches (and, in particular, *cumulative selection*) produce a goodness value comparable with the optimal value obtained via the brute-force strategy.

In order to test the dynamic filling over larger sets of queries and users, we used the entire 10K*10K matrix. The goal was to determine workloads of sizes 10,000, 20,000 and 25,000 respectively. For each heuristic algorithm, we used an increment of size $K' = 10$ i.e., the algorithms recompute similarities and determine a new set of $K - K'$ pairs after incremental addition of 10 ranking functions to the existing workload. We compared the goodness obtained with the one obtained by using the clustering technique listed above, as well as with a randomly selected workload. Figures 4.5 and 4.6, similar to static filling, show the effectiveness of our proposal for the two databases respectively. It can be observed that even for a dynamic filling, the *cumulative* approach outperforms the *independent* selection process.

Static versus Dynamic Workload Filling: Figures 4.7 and 4.8 summarize, for the *vehicle* and the *real estate* database respectively, the comparison between the goodness value

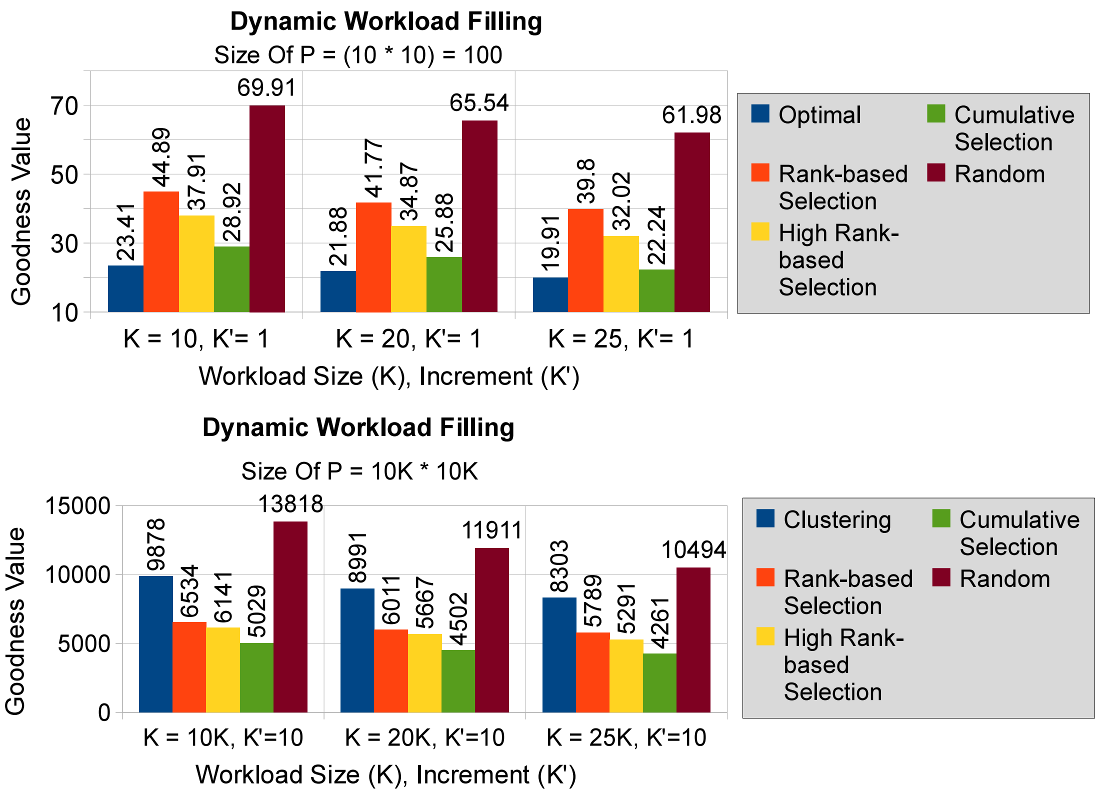
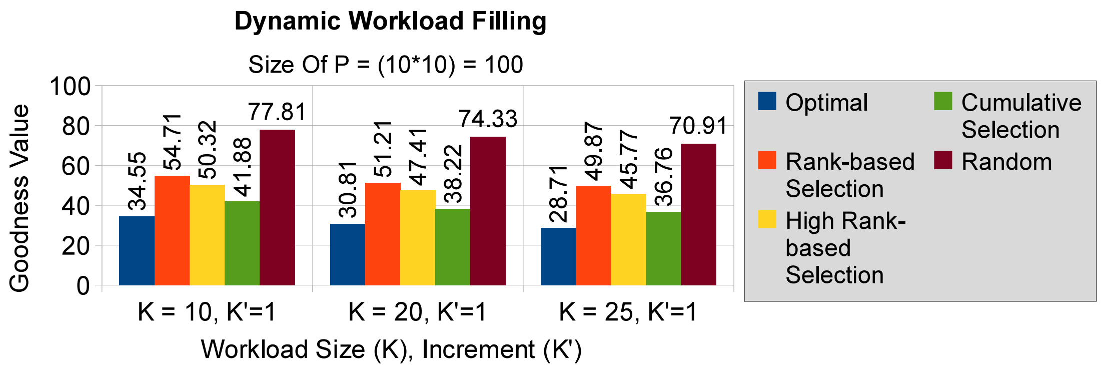


Figure 4.5. Evaluation For Dynamic Filling: ***Vehicle*** Database.

obtained by the heuristic algorithms in a **static** and a **dynamic** filling environment. The percentage improvement in the goodness value obtained by using the dynamic (instead of static) filling approach is shown in Table 4.8 and 4.9 for the two databases respectively. Given that user similarities in most Web databases will undergo frequent changes (due to addition of ranking functions, arrival of new users, and so on) the improvement provided by the dynamic filling makes it an automatic choice for establishing actual workloads in real Web database application.

4.4.3 Efficiency Evaluation

The goal of this study was to determine whether the proposed solution can be used for determining the requisite workload in a real-time application. The *Static Filling* ap-



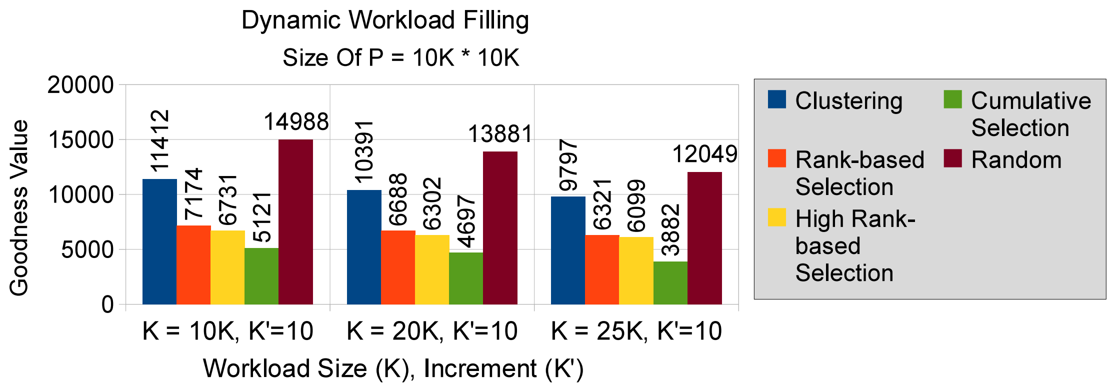


Figure 4.6. Evaluation For Dynamic Filling: ***Real Estate*** Database.

proach incurs only a one-time cost for determining the workload. However, the need for providing heuristic solutions stems from the difficulty in obtaining the optimal solution for Web databases having large sets of users and queries. For instance, it took us well over 12 hours (using an efficient multi-threaded program) to determine the optimal workload of size $K = 10$ over a small set of 10*10 user-query pairs. In contrast, the *Independent* approaches took less than 5 minutes whereas the *Cumulative* algorithm needs less than 15 minutes to determine the workloads of the same size.

In contrast, the *Dynamic Filling* process is computationally expensive since the process of determining the workload is repeated multiple times along with the re-computations for user similarities and obtaining the ranked lists for each pair. However, with a combination of hashing (to store ranked lists of users and queries separately) and an efficient

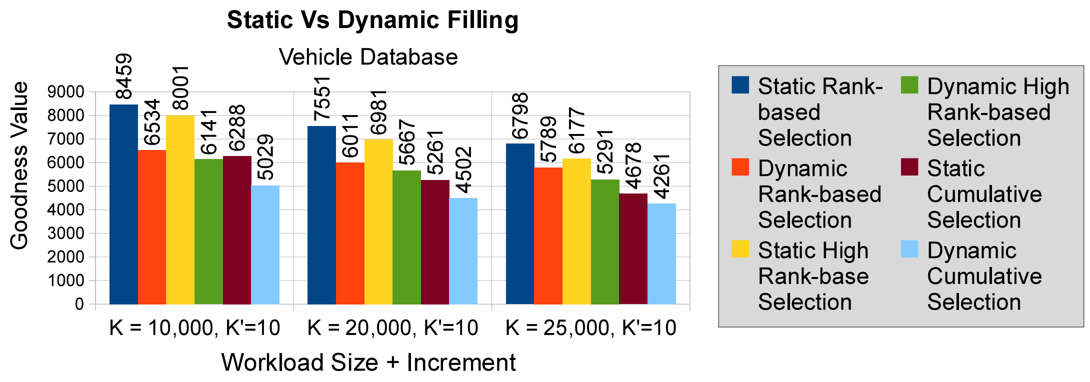


Figure 4.7. Static v/s Dynamic Workload Filling: ***Vehicle*** Database.

Table 4.8. ***Vehicle*** Database: % Improvement With Dynamic Filling

	$\|\mathbf{K}\|$ = 10,000	$\|\mathbf{K}\|$ = 20,000	$\|\mathbf{K}\|$ = 25,000
Independent Rank-based	15.84	20.51	22.76
Independent High Rank-based	16.67	18.82	23.24
Cumulative	11.91	16.41	20.02

priority queue (for managing the corresponding ranked lists of pairs), the time involved in performing this process is greatly reduced. For instance, over a 10K*10K set of pairs, the workload is computed whenever 10 functions are added to the existing workload. The overall process of recomputing similarities and determining a fresh workload can be performed in under 2 minutes. Further, by deferring the computation of user similarities, instead of determining them whenever a user asks a query, the time to determine the requisite pairs can be substantially reduced; albeit by incurring a small loss of the quality of workload goodness.

4.5 Conclusion

In this Chapter, we proposed a solution to the *Workload Filling* problem i.e., determining a “good” set $\mathbf{W}_K$ of K user-query pairs to represent a workload for assisting *user-*

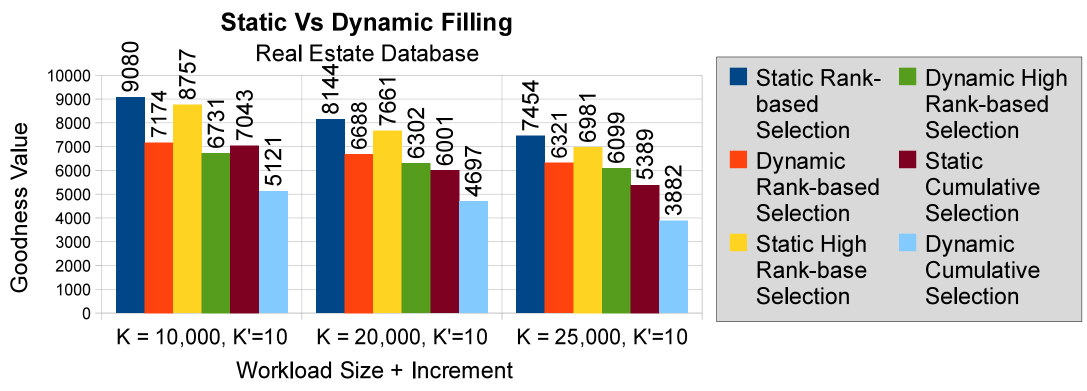


Figure 4.8. Static v/s Dynamic Workload Filling: ***Real Estate*** Database.

Table 4.9. ***Real Estate*** Database: % Improvement With Dynamic Filling

	$\lvert\mathbf{K}\rvert$ = 10,000	$\lvert\mathbf{K}\rvert$ = 20,000	$\lvert\mathbf{K}\rvert$ = 25,000
Independent Rank-based	15.21	17.87	20.99
Independent High Rank-based	13.63	17.73	23.13
Cumulative	19.83	21.77	27.28

and *query*-dependent ranking on Web databases. In order to quantify the quality of the workload and the subsequent ranking, we proposed a novel metric of *Workload Goodness*. We further demonstrated that finding an optimal workload is intractable in practice; and to overcome this challenge, we proposed a suite of heuristic algorithms. Further, we showed the applicability of our approach in determining the requisite workload in a *static* as well as *dynamic* environment. We analytically explained the effectiveness of our proposal and validated it experimentally over two real Web databases.

We now present the extensions to our proposed models of similarity i.e., a generic model of query similarity to span all types of SPJ queries, and a combined model of user similarity based on dynamic browsing choices and static user profiles.

CHAPTER 5

AN EXTENDED MODEL OF QUERY- AND USER-SIMILARITY

As alluded to in the earlier Chapter, the emergence of the deep Web provided a new connotation to the concept of ranking database query results. In order to address this challenge, we chose to depart from the earlier approaches for ranking (that resorted to either a query-dependent model by analyzing frequencies of database values and query logs, or a user-dependent model that utilized user profiles), and instead, advanced a novel similarity-based framework for performing query- and user-dependent ranking in Web databases. Although proven to be applicable in the context of two real Web databases in the domains of automobiles and real estate, the current framework has certain drawbacks that limit its applicability across any generic traditional and/or Web database.

The focus of this Chapter thus, is to address these drawbacks and advance appropriate solutions for the same. Specifically, we extend the currently proposed model of query similarity (specifically, query-condition similarity) so as to support the computation of pair-wise query similarity for arbitrary Select-Project-Join (SPJ) queries. Additionally, we present a combined user similarity model that blends static user profiles with the users' subsequent behavior over the database to fully support user-dependent ranking across all kinds of users. We discuss the effectiveness of our proposal analytically as well as experimentally over two Web databases.

5.1 Introduction

An important contribution of this dissertation is the similarity-based ranking framework (elaborated in Chapter 3) for performing user- and query-dependent ranking in the context of deep Web sources. This framework is based on the two proposed models of: *Query Similarity* and *User Similarity*. However, the current definitions of these models have certain drawbacks, listed below, that limit the applicability of this ranking framework across any arbitrary Web and/or traditional database.

1. **No support for query conditions with *range* and *IN* operators:** The existing *query similarity* component can **only** compute similarities between queries with **point** conditions (e.g., “Make = Honda”, “Color = Red”, etc.). However, many Web users would prefer to express their intent in a query involving range conditions (e.g., “Price $<$ 10K”) as well as IN conditions (e.g., “Make IN $\{$Honda, Toyota, Nissan$\}$”). Thus, extending the *query similarity* component to compute similarity between query conditions with range and IN operators is essential.
2. **No support for query conditions with *disjunction* and *negation* operators:** The current model of *query similarity* supports **only** those queries whose conditions are separated by a **conjunctive** (AND) operator (e.g., “Make = Honda AND Location = Dallas, TX”). However, given the diversity of Web users, it is likely that some users may have specific preferences for which supporting the negation (NOT) operator (e.g., “Make=Toyota AND NOT Model = Corolla”) as well as the disjunctive (OR) operator (e.g., “Make = Nissan AND (Price $<$ 5,000 OR Year $>$ 1994)”) becomes necessary. Thus, supporting query conditions separated by either of the – AND, OR and NOT operators and extending the *query similarity* component to support computation of similarities between such queries is vital.
3. **No support for *projection* and *join* predicates:** The existing *query similarity* component assumes a single relation database (which seems sufficient for most Web databases), and further reckons that users are interested in all attributes of this relation. However, in general, a database will have multiple relations and user queries may be cast (internally by the Web application) to span multiple relations using the

JOIN operator. Likewise, users may be interested in viewing the values associated with a smaller and more relevant subset of attributes (rather than a union of all attributes across all relations), and these preferences may be expressed (again, internally by the Web application) via a PROJECTION predicate in the query. Therefore, extending the *query similarity* component such that it can determine similarities between *join* as well as *projection* predicates (in addition to the *selection* predicates) thus, becomes important.

4. **No usage of functional dependencies and attribute correlations:** While comparing similarity between queries, the *query similarity* model currently assumes attribute independence. However, utilizing the knowledge of functional dependencies (e.g., Model $\rightarrow$ Make) and attribute correlations (e.g., high-priced cars typically have low mileage and vice versa) can further strengthen the similarity computation and hence should be incorporated in the model of *query similarity*.

5. **Similarity between users computed using *only* ranking functions:** The model of *user similarity* presently calculates pairwise similarities between users based on the similarity between their respective ranking functions over the set of common queries asked by them. This model is dynamic, since addition of ranking functions to the workload can influence the corresponding changes in the similarities between users. However, in certain cases (e.g., a user having very few or no prior ranking functions in the workload), determining his/her similarity with the remaining users may not be possible; thus, diminishing the effect of this model towards finding a good ranking function for these class of users. Therefore, extending the current component to incorporate static user profiles for ensuring user-dependent ranking uniformly across all users (new and/or experienced) is important.

Thus, the overall motivation of this Chapter is to extend the existing similarity-based ranking model (of Chapter 3) to a comprehensive and complete model that addresses all aspects of *query-* and *user-similarity* computations. Specifically, we propose an appropriate solution for each of the drawback listed above.

Contributions: The contributions of this Chapter are -

- Extending the current similarity-based ranking framework to a complete and comprehensive model to encompass all aspects of query- and user-similarity computations.
- Broaden the current model of *query similarity* to support queries with conditions containing selection, join and projection predicates as well as conjunction, disjunction and negation operators, coupled with the incorporation of the knowledge of functional dependencies and attribute correlations
- Modify the existing model of *user similarity* to include the notion of static user profiles (in addition to existing dynamic function-based profiles) for supporting user-dependent ranking for new as well as experienced users.
- An elaborate set of experimental results on real Web databases with the aid of real users (obtained from Amazon Mechanical Turk) that establishes the quality of our proposed solution.

Roadmap: In Sections 5.2 and 5.3, we respectively elaborate on our proposed approaches for extending the current *query* and *user similarity* models. Section 5.4 presents the results of our experimental evaluation, and we conclude in Section 5.5.

5.2 An Extended Model Of Query Similarity

In this section, we elaborate on the details that extend the current definition of the *query similarity* model (described in Section 3.3 of Chapter 3) to support the computation of similarity between any pair of arbitrary SPJ queries[1].

[1]It must, however, be noted that this extended model does not cover queries involving GROUP-BY and HAVING predicates. Furthermore, Aggregate operators are also not supported in this extension.

It is important to not that unlike its *query-condition* counterpart, the *query-result* similarity model (Section 3.3.2) does not require any extension and is capable of determining the similarity between any two queries; irrespective of the type of conditions and operators expressed in those queries. The reason being that this model establishes a similarity by *only* comparing the results produced in response to the given pair of queries. However, apart from a lack of applicability (demonstrated in Sections 3.3.3 and 3.6.3.1 of Chapter 3), the *query-result* model suffers from an extensive overhead if applied in a real-time scenario.

For instance, consider an incoming query Q and a large log of queries $\{Q_1$, Q_2, ..., $Q_N\}$ against which the similarity of Q is to be computed. Assuming that T tuples are obtained as the result for each query (in the log as well as for Q) and the database schema comprises of p attributes, the process of determining the similarity between Q and any query Q_i (in the log) would incur a time complexity of O $(T^2 * p) \approxeq$ O (T^2) (since $T >> p$). Consequently, determining the similarity of Q with every query in the log would take a total time of the order of O $(N * T^2)$. Furthermore, this complexity does not include the time for computing the result of the queries. Thus, although simplistic in nature, this model of *query-result* similarity renders itself to be impractical to use in practice.

Hence, the focus of this Chapter is to extend the *query-condition* model (Section 3.3.1), proven to be applicable (based on the experimental results in Section 3.6.3.1) and intuitive (as shown in Section 3.3.3) for real Web databases. Specifically, we extend this model such that computation of similarity between any pair of arbitrary SPJ queries would be accordingly possible. We first detail the preliminaries for extending this model, and then present the approach for the same.

5.2.1 Preliminaries

Consider a database *D* comprising of relations – $\{R_1$, R_2, ..., $R_r\}$ wherein the schema of each relation R_i ($\in$ *D*) consists of attributes – $\{A_{i_1}$, A_{i_2}, ... $A_{i_p}\}$. Further, the domain of an attribute A_{i_j} represents values – $\{v_{i_j}^1$, $v_{i_j}^2$, ..., $v_{i_j}^n\}$. Note that for the sake of simplicity, we assume the schema of each relation contains p attributes, and the domain of each of these attributes is of size n. However, in practice these numbers may vary for each relation and the corresponding attributes.

Let $\mathcal{Q}$**:** $< \mathcal{R}$**,** $\mathcal{S}$**,** $\mathcal{P}$**,** $\mathcal{J} >$ represent a typical SQL query, whose selection and join predicates are assumed to represented in the Conjunctive Normal Form, over D where:

- $\mathcal{R}$: $\{R_1$, R_2, ..., $R_w\}$ represent the set of relations specified in the FROM clause,
- $\mathcal{P}$: $\{P_1$, P_2, ... $P_x\}$ represents the set of projection attributes in the SELECT clause where each P_i ($\in \mathcal{P}$) is an attribute within the schema of a relation in *D*,
- $\mathcal{S}$: $\{C_1, C_2$, ..., $C_y\}$ represents the set of selection conditions (in the WHERE clause) where each $C_i \in \mathcal{S}$ is either
  - a single predicate: C_i, or
  - a disjunction predicate: (C_{i_1} OR ... OR C_{i_w}), or
  - a negation predicate: NOT C_i, and

  each individual predicate C_i (or C_{i_k} in case of disjunction) is:
  1. a point condition (e.g., $R_1.A_{1_4}$ = $v_{1_4}^{10}$), or
  2. a range condition (e.g., $R_2.A_{2_5}$ $\{<, <=, >, >=\}$ $v_{2_5}^{15}$), or
  3. an IN condition (e.g., $R_3.A_{3_6}$ IN $\{v_{3_6}^{16}, v_{3_6}^{17}, v_{3_6}^{18}\}$).
- $\mathcal{J}$: $\{J_1$, J_2, ... $J_z\}$ represents the set of joins (in the WHERE clause) where each $J_i \in \mathcal{J}$ is a join condition (e.g., "$R_1.A_{1_3}$ OP $R_2.A_{2_4}$" where OP $\in \{=, <=, <, >, >=, !=\}$).

Based on this setup, given two queries – $\mathcal{Q}$: $< \mathcal{R}$, $\mathcal{S}$, $\mathcal{P}$, $\mathcal{J} >$, and $\mathcal{Q}'$: $< \mathcal{R}'$, $\mathcal{S}'$, $\mathcal{P}'$, $\mathcal{J}' >$, our goal is to extend the current *query-condition* model such that it can compute similarity between these two queries.

5.2.2 Query Re-writing

The *query-condition* similarity model currently compares the conditions between a given pair of queries for determining the resulting similarity. However, based on its definition (provided in Section 3.3.1 of Chapter 3), to compare two conditions within a given pair of queries, it requires that both these conditions be formed on the same attribute of a given relation. Although strict, this constraint (of comparing two conditions only when they formed on the same attribute) is justified since comparing a condition like "Make=Toyota" with "Make=Honda" is more intuitive than comparing it with a condition like "Color=Blue". However, expecting all queries to be formed on an identical set of attributes over the same of relations is definitely unrealistic. For instance, consider the following pair of (Join/Cartesian product) queries:

- Q_1: SELECT * FROM R_1, R_2 WHERE $R_1.A_{1_1} = v_{1_1}^{10}$
- Q_2: SELECT * FROM R_1, R_2 WHERE $R_2.A_{2_2} = v_{2_2}^{11}$

It is evident that while Q_1 and Q_2 are on the same set of relations (R_1 and R_2), there is no condition on A_{1_1} in Q_2 with which we can compare the condition "$R_1.A_{1_1} = v_{1_1}^{10}$" in Q_1; similarly, there is no condition on A_{2_2} in Q_1. Thus, establishing similarity between this pair of queries would not be possible. Intuitively, based on the conditions (on different attributes) in Q_1 and Q_2, they are less likely to be similar to each other. However, in order to ensure a consistent model wherein any given pair of queries can be compared for similarity, we rewrite the above queries. This is done by appending *missing relations* and *missing predicates* to them such that, given two initial queries $\mathcal{Q}$ and $\mathcal{Q}'$, their rewritten versions have –

- $\mathcal{R} = \mathcal{R}'$,
- $|\mathcal{S}| = |\mathcal{S}'|$, and $S_i \in \mathcal{S}$ as well as the corresponding $S_i' \in \mathcal{S}'$ are specified on the same attribute/s of a given relation, and

- $|\mathcal{J}| = |\mathcal{J}'|$, and $J_i \in \mathcal{J}$ along with $J_i' \in \mathcal{J}'$ are stipulated over the same attributes within a given set of relations.

While a common set of relations can be appended to both queries in a straightforward manner (i.e., by ensuring $\mathcal{R}$ (as well as $\mathcal{R}'$) = $\mathcal{R} \bigcup \mathcal{R}'$), in order to append conditions on identical attributes, we analyze the intent behind the specified query. For instance, given Q_1, since no condition is specified on A_{2_2}, we can presume that any value returned for this attribute in the result tuples is acceptable. Consequently, we can append a condition of the form: "$R_2.A_{2_2}$ IN $dom(A_{2_2})$" in conjunction to the existing condition in Q_1. Following this notion, we re-write the above queries as:

- Q_{1_r}: SELECT * FROM R_1, R_2

  WHERE $R_1.A_{1_1}$ = $v_{1_1}^{10}$ AND $R_2.A_{2_2}$ IN $dom(A_{2_2})$
- Q_{2_r}: SELECT * FROM R_1, R_2

  WHERE $R_1.A_{1_1}$ IN $dom(A_{1_1})$ AND $R_2.A_{2_2}$ = $v_{2_2}^{11}$

As is evident, comparing a specific value to the entire domain of an attribute will yield a poor value of similarity; thus ensuring the overall similarity computed between Q_{1_r} and Q_{2_r} to be a very small value. Thus, in-spite of re-writing and appending conditions, the overall intuition (that the similarity between these queries is low) is not lost. The above re-written queries only represent *selection* predicates. However, as we shall see later in Section 5.2.5, *join* predicates are transformed into selection predicates (using the IN operator); hence, explicit re-writing of join predicates is not necessary.

The process of appending a condition with the attribute value being specified as its entire domain can be further refined using the knowledge of functional dependencies and attribute correlations in providing a better value for the attribute (than assigning the entire domain). We address the utilization of dependencies and correlations for re-writing queries later in Section 5.2.6. It must be noted that process of re-writing queries to append relations and predicates is performed ***only*** for the sake of computing similarities between queries.

The actual queries (and **not** their rewritten versions) are the ones specified on the database to yield and display the results to the users.

5.2.3 Projection Similarity

Projection refers to the set of attributes whose corresponding values the user prefers to be displayed as the output of the query. For queries involving multiple relations, projections attributes will be typically expressed as a combination of the relation and its attribute (e.g., "SELECT $R_1.A_{1_2}$, $R_2.A_{2_5}$ FROM R_1, R_2 WHERE ..."). However, for the sake of simplicity, we assume that the projection attributes in the queries $\mathcal{Q}$ and $\mathcal{Q}'$ are represented by sets – $\mathcal{P} = \{P_1, P_2, ...P_x\}$, and $\mathcal{P}' = \{P'_1, P'_2, ...P'_y\}$, where each P_i (and P'_i) is of the form: $R_a.A_{a_b}$, with $R_a \in \mathcal{R}$ and $A_{a_b} \in schema(R_a)$.

The intuitive similarity between these two projections can be estimated based on the number of common attributes listed in them. For instance, if $\mathcal{P}$ and $\mathcal{P}'$ represent an identical set of attributes, it is evident that the tuples corresponding to the same set of attributes is desired across both queries; thus, indicating that two projections are similar (and in this case, equivalent). In contrast, if $\mathcal{P}$ and $\mathcal{P}'$ are disjoint, it indicates the preferences towards the output variables (in terms of attributes and their values) are completely different for the two; hence, allowing us to conclude that the projection predicates are not similar. Consequently, we can conclude that higher the overlap between $\mathcal{P}$ and $\mathcal{P}'$, greater will be the similarity between the projections, and vice-versa.

Formally, for queries $\mathcal{Q}$ and $\mathcal{Q}'$, the similarity between the respective projection predicates – $\mathcal{P}$ and $\mathcal{P}'$ is computed as the Jaccard coefficient [76] between the two sets –

$$sim(\mathcal{P}, \mathcal{P}') = \frac{|\mathcal{P} \bigcap \mathcal{P}'|}{|\mathcal{P} \bigcup \mathcal{P}'|} \tag{5.1}$$

This similarity between the projection predicates varies from a value of 1 (when $\mathcal{P}$ and $\mathcal{P}'$ are identical) to a value of 0 (when they are disjoint).

5.2.4 Selection Condition Similarity

Given $\mathcal{Q}$ and $\mathcal{Q}'$ in their re-written forms (as shown in Section 5.2.2), let their respective selection conditions be:

- $\mathcal{S}$: "WHERE C_1 AND C_2 AND ... AND C_x"
- $\mathcal{S}'$: "WHERE C_1' AND C_2' AND ... AND C_x'"

where C_i and C_i' represent the condition/s on the attribute/s of the same relation/s; however C_i and C_j (and correspondingly, C_i' and C_j') can be specified on the attributes of different relations. Given that we assume a Conjunctive Normal Form, the similarity between the selection conditions (represented as $sim(\mathcal{S}, \mathcal{S}')$) is established (as per the notion of *query-condition similarity* defined in Section 3.3.1 of Chapter 3) as the conjunction of the individual similarities computed between each pair of selection conditions. Formally,

$$sim(\mathcal{S}, \mathcal{S}') \quad = \quad \prod_{i=1}^{x} sim(C_i, C_i') \tag{5.2}$$

Now, in order to determine the right hand side of the above equation, we need a mechanism to determine similarity between a given pair of conditions. However, a given condition C_i (and C_i') could either be a single predicate, a negation of a single predicate (i.e., expressed by the NOT operator), or a disjunction of multiple single predicates (i.e., separated by the OR operator). Further, each predicate can contain a point (=), a range ($<, <=, >, >=$), or the IN operator. Therefore, we need to establish a model that can determine similarity between any pair of conditions comprising of either of the above specified cases. We start by extending the current query-similarity model for single predicates, and subsequently expand it to address negation and disjunction predicates.

5.2.4.1 Single Point Predicates

Consider single *point* conditions, C_1 ($\in \mathcal{S}$) and C'_1 ($\in \mathcal{S}'$) respectively, of the form –

- C_1: "WHERE $R_1.A_{1_1}$ = $v^1_{1_1}$"
- C'_1: "WHERE $R_1.A_{1_1}$ = $v^2_{1_1}$"

As per Equation 3.2 (in Section 3.3.1 of Chapter 3), the similarity between these conditions is estimated as the similarity between the corresponding values $v^1_{1_1}$ and $v^2_{1_1}$ i.e.,

$$sim(C_1, C'_1) = sim(v^1_{1_1}, v^2_{1_1}) \tag{5.3}$$

5.2.4.2 Single IN Predicates

Consider a sample pair of single *IN* conditions, C_2 ($\in \mathcal{S}$) and C'_2 ($\in \mathcal{S}'$) respectively, of the form –

- C_2: "WHERE $R_1.A_{1_2}$ IN $\{v^2_{1_2}$, $v^3_{1_2}$, $v^4_{1_2}\}$"
- C'_2: "WHERE $R_1.A_{1_2}$ IN $\{v^5_{1_2}$, $v^6_{1_2}\}$"

We adhere to the same procedure established for single *point* predicates i.e., we analyze the similarity between the values specified in the conditions via the IN operator. If we separately apply both given conditions on the database, two sets of tuples – $N_{2,3,4}$ (for C_2) and $N_{5,6}$ (for C'_2) will be obtained.

Applying the intuition of the single point predicates described above i.e., the similarity between two sets of results eventually equates to the similarity between the corresponding conditions, we can state:

$$sim(C_2, C'_2) = sim(N_{2,3,4}, N_{5,6}) \tag{5.4}$$

Since the values – $\{v^2_2$, v^3_2, v^4_2, v^5_2, $v^6_2\}$ correspond to the domain of the same attribute (A_{1_2}), the tuples corresponding to each of these values will be disjoint. As a result, we can represent $N_{2,3,4}$ and $N_{5,6}$ as: $N_{2,3,4}$ = $N_2 \bigcup N_3 \bigcup N_4$, and $N_{5,6}$ = $N_5 \bigcup N_6$. Therefore, the similarity between the two sets of result tuples – $N_{2,3,4}$ and $N_{5,6}$, can be evaluated as the mean of the aggregate similarities between each of the sets $\{N_2, N_3, N_4\}$ (in $N_{2,3,4}$) with respect to each of the individual sets $\{N_5, N_6\}$ (in $N_{5,6}$). This follows the principle that the similarity of N_2 with $N_{5,6}$ is the average aggregate similarity between the tuples of N_2 with the tuples of N_5 and the tuples of N_6.

However, we know from Equations 5.3 and 3.6 that similarity between two sets of tuples (e.g., N_2 and N_5) corresponds to the eventual similarity between the attributes values (i.e., v_2^2 and v_2^6) representing these sets of tuples. Using this knowledge, we generalize the similarity computation between two single IN predicates. Given C_2 and C_2' respectively of the form[2]:

- C_2: "WHERE $R_1.A_{1_2}$ IN $\{v_{1_2}^1, v_{1_2}^2, ..., v_{1_2}^r\}$"
- C_2': "WHERE $R_1.A_{1_2}$ IN $\{v_{1_2}^1, v_{1_2}^2, ..., v_{1_2}^s\}$"

the similarity between these two conditions is calculated as:

$$sim(C_2, C_2') = \frac{\sum_{i=1}^{r} \sum_{j=1}^{s} sim(v_{2_2}^i, v_{2_2}^j)}{r \cdot s} \tag{5.5}$$

where, the similarity between individual pairs of attributes values can be computed using Equation 5.3.

5.2.4.3 Single Range Predicates

Consider a sample pair of single *range* conditions, C_3 ($\in \mathcal{S}$) and C_3' ($\in \mathcal{S}'$) respectively, of the form –

- C_3: "WHERE $R_1.A_{1_3} < v_{1_3}^7$"
- C_3': "WHERE $R_1.A_{1_3} > v_{1_3}^8$"

[2]Without loss of generality, it can be assumed that $\{v_2^1, v_2^2, ..., v_2^r\}$, and $\{v_2^1, v_2^2, ..., v_2^s\}$ are the set of values specified in C_2 and C_2', although they can be any values in the domain of attribute A_{2_2}.

Given that $dom(A_{1_3})$ represents all the values in the domain of attribute A_{1_3}, let $dom(C_3)$ ($\subset dom(A_{1_3})$) wherein each element in $dom(C_3)$ represents an element whose numerical value is less than $v_{1_3}^7$. Likewise, let $dom(C_3')$ ($\subset dom(A_{1_3})$) such that each element in this subset is numerically greater than $v_{1_3}^8$. Consequently, conditions C_3 and C_3' can be respectively rewritten as –

- C_{3^1}: "WHERE $R_1.A_{1_3}$ IN $dom(C_3)$"
- C_{3^2}': "WHERE $R_1.A_{1_3}$ IN $dom(C_3')$"

Given that these two conditions C_{3^1} and C_{3^2} now represent single IN predicates specified over a set of values corresponding to $dom(C_3)$ and $dom(C_3')$ respectively, the similarity between them can be computed using Equation 5.5. We generalize this notion of similarity between any two *range* predicates as follows – Given any two single range conditions C_3 and C_3' respectively of the form:

- C_3: "WHERE $R_1.A_{1_3}$ OP $v_{1_3}^a$"
- C_3': "WHERE $R_1.A_{1_3}$ OP $v_{1_3}^b$"

where OP $\in \{<, <=, =>, >\}$, the similarity between C_3 and C_3' is determined as:

$$\begin{aligned} sim(C_3, C_3') &= sim(dom(C_3), dom(C_3')) \\ &= \frac{\sum_{i=1}^{|dom(C_3)|} \sum_{j=1}^{|dom(C_3')|} sim(v_{1_3}^i, v_{1_3}^j)}{|dom(C_3)| \cdot |dom(C_3')|} \end{aligned} \tag{5.6}$$

where, $dom(C_3)$ and $dom(C_3')$ represent the set of values satisfying the conditions – $R_1.A_{1_3}$ OP $v_{1_3}^a$ and $R_1.A_{1_3}$ OP $v_{1_3}^b$ respectively. Furthermore, the pairwise similarity between the values in the given domains can be computed using Equation 5.3.

5.2.4.4 Negation Predicates

Let C_4 ($\in \mathcal{S}$) and C_4' ($\in \mathcal{S}'$) represent two *negation* predicates of the form –

- C_4: "WHERE **NOT** $R_1.A_{1_4}$ OP V_4"
- C_4': "WHERE **NOT** $R_1.A_{1_4}$ OP V_4'"

where V_4 and V_4' either represent a single value (e.g., v_9^{14} and v_{10}^{14} respectively) if OP $\in \{<, <=, =>, >, =\}$, or a set of values when OP represents the IN operator.

Now, let $\overline{dom(C_4)}$ and $\overline{dom(C_4')}$ be subsets of $dom(A_{1_4})$ that satisfy the respective conditions – "WHERE $R_1.A_{1_4}$ OP V_4" and "WHERE $R_1.A_{1_4}$ OP V_4'". Given that the original two conditions involve the **NOT** operator, the resulting set of tuples for C_4 and C_4' would be – $\{dom(A_{1_4})$ - $\overline{dom(C_4)}\}$, and $\{dom(A_{1_4})$ - $\overline{dom(C_4')}\}$ correspondingly. Thereby, the similarity between the two negation predicates C_4 and $C_4')$ can be estimated as:

$$sim(C_4, C_4') \quad = \quad sim(\{dom(A_1^4) - \overline{dom(C_4)}\}, \{dom(A_1^4) - \overline{dom(C_4')}\}) \qquad (5.7)$$

The right hand side of above equation is similar to the right hand side of Equation 5.6 (i.e., they both represent IN conditions over a set of values) for single range predicates, and thus, can be accordingly determined using Equation 5.6.

5.2.4.5 Disjunction Predicates

Consider a sample pair of two disjunction conditions C_5 ($\in \mathcal{S}$) and C_5' ($\in \mathcal{S}'$) respectively, of the form:

- C_5: "WHERE ($R_1.A_{1_1}$ OP V_1 OR $R_1.A_{1_2}$ OP V_2)"
- C_5': "WHERE ($R_1.A_{1_1}$ OP V_1' OR $R_1.A_{1_2}$ OP V_2')"

where OP $\in \{<, <=, =>, >, =, IN\}$, and V_1, V_2, V_1', and V_2' represent either a single or a set of values accordingly. Now, let $N_{1,2}$ and $N_{1,2}'$ represent the set of tuples that respectively represent the conditions C_5 and C_5' in the database. Given that each condition is on the values of different attributes, unlike the case of single IN predicates, we have: $N_{1,2}$ = $N_1 \bigcup N_2$, and $N_{1,2}'$ = $N_1' \bigcup N_2'$.

Further, the similarity between N_1 and N_1' represents the similarity between V_1 and V_1'. Likewise, the similarity between N_2 and N_2' equates to the similarity between V_2 and V_2'. Since these two conditions represent disjunction of single predicates, the similarity between the individual predicates can be computed using Equations 5.3, 5.5 or 5.6 (depending on whether the predicates contain point, range or IN operators). Further since the above-specified conditions adhere to the disjunctive normal form, the overall similarity between the complete conditions can be computed as a summation of the similarities between the individual predicates.

On a generalized note, given two disjunction conditions C_5 and C'_5 of the form –

- C_5: "WHERE (C_{5_1} OR C_{5_2} OR ... OR C_{5_k})"
- C'_5: "WHERE (C'_{5_1} OR C'_{5_2} OR ... OR C'_{5_k})"

the similarity between these conditions is computed as:

$$sim(C_5, C'_5) = \sum_{i=1}^{k} sim(C_{5_i}, C'_{5_i}) \tag{5.8}$$

5.2.5 Join Condition Similarity

Given $\mathcal{Q}$ and $\mathcal{Q}'$, let the respective join conditions in these queries be of the form:

- $\mathcal{J}$: "WHERE J_1 AND J_2 AND ... AND J_z"
- $\mathcal{J}'$: "WHERE J'_1 AND J'_2 AND ... AND J'_z"

where, we assume J_i and J'_i represent the join condition/s on the same attribute/s of two relations albeit over different operators; however, J_i and J_k (and correspondingly, J'_i and J'_k) could be joins on different attributes of different relations. Now, consider two join conditions J_i and J'_i specified in $\mathcal{Q}$ and $\mathcal{Q}'$ of the respective form –

- J_i: "WHERE $R_a.A_{a_1}$ OP_x $R_b.A_{b_2}$"
- J'_j: "WHERE $R_a.A_{a_1}$ OP_y $R_b.A_{b_2}$"

where OP_x and $OP_y \in \{<, <=, =>, >, ==, != \}$. Considering J_i, the attribute A_{a_1} is joined with A_{b_2} using operator OP_x. The result of this join represents the set of values (denoted as $V_{1,2}$) in the domain of A_{a_1} that satisfy this join condition. Specifically, we can assume that the tuples containing these values are the ones of eventual interest. Likewise, let the set $V'_{1,2}$ represent the values in the domain of A_{a_1} that are desired to occur in the tuples generated by the join conditions J'_i. Thereby, the original join conditions can be re-expressed as –

- J_{i_r}: "WHERE $R_a.A_{a_1}$ IN $V_{1,2}$"
- J'_{i_r}: "WHERE $R_a.A_{a_1}$ IN $V'_{1,2}$"

Thus, the given join conditions can be expressed as single predicates containing the IN operator over those attribute values that represent the resulting join operation. It must be noted that, the given query with the join condition will be the one that will be actually applied on the database for obtaining the requisite results. The process of converting a join condition to a non-join IN predicate is done *only* to establish similarity between two join conditions in the given two queries. Furthermore, this process can be performed for all join attributes over different operators, assuming the knowledge of join-compatible attributes amongst the relations in the schema, at the time of establishing the pairwise similarities between attributes values.

Correspondingly, the similarity between the join conditions J_i and J'_i can then be computed as:

$$sim(J_i, J'_i) = sim(R_a.A_{a_1}\, \mathit{IN}\, V_{1,2}, R_a.A_{a_1}\, \mathit{IN}\, V'_{1,2}) \tag{5.9}$$

The right hand side of the above equation can be calculated using Equation 5.5. Thus, given that $\mathcal{J}$ = $\{J_1,\, J_2,\, ...\, J_z\}$ and $\mathcal{J}'$ = $\{J'_1,\, J'_2,\, ...\, J'_z\}$ are represented in the Conjunctive Normal Form, the *join similarity* between them is given as:

$$sim(\mathcal{J}, \mathcal{J}') \;=\; \prod_{i=1}^{z} sim(J_i, J'_i) \tag{5.10}$$

where the right hand side of the equation is determined using Equation 5.9.

5.2.6 Incorporating Correlations & Dependencies

So far we assumed that we do not have any additional information about the database schema. However, some semantic information in the form of functional dependencies may be available. It may also be possible to determine correlation between attribute values from the extant database. During query re-writing (Section 5.2.2), we insert a missing predicate by specifying the entire domain as the value for the given attribute. However, it is possible that this attribute is functionally dependent and/ or correlated to other attributes; in such cases, inserting an appropriate value instead of the entire domain may assist in strengthening the similarity computations between the queries.

5.2.6.1 Incorporating Attribute Correlations

We explain this process via the example of a *vehicle* database. In the context of this database, it is to be expected that there is a strong correlation between the attributes – "Price" and "Mileage" i.e., *high* priced cars typically have *low* mileage whereas *low* priced cars have a considerably *high* mileage. Hence, a query asking for a low priced vehicle will be very similar to a query asking for a vehicle with high mileage, and vice versa.

For instance, consider, Q_a: "Make = Honda AND Price = 5K-10K dollars", and Q_b: "Make = Nissan AND Mileage = 125K-150K miles". Based on the scheme elaborated above for inserting *missing predicates*, the two queries will be re-written as –

- Q'_a: "Make = Honda AND Price = 5K-10K dollars AND Mileage IN $dom(Mileage)$"
- Q'_b: "Make = Nissan AND PRICE IN $dom(Price)$ AND Mileage = 125K-150K miles"

Now, the similarity between the above two queries will be computed such that for Mileage attribute, the value of "125K-150K" will be compared to all the values in the domain of Mileage, and likewise for the Price attribute. However, it is obvious that comparing a specific value (e.g., “125K-150K”) to the entire domain of an attribute will yield a much lower value of similarity. In contrast, if the attributes are correlated, specific value/s of one attribute will correspond to specific value/s of the second attribute. For instance, a car with a mileage between “125K-150K” miles is typically priced in the ranges of “\$ 5K-10K” and “\$ 0-5K” whereas a car priced in the range of “\$ 5K-10K” will be in the mileage range of “100K-125K” and “125K-150K”. Therefor, the above two queries can be rewritten as –

- Q''_a: “Make = Honda AND Price = \$ 5K-10K AND Mileage IN {100K-125K, 125K-150K}”
- Q''_b: “Make = Nissan AND PRICE IN \$ {0-5K, 5K-10K} AND Mileage = 125K-150K miles”

Comparing the respective conditions will yield a higher and a more accurate value of similarity than the one obtained if such correlation was not considered. Such correlation between attributes and the corresponding correlated values can be obtained by using the Pearson’s chi-square test [77] for numerical attributes, and its variant [78] for categorical attributes in Matlab [79]. Specifically, Matlab’s correlation toolbox [79] generates a coefficient of correlation in the range of -1 to +1. For a given pair of attributes A_{a_i}, A_{a_j}, the correlation coefficient is –

1. 0 if the attributes are independent,
2. in the range of 0 to +1 if they are positively correlated, and
3. in the range of 0 to -1 if they are negatively correlated.

Further, for each distinct categorical/numerical value in A_{a_i}, the Matlab function *candgen* return the set of matching value/s in A_{a_j} based on the correlation (an empty set is returned if coefficient is 0), and likewise returns the set of matching values in A_{a_i} correlated to a specific value in A_{a_j}. Using this knowledge of attribute correlation between every pair of attributes in the database schema, the *missing predicates* can be inserted using an appropriate value for a given attribute, instead of it’s entire domain.

5.2.6.2 Incorporating Functional Dependencies

In the context of vehicle databases, the dependency “Model $\rightarrow$ Make” always holds true. Similar to the above pair of queries, consider that we have two queries – Q_c: “WHERE Model = Camry”, and Q_d: “WHERE Make = Toyota”.

Based on the current definition of similarity, “Camry” will be compared to each value in the domain of Model, whereas “Toyota” will be compared to each value in the domain of attribute Make. This will lead to a very poor value of similarity between these two queries; although all Camry vehicles, being of the make Toyota, should have a higher similarity with Toyota vehicles than the one computed presently. Using the appropriate functional dependency, we know that the model “Camry” will be identified by a unique value of Make (in this case, Toyota). Hence, instead of finding the average similarity of “Toyota” to cars of all Make, we can rewrite the queries as –

- Q'_c: “WHERE Make = Toyota AND Model = Camry”
- Q'_d: “WHERE Make = Toyota AND Model IN $dom(Model)$”

The similarity between these conditions, and hence the queries would be higher and more intuitive than the one obtained if dependencies were not assumed. However, unlike attribute correlations, functional dependencies cannot be derived automatically. For this, we assume that certain domain knowledge and meta-data is available for the underlying database from which, an appropriate set of dependencies can be perceived. Using such knowledge of dependent attributes, and their values, the *missing predicates* can be inserted in an appropriate manner.

5.2.7 The Complete Query Condition Similarity Model

Given $\mathcal{Q}$: $< \mathcal{R}$, $\mathcal{S}$, $\mathcal{P}$, $\mathcal{J} >$, and $\mathcal{Q}'$: $< \mathcal{R}'$, $\mathcal{S}'$, $\mathcal{P}'$, $\mathcal{J}' >$, based on the formalization provided in Sections 5.2.4, 5.2.5, and 5.2.3, we have at our disposal, the similarities between – ($\mathcal{S}$, $\mathcal{S}'$), ($\mathcal{J}$, $\mathcal{J}'$), and ($\mathcal{P}$, $\mathcal{P}'$).

Further, given a SPJ query, we have assumed that $\mathcal{S}$ and $\mathcal{J}$ (as well as $\mathcal{S}'$ and $\mathcal{J}'$), are represented in the WHERE clause as a combined Conjunctive Normal Form. Therefore, following the definition of similarity established for Selection and Join conditions, the overall similarity of all conditions in the WHERE clause is a conjunction of these two similarities. In contrast, it is important to factor in the the Projection similarity with the overall Condition similarity since two queries having identical conditions may have disjoint projection attributes. In this scenario, the queries will generate altogether different results; hence, claiming that the two queries are similar may not be intuitive.

Therefore, we establish the overall similarity between the given pair of queries as the conjunction of the Projection similarity with the product of the Selection and Join similarities. Formally,

$$sim(\mathcal{Q}, \mathcal{Q}') = sim(\mathcal{P}, \mathcal{P}') \cdot (sim(\mathcal{S}, \mathcal{S}') \cdot sim(\mathcal{J}, \mathcal{J}')) \tag{5.11}$$

Thus, given any two queries – $\mathcal{Q}$: $< \mathcal{R}, \mathcal{S}, \mathcal{P}, \mathcal{J} >$, and $\mathcal{Q}'$: $< \mathcal{R}', \mathcal{S}', \mathcal{P}', \mathcal{J}' >$, the **extended** *query-condition* model can compute similarity between these two queries using Equation 5.11.

5.3 An Extended Model Of User Similarity

As described in Section 3.4 (of Chapter 3), the current model for *user-similarity* in the ranking framework computes a similarity between a given pair of users, by *only* considering the set of respective ranking functions obtained over the common queries asked by these users. This model departs from the traditional forms of user similarity computed typically using static profiles of users [28] [29] [30] [31] [35] [36], and hence, is dynamic in nature since it adjusts the similarities between users based on the changes in the users' behavior (in form of their ranking functions). However, this model's applicability is limited in situations when the pair of users being compared have no queries common between them – a realistic scenario in the context of Web database applications involving new and/or infrequent users.

In order to support *user-dependent* ranking uniformly across all types of users (new, infrequent as well as frequent), we extend the current model into a unified model made up of two components – i) dynamic user model, and ii) static user model. The former ensures that it accounts for changes in similarities between users based on their subsequent interactions (and hence, their derived ranking functions) with the Web databases, while the latter ensures that similarity can be computed even for those users (new or infrequent) who may not have too many functions associated with them with respect to the Web database.

5.3.1 Combined User Similarity

The proposed extension to the current model of *user similarity* transforms it into a unified model with a dynamic and a static component. The dynamic user model is adopted directly from the current definition of *user-similarity* given by Equation 3.7. We represent this similarity between a given pair of users, say – U_a and U_b, as $sim_D(U_a, U_b)$.

In contrast, the static user model relies on the profile information collected from users by most Web database applications such as Yahoo!, Google, Amazon, and others. Given that, a user U_a's profile can be represented as a vector – $\{u_{a_1}$, u_{a_m},..., $u_{a_m}\}$ over a set of m profile attributes (e.g., sex, location, profession, income-group, etc.) with each u_{a_i} representing the user's details for the i^{th} attribute, the similarity between a given pair of users – U_a and U_b, is represented as $sim_S(U_a, U_b)$, and can be computed by adopting the variant of cosine-similarity model given by Equation 5.12 (in Section 3.3) i.e.,

$$sim_S(U_a, U_b) = \sum_{i=1}^{m} sim(u_{a_i}, u_{k_i}) \tag{5.12}$$

where,

$$sim(u_{a_i}, u_{k_i}) = \begin{cases} 1 & \text{if } u_{a_i} = u_{k_i}, \\ 0 & \text{if } u_{a_i} \neq u_{k_i}. \end{cases} \tag{5.13}$$

It must be noted that various techniques (e.g., Jaccard coefficient, Support Vector Machines (SVM), and others) for determining the static similarity between users, based on profiles, can also be adopted instead of the proposed cosine-similarity computation. Given that there exists a large body of work for establishing profile-based user similarity, any suitable technique from these works can be plugged in to Equation 5.12. In the context of this dissertation, our primary goal was to combine the static and dynamic models, and not on devising new techniques for computing static user similarity. For the sake of consistency however, we use the variant of the cosine-similarity model for determining this similarity.

We now combine these two models into a single *user similarity* model. Toward that, consider the above two users U_a and U_b. Now the intuition behind a combined model is that, when either one or both these users are new, there exist no common queries between them. Correspondingly, $sim_D(U_a, U_b)$ is undefined (as per Equation 3.7)), and hence is assigned a value of 0 by the current *user-similarity* model; thus, rendering the dynamic model useless in this scenario. In contrast, $sim_S(U_a, U_b)$ will always exist (assuming the users have provided basic profile information), and hence the static model will be the lone driving force to establish similarity between these users.

Subsequently, assuming that both these users ask a common query (for which their respective ranking functions are obtained), computing $sim_D(U_a, U_b)$ would be possible. However, a single common query may not be sufficient to abandon the static model and simply use the dynamic model. In fact, the significance of the dynamic model should gradually, or rather, exponentially increase as the number of common queries between the two users increase. In contrast, the static model will always yield the same similarity, and hence its effect should be lessened as the common queries between the two users increases. Based on this intuition, we combined these two models wherein the effect of the dynamic model increases exponentially based on a factor given by the number of common queries, between these users, for which ranking functions have been deduced. Formally, the unified model of user similarity is given as:

Definition Given two users – U and U', and x ($\in \{0...n\}$) representing the number of queries common between them, the combined user similarity, represented as $sim(U, U')$, is given by Equation 5.14:

$$sim(U, U') = sim_S(U, U') + e^{ln(x)} * sim_D(U, U') \tag{5.14}$$

Based on Equation 5.14, the effect of the dynamic model will increase in an exponential fashion as the number of queries common between the users increases. Note that we take the exponent as a natural logarithm of x since this value of x can go in hundreds, or even thousands (although this may be difficult to achieve in practice).

We would like to point out that there exists a large body of work that establishes similarity between users based on profiles. Any of these techniques can be used (instead of the proposed static model) in Equation 5.14. The primary contribution of this work is the blending of a static profile-based similarity with a dynamic function-based similarity into a single model to support *user-dependent* ranking across all types of users.

5.4 Experimental Evaluation

We evaluated the extended models of *query-* and *user-similarity* for *quality/accuracy* in terms of the ranking framework presented in Chapter 3. Given that the (dynamic) model of *user-similarity* is based on ranking functions, testing this model in the context of ranking makes sense. However, the *query-similarity* model relies on the contents of the database for estimating similarity between queries; hence, in addition to testing it in the context of ranking, we also evaluated this extended model in isolation.

5.4.1 Setup and Workload Generation

For evaluation in the context of the ranking framework, we relied on the same two real Web databases provided by Google Base and used for the earlier evaluations in Chapters 3 and 4 i.e., the *vehicle* database comprising of 8 attributes (Make, Vehicle-Type, Mileage, Price, Color, etc.), and the *real estate* database with 12 categorical/discretized attributes (Location, Price, House Area, Bedrooms, Bathrooms, etc.). Further, given that Google exports a single relation for these database schema, we split this schema into three meaningful relations for both databases. For instance, the schema of the *vehicle* database was represented as – R_{v_1}: $\{$Make, Model, Vehicle-type$\}$, R_{v_2}: $\{$Model, Price, Mileage, Year$\}$, and R_{v_3}: $\{$Model, Transmission, Number-of-doors$\}$. A similar splitting of the *real estate* database was done to yield three relations for this schema as well.

Query Generation: In order to test the extended model of *query similarity* in the context of the ranking framework, we generated over both databases, a pool of 60 random queries (comprising of conditions based on randomly selected attributes and their values) on the single schema (i.e., assuming a single relation database) exported by Google Base. We then manually selected 21 representative queries that are likely to be formulated by real users for either database. Table 5.1 and 5.2 show for the two databases respectively, three such queries. We took sufficient care to ensure that a right mix of point, range, IN, disjunc-

Table 5.1. Sample Experimental Queries: ***Vehicle*** Database

Q_1	"Make IN $\{$Honda, Toyota$\}$ AND (Color = Blue OR Year $<$ 2009)"
Q_2	"Model IN $\{$Corolla, Civic$\}$" AND Location = Miami,FL AND Price$<$8,000$"
...	...
Q_{10}	"NOT Location=Chicago,IL AND (Mileage $<$ 100,500 OR Color = Red)"
...	...

tion and negation predicates for the *selection* conditions were involved in these queries. However, for this experiment, we did not employ the use of *join* and *projection* conditions, since we felt, that Web users typically are not capable of expressing queries involving these predicates.

In addition, based on the set of attributes exported by the schema of Google Base, we generated a small set of functional dependencies (e.g., "Model $\rightarrow$ Make", "Model $\rightarrow$ Vehicle-type" for Vehicle database, and "Zipcode $\rightarrow$ Location" for Real Estate database). Similarly, using Matlab, we established the set of correlated attributes and their corresponding values (e.g., "Price" and "Mileage are negatively correlated whereas "Color" and "Transmission" are independent in the Vehicle database; likewise, "Rent" and "Area" are positively correlated in the Real Estate database).

To test the extended *query similarity* model in isolation, we employed the multiple relations that were created (and shown above) for the two databases. Correspondingly, we generated a total of 1 Million queries that adhered to the SPJ semantics i.e., the queries had a right mix of *projection*, *selection*, and *join* conditions. The *selection* conditions further were designed to involve point, range, IN, disjunction and negation predicates. The dependencies and correlations, discussed in the above paragraph were retained for these queries as well.

User Generation: The users in these experiments comprised of actual users on Amazon Mechanical Turk. Each user provided us with a basic profile information (over 12 attributes

Table 5.2. Sample Experimental Queries: ***Real Estate*** Database

Q_1	"Location=Dallas,TX AND (Beds = 3 OR To = Buy)"
Q_2	"Location=Denton, TX AND (Beds => 2 OR Area > 1000 sq.ft)"
$\cdots$	$\cdots$
Q_{16}	"Location=Dallas, TX AND NOT Type=Townhouse AND To = Rent"
$\cdots$	$\cdots$

like Gender, Age-range, Location, Education, Profession, Marital Status, Favorite Color, and so on.). Further, due to the high level of security provided by Amazon, no privacy regarding the user was violated in terms of name, location, and identification.

We then conducted two separate surveys (one for each database) where every user was shown the 21 generated queries (one-at-a-time) and asked to input, for each query, a ranking function by assigning weights to the schema attributes (on a scale of 1 to 10). For aiding the user in expressing these preferences, we also displayed the set of results returned for each query. Each *explicit* ranking provided by a user for a particular query was then stored in the associated workload $\mathbf{W}$. The *vehicle* database survey was taken by 110 distinct users whereas the *real estate* database survey was taken by 115 distinct users who provided ranking functions for all the queries displayed to them. Thus, we generated a workload of 2310 ranking functions for the *vehicle* database and 2415 functions for the *real estate* database. Like we did for the evaluation of the similarity model in Chapter 3 (Section 3.6), we masked out 95% of the ranking functions from the workload i.e., the workload for the *vehicle* database consists of only 115 (5% of 1100) ranking functions, and the *real estate* database comprises of 120 function.

Our experiments were performed on a 2.6 GHz AMD Opteron Duo Core Processor machine with 4GB RAM running on a 32-bit Redhat Linux installation, and the algorithms were written in Java.

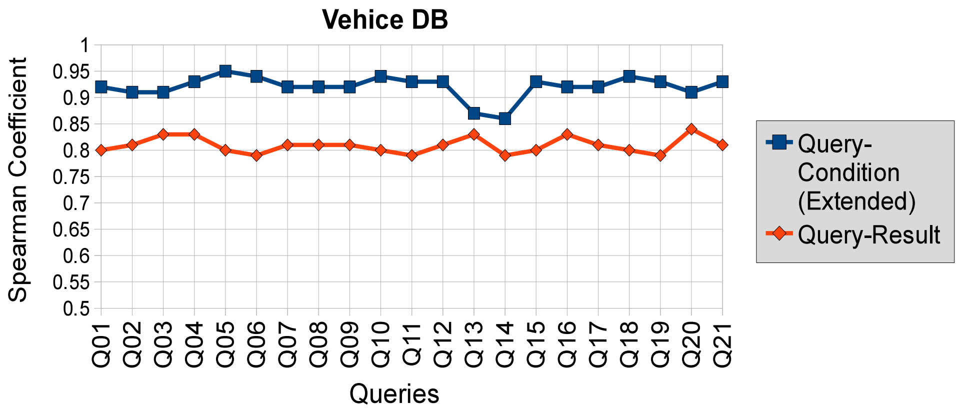


Figure 5.1. Quality Of Extended Query Similarity: ***Vehicle*** DB.

5.4.2 Query Similarity Evaluation

We first test the quality of this model in the context of the similarity-based ranking framework, and then evaluate it in isolation.

Given that we extended the *query-condition* model, we test the quality of ranking produced by this model versus the one obtained by applying the standard *query-result* model (that required no extensions). The quality is tested as follows: Given a query Q by user U, we obtain the most similar query Q_c (using the extended *query condition* model) and Q_r (using the *query result* model). We then apply the function $\mathcal{F}_{Q_c,U}$ and $\mathcal{F}_{Q_r,U}$ to the results of Q to obtain two sets of ranked results – R_c and R_r. We then apply the original ranking function $\mathcal{F}_{Q,U}$ (provided by U for Q) to the results of Q and get a ranked set of results R. We then determine the quality of these models as the *Spearman rank correlation coefficient* (Equation 3.8) between R and R_c, and between R and R_r. If the coefficient between R and R_c is greater than the one between R and R_r, our understanding that extending the *query-condition* model would be better and provide an improved quality of ranking than the one obtained by the *query-result* model will be validated.

We performed the above process for each of the user (110 and 115 for vehicle and real estate database) asking every query. Figures 5.1 and 5.2 show, for the two databases

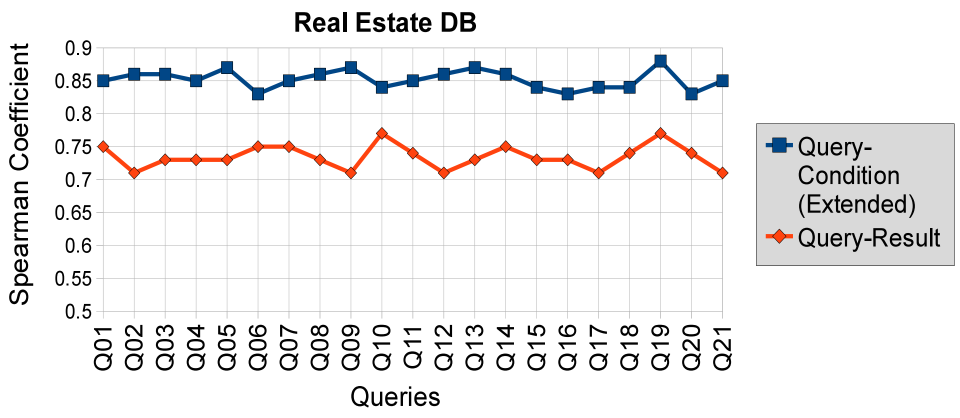


Figure 5.2. Quality Of Extended Query Similarity: ***Real Estate*** DB.

respectively, the *average query-condition* similarity (as well as the *average query-result* similarity) obtained across every query. The horizontal axis represents the queries; whereas the vertical axis represents the average value of the resulting *Spearman coefficient*. As the graph shows, over both the domains, the *query-condition* model outperforms the *query-result* model.

Departing from the ranking context, we tested the extended model of *query-condition* similarity in isolation. The motivation of this evaluation was to simply test the query similarity models without involving the ranking functions (and the quality of ranking thereafter). Primarily, the goal was to determine the exact values of similarity obtained by either of the two models versus the one obtained from a randomly selected query. Given that the value of similarity ranges from 0 to 1 (with 1 indicating equivalent or the most similar query), we evaluated the values yielded by these models for various queries on the database. Specifically, we use the 1 Million queries generated over either database. We then randomly generate additional 20 queries. For each of these 20 queries, we determine the most similar query (by comparing with the set of 1 Million queries) using the extended *query-condition* and the original *query-result* model. Likewise, for each query, we randomly select a query

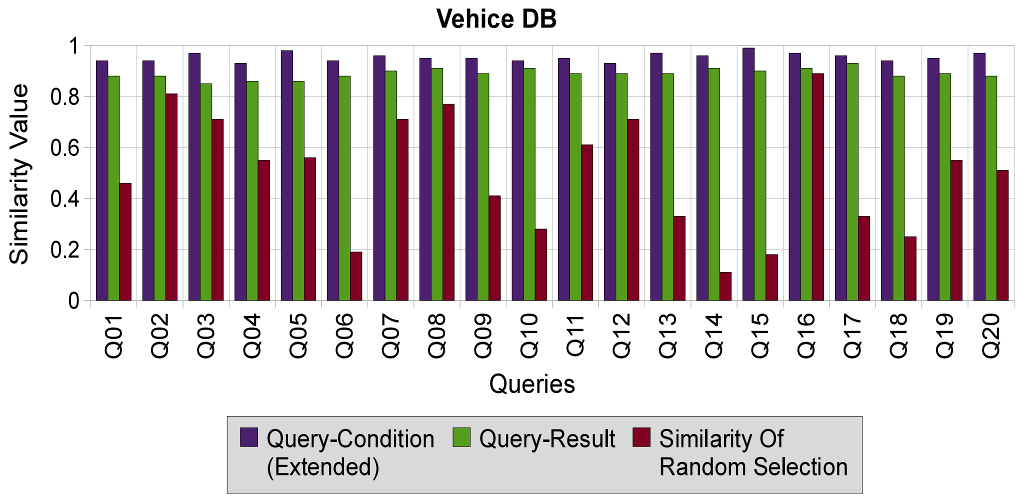


Figure 5.3. Evaluation Of Query Similarity Values: ***Vehicle*** DB.

(from the 1 Million queries) and compute its similarity with the former query using the *query-condition* model.

The results of this experiment are shown in Figures 5.3 and 5.4. The horizontal axis represents the 20 queries whereas the vertical axis represents the resulting value of query similarity (between 0 and 1). As seen in the figures, the *query-condition* model generates a more reasonable value of similarity as compared to its *query-result* counterpart. Furthermore, it can be clearly observed that employing a random model for selecting a query (either in the context of ranking or otherwise) does not seem to be a desirable approach.

5.4.3 User Similarity Evaluation

The goal of this evaluation was to validate that the unified model involving the dynamic as well as the static user similarity component would yield a better quality of overall ranking across a wide set of users than the one obtained if either of them were to be applied in isolation. In the workloads used for both the databases in the experimental setup, we have masked 95% of the ranking functions; therefore, there would be a large number of

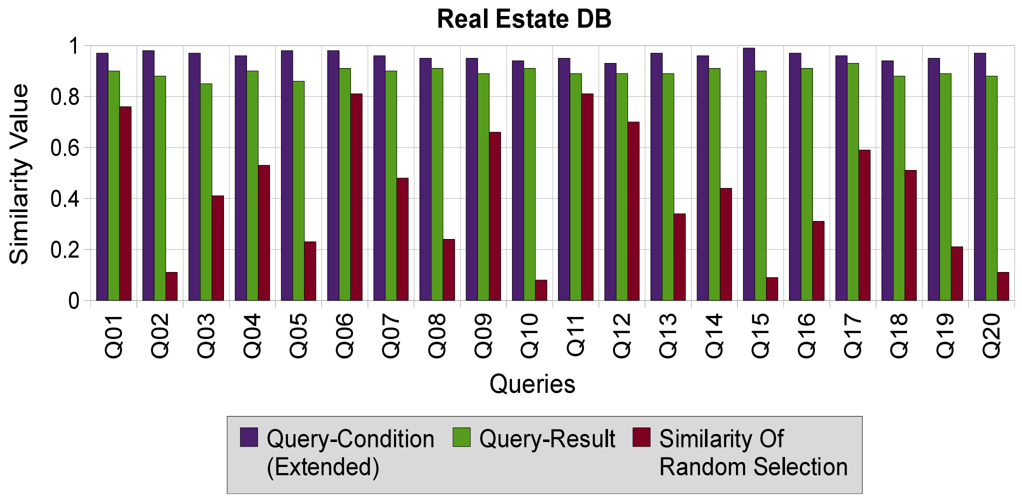


Figure 5.4. Evaluation Of Query Similarity Values: ***Real Estate*** DB.

users who would have very few queries common amongst them; thus reducing the utility of the dynamic user model. In contrast, the static model would establish a single ordering of users with respect to a given user across all queries; thus, not taking into account that changes in user behavior may reflect in the corresponding similarities between them. However, a combined model would, according to our hypothesis, yield a better value of similarity and hence, a finer quality of overall ranking.

The quality of this model is tested as follows: For a user U asking query Q, we determine the most similar user U_s using the static user model (i.e., based on user profiles), and the most similar user U_d based on the similarity of the ranking functions across the common queries between these two users (for this, we employed the *top-K* suite (presented in Section 3.4.3) of user similarity models). We apply $\mathcal{F}_{Q,U_s}$ and $\mathcal{F}_{Q,U_d}$ to the results of Q, and obtain two sets of ranked results – R_s and R_d. Likewise, we determine the most similar user U_c based on the unified model (as per Equation 5.14) and obtain the ranked set of results R_c by applying $\mathcal{F}_{Q,U_c}$ to Q's results. We then apply the original (masked) function $\mathcal{F}_{Q,U}$ to Q's results and obtain the ranked set of results R. The quality of these models is

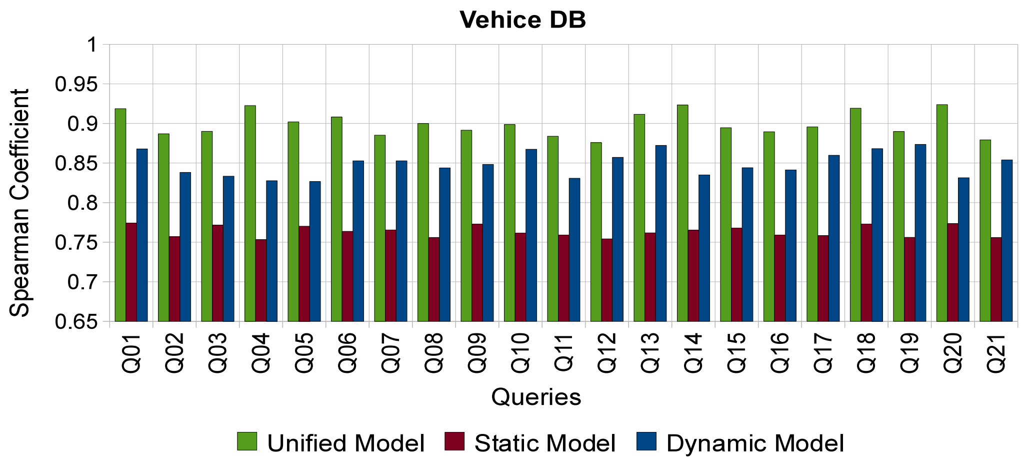


Figure 5.5. Quality Of Combined User Similarity: ***Vehice*** DB.

then evaluated as the *Spearman rank correlation coefficient* (Equation 3.8) between R and R_s, between R and R_d, and between R and R_c. If the coefficient between R and R_c is greater than the one obtained for the other two models, our understanding that a combined *user similarity* model would be better and provide an improved quality of ranking than the one obtained by any of the individual models would be validated.

We performed the above process for each of the users (110 and 115 for vehicle and real estate database) asking each of the 20 queries. Figures 5.5 and 5.6 show, for the *vehicle* and the *real estate* database respectively, the *average user* similarity obtained for the three models across every query. The horizontal axis represents the queries; whereas the vertical axis represents the average value of the resulting *Spearman coefficient*. As depicted by the results, over both databases, the combined *user similarity* model outperforms the individual static as well as the dynamic user models. Thus, these results verify the claim that over a Web database with sparse workloads, a model that employs the static information from user profiles combined with a model based on dynamic browsing behavior will prove to be more effective than applying either of the two models in isolation.

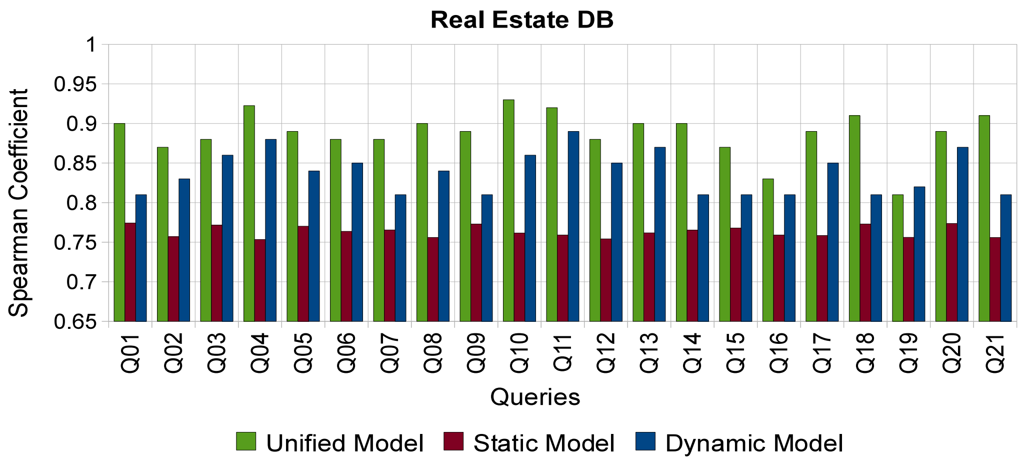


Figure 5.6. Quality Of Combined User Similarity: ***Real Estate*** DB.

5.5 Conclusion

In this Chapter, we explored the notion of *query similarity* in the context of database queries (SQL). Specifically, we proposed and elaborated on the details for extending the *query-condition* similarity model that analyzes the components such as selections, joins and projections between queries, to establish similarity between them. Likewise, we extended the notion of *user similarity* into a unified model comprising of a *static* (i.e., user profiles) and a *dynamic* (i.e., based on users' browsing behavior) component. We presented the experimental results over two Web databases and real Web users to corroborate the validity of our proposed extensions to these models.

CHAPTER 6

RELATED WORK

This Chapter presents the survey of the work related to the dissertation. We organize the related work into two categories – *intent specification* over the Web, and *ranking* in the context of different forms of information/data retrieval, and discuss their relevance to the work presented in this dissertation.

6.1 Intent Specification

Almost every form of data/information retrieval involves – the provision for users to specify their intent, a mechanism to capture and translate this intent into queries (to be posed over the retrieval systems), and some form of meta-data that allows these frameworks to surmise the specified intent. Correspondingly, there exists a considerable amount of work that independently address these aspects of intent specification over different forms of retrieval frameworks.

6.1.1 Query/Search Formulation

In the context of database systems, the earliest and the most relevant work in terms of formulating queries using templates/skeletons (with multiple interactions from the user) has been the popularly accepted paradigm of Query-By-Example (or QBE) paradigm [13]. In addition, template-based query formulation using multiple interactions with the user has been developed for database systems such as SQL Server, Oracle and Microsoft Access. Similarly, the CLIDE framework [80] adopts a multiple-interaction visual framework for allowing users to formulate queries. The primary motivation of the CLIDE architecture is to determine which queries would yield results versus those which produce a null result-set. However, formulating queries using these mechanisms requires the user to have knowledge about the types of queries supported by the underlying schema as well as a minimal understanding of the query language of the data model.

With the emergence of Web databases, most deep Web portals such as Expedia (`www.expedia.com`) or Amazon (`www.amazon.com`) relied on the QBE paradigm, in the form of templates, facets, and menus, to allows users to specify their intent. However, the queries to these systems are restricted to the schema of a single domain such as travel, shopping, and so on, and thus, lack the flexibility to support complex queries that span multiple domains. Furthermore, since the underlying schemas of the Web databases are well-defined and user queries are purely based on these schemas, the need to support arbitrary queries/intents with varying conditions on multiple types of operators does not arise.

Retrieval frameworks such as search engines (e.g., Google) and meta-search engines (e.g., Vivisimo [81]) use the keyword query paradigm and its success forms the motivation for our QBK approach. However, these search engine mechanisms do not convert the keywords into queries as their aim is to perform a lookup followed by a post-process of results (such as ranking, clustering, etc.) instead of sophisticated query processing. Although some search engines such as Ask (`www.ask.com`) and Kartoo (`www.kartoo.com`), and a few question-answering frameworks (e.g., START [5]) accept natural language input, we believe that they do not transform them into structured queries.

Frameworks that support queries on multiple sources employ either a keyword-based query paradigm (e.g., Havasu [15]) or mediated query interfaces (e.g., WHIRL [16]) for accepting user intents. Similarly, commercial systems such as Google Base [82] advocate the usage of keyword queries. Faceted-search systems such as DBLP (`www.dblp.l3s.de`) support query formulation using keywords in multiple navigational steps till the user gets the desired results. However, the focus of these frameworks is to perform a simple text/Web-search to obtain different types of data in response to the keywords (e.g., blogs, web-links, videos, etc.) instead of formulating a query where every keyword corresponds to a distinct entity.

In the context of information integration i.e., retrieval of information from multiple sources, existing frameworks (e.g., Ariadne [83], TSIMMIS [84], and WHIRL [85]) have extended the database querying models using combinations of templates or menu-based forms to incorporate queries that are restricted to a single domain (or a set of domains). Other frameworks (such as Havasu [86]) employ an interface similar to search engines, that take relevant keywords (associated with a concept) from the user and retrieve information for this particular concept from a range of sources. However, as the domains for querying established by these systems are fixed (although the sources within the domain might change), the problem of designing a querying mechanism is simplified to a great extent. When a more involved query needs to be posed, users may not know how to unambiguously express their needs and may formulate queries that lead to unsatisfactory results. Moreover, providing a rigid specification format may restrict the user from providing complete information about his/her intent.

Additionally, most of the above-mentioned frameworks fail to capture queries that involve a combination of *spatial*, *temporal*, and *spatio-temporal* conditions. A few systems (e.g., Hermes [87] and TerraWorld [88]) allow a limited set of spatial operations (such as *close to*, *travel time*) through its *push-button listing-based* interface or a *form-based* interface. Currently, centralized *Web-based mapping* interfaces like Google Maps (`www.maps.google.com`) and Bing Maps (`http://www.bing.com/maps/`) allow searching and overlaying spatial layers (e.g., all hotels and metro stations in current window or a given geo-region) to examine the relationships among them visually. However, these user interfaces are not expressive enough and restrict users from specifying their intent in a flexible manner.

To the best of our knowledge, the problem of formulating arbitrary queries that span multiple domains comprising of structured and/or unstructured sources has not been addressed. We believe that the Query-By-Keywords (QBK) approach proposed in this dissertation (Chapter 2) could be the first step towards solving this problem, which may in turn, assist the next generation of retrieval frameworks.

6.1.2 Mapping Intent into Queries

Given an intent specified by the user, the next challenge is to transform this intent into an appropriate query format that can be represented using a variant of relational algebra (or similar established mechanisms). Since the queries (as seen in *Example-1* in Chapter 1) are complex and involve a myriad set of conditions, it is obvious that applying the existing formalisms of relational algebra may not be sufficient.

Over the past decade, several querying languages that extend the basics of relational algebra and allow access to *structured data* (SQL, OOQL [9], Whirl [85]), *semi-structured data* (SemQL [10], CARIN [89], StruQL [90]) and *vague (or unstructured) data* (VAGUE [91]) have been designed. These languages have, *with limited success*, incorporated imprecise user queries posed on a single-domain (or fixed set of multiple domains). Additionally, several frameworks have deployed customized models that translate the user query to a query format supported by the internal global schema (that provides an interface to the underlying sources). Briefly, Havasu's *QPIAD* [92] maps *imprecise* user queries to a more generic query using a combination of data-mining techniques. Similarly, Ariadne [83] interprets the user-specified conditions as a sequence of *LOOM statements* that are combined to generate a single query. MetaQuerier's *form assistant* [57] consists of built-in type handlers that aids the query translation process with moderate human efforts.

However, existing mechanisms will prove to be insufficient to represent complex intent spanning several domains. Hence, as alluded to by the Query-By-Keywords approach, it becomes necessary to use domain-related taxonomies/ontologies and source-related semantics to disambiguate as well as generate multiple potential queries from the user intent. A feedback and learning mechanism may be appropriate to learn user intent from the combinations of concepts provided based on user feedback. If multiple queries are generated (which is very much possible on account of the ambiguity of natural language and the volume of concepts involved in the domains of integration), an ordering mechanism may be useful to obtain valuable feedback from the user. Once the query is finalized, a canonical representation can be used to further transform the query into its components and elaboration.

6.1.3 Domain Discovery And Source Identification

As elucidated by *Example-1* (in Chapter 1), the Query-By-Keywords approach relies on the availability of meta-data in the form of ontologies and taxonomies that assist formulation of user queries pertaining to multiple concepts/entities across several domains. Gathering appropriate knowledge about the domains and the corresponding sources within these domains is thus, vital to the success of the QBK approach. In order to relate various parts of a user query to appropriate domains (or concepts), the meaning of *information* that is interchanged across the system has to be understood.

Over the past decade, several customized techniques have been adapted by different frameworks that focus on capturing such meta-data about concepts and sources that facilitate easy mapping of queries over the global schema and/or the underlying sources. Havasu's *attribute value hierarchies* [86], InfoMaster's *knowledge-base* [93], Information Manifold's *CARIN* [89], TSIMMIS's *OLE model* [84], Ariadne's *LIM* [83], and Tukwila's *data-source catalog* [94] are some of the important advances in formulation of a comprehensive source repository replete with adequate domain knowledge.

For instance, Havasu's *attribute-valued hierarchies* [86] maintain a classification of the attributes of the data sources over which the user queries are formed. Ariadne uses an independent *domain model* [83] for each application, that integrates the information from the underlying sources and provides a single terminology for querying. This model is represented using the LOOM knowledge representation system [95]. TSIMMIS adopts an *Object Exchange Model* (OEM) [84], a self-describing (tagged) object model, in which objects are identified by labels, types, values, and an optional identifier. Information Manifold's CARIN [89] proposes a method for representing local-source completeness and an algorithm for exploiting source information in query processing. This is an important feature for integration systems, since, in most scenarios, data sources may be incomplete for the domain they are covering. Furthermore, it suggests the use of *probabilistic reasoning* for the ordering of data sources that appear relevant to answer a given query. InfoMaster's *knowledge base* [93] is responsible for the storage of all the rules and constraints required to describe heterogeneous data sources and their relationships with each other. In Tukwila, the metadata obtained from several sources is stored in a single *data source catalog* [94], and holds different type of information about the data sources such as – *semantic description* of the contents of the data sources, *overlap information* about pairs of data sources, and *key statistics* about the data, such as the cost of accessing each source, the sizes of the relations in the sources, and selectivity information. Additionally, the use of *ontologies* for modeling implicit and hidden knowledge has been considered as a possible technique to overcome the problem of semantic heterogeneity by a number of frameworks such as KRAFT [96], SIMS [97], OntoBroker [98], and others.

The proliferation of data on the Internet has ensured that within each domain, there exist vast number of sources providing adequate yet similar information. For instance, portals such as Expedia (`www.expedia.com`) and Travelocity (`www.travelocity.com`) provide information for the domain of *Air Travel*. Similarly, sources such as Google Scholar (`www.scholar.google.com`) and Microsoft Academic Search (`academic.research.microsoft.com/`) generate adequate and similar results for the domain of *Publications* and *Literature*. Thus, the next logical challenge is to automate the current manual process of identifying appropriate sources associated with individual domains. Semantic discovery of sources, that involves a combination of - web crawling, interface extraction, source clustering, semantic matching and source classification, has been extensively researched by the *Semantic Web* community [99]. Currently, a significant and increasing amount of information obtained from the web is hidden behind the query interfaces of searchable databases. The potential of integrating data from such *hidden* data sources [100] is enormous. The MetaQuerier project [101] addresses the challenges for integrating these deep-web sources such as – discovering and integrating sources automatically, finding an appropriate mechanism for mapping independent user-queries to source-specific sub-queries, and developing mass collaboration techniques for the management, description and rating of such sources.

To conclude, an ideal archetype would be to design a *global taxonomy* (that models all the heterogeneous domains across which user queries might be posed), and a *domain taxonomy* (that models all the sources belonging to the domain and orders them based on distinct criteria specified by the integration system). The construction of such a multi-level ontology requires extensive efforts in the areas of – *domain knowledge aggregation*, *deep-web exploration*, and *statistics collection*. However, the earlier work on databases (use of equivalences and statistics in centralized databases, use of source schemas for obtaining a global schema) and recent work on information integration (as elaborated earlier) provide adequate reasons to believe that this can be extended to multi-domain queries and computations that include spatial and temporal constraints.

6.2 Ranking Relevant Information

Although there was no notion of ranking in traditional databases, it has existed in the context of information retrieval for quite some time. With the advent of the Web, ranking gained prominence due to the volume of information being searched/browsed. Currently, ranking has become indispensable and is used in document retrieval systems, recommender systems, Web search/browsing, and traditional databases as well.

The focus of this dissertation is the development of a personalized ranking model based on a holistic framework that combines user choices for select queries (for obtaining ranking functions and user similarity), query analysis (for determining query similarity), and importantly the above two to identify a relevant ranking function for a new query. Logically, ranking has to be based on: user, query, the database content, and their inter-relationship all of which forms the basis for our framework. To the best of our knowledge, *user*- and *query-dependent* ranking in the context of Web database queries have not been addressed in conjunction in current literature. Below, we relate our effort to earlier work in these areas.

6.2.1 Ranking In Recommendation Systems

Given the notion of *user*- and *query*-similarity, it appears that our proposal is similar to the techniques of **collaborative** [28] [29] [30] [31] and **content** filtering [32] [33] [34] used in recommendation systems. However, there are some important differences (between ranking tuples for database queries versus recommending items in a specific order) that distinguish our work. For instance, each cell in the *user-item* matrix of recommendation systems represents a single scalar value that indicates the rating/preference of a particular user towards a specific item. Similarly, in the context of recommendations for social tagging [102] [103] [104] [105], each cell in the corresponding *user-URL/item-tag* matrix indicates the presence or absence of a tag provided by a user for a given URL/item. In contrast, each cell in the *user-query* matrix (used for database ranking) contains an ordered set of tuples (represented by a ranking function). Further, although the rating/relevance given to each tuple (in the results of a given query) by a user can be considered to be similar to a rating given for an item in recommendation systems, if the same tuple occurs in the results of distinct queries, it may receive different ratings from the same user. This aspect of the same item receiving varied ratings by the same user in different contexts makes it a different problem, and hence, current recommender techniques cannot be applied for solving this problem.

Another important distinction that sets our work apart from recommendation systems is the notion of *similarity*. In content filtering, the similarity between items is established either using a domain expert, or user profiles [34], or by using a feature recognition algorithm [32] over the different features of an item (e.g., author and publisher of a book, director and actor in a movie, etc.). In contrast, since our framework requires establishing similarity between actual SQL queries (instead of simple keyword queries), the direct application of these techniques does not seem to be applicable.

To the best of our knowledge, the notion of similarity has not been explored and leveraged in the context of database queries (SQL). Although the *no answers* problem [67] is typically addressed by using several techniques proposed for query relaxation [106] [107] that rewrite queries, the notion of similarity is employed in a delimited context to generate at least a few acceptable results. In contrast, similarity in the form of the information retrieval models [24] is used for ordering tuples in the *many answers* problem [43]. In contrast, equivalence and subsumption of queries have been well-established [108]; however the focus of these works is still toward obtaining answers without focusing on the aspect of ranking these answers; a requirement that needs to be relaxed in the context of Web databases. Furthermore, since we assume that the same user may have different preferences for different queries, capturing this information via profiles will not yield the same level of personalization that is, in contrast, provided by our approach.

The notion of user similarity used in our framework is identical to the one adopted in collaborative filtering; however, the technique used for determining this similarity is different. In collaborative filtering, users are compared based on the ratings given to individual items (i.e., if two users have given a positive/negative rating for the same items, then the two users are similar). In the context of database ranking, we propose a rigorous definition of user similarity based on the similarity between their respective ranking functions, and hence ranked orders. Furthermore, this work extends user-personalization using context information based on user and query similarity instead of static profiles and data analysis.

6.2.2 Ranking In Database Systems

Although ranking query results for relational and Web databases has received significant attention over the past years, a holistic support for automated similarity-based *user-* and *query-dependent* ranking has not been addressed in this context. For instance, [42] [67] address the problem of *query-dependent* ranking by adapting the vector model from information retrieval, whereas [43] [44] do the same by adapting the probabilistic model. However, for a given query, these techniques provide the same ordering of tuples across all users.

Employing user personalizations by considering the context and profiles of users for *user-dependent* ranking in databases has been proposed in [35] and [36]. Similarly, the work proposed in [48] requires the user to specify an ordering across the database tuples, without posing any specific query, from which a global ordering is obtained for each user. A drawback in all these works is that they do not consider that the same user may have varied ranking preferences for different queries.

The closest form of *query-* and *user-dependent* ranking in relational databases involves manual specification [38] [39] [37] [40] [41] of the ranking function/preferences as part of SQL queries. However, this technique is unsuitable for Web users who are not proficient with query languages and ranking functions. In contrast, our framework provides an automated *query-* as well as *user-dependent* ranking solution without requiring users to possess knowledge about query languages, data models and ranking mechanisms.

6.2.3 Ranking In Information Retrieval

Ranking has been extensively investigated in the domain of information retrieval. The cosine-similarity metric [25] is very successful in practice, and we employ its variant [42] for establishing similarities between attribute-value pairs as well as query results in our framework. The problem of integrating information retrieval system and database systems have been attempted [109] [24] with a view to apply the ranking models (devised for the former) to the latter; however, the as illustrated in the studies conducted by [110] [111], the intrinsic differences between their underlying models is a major problem.

6.2.4 Inferring Functions Via Relevance Feedback

Inferring a ranking function by analyzing the user's interaction with the query results originates from the concepts of relevance feedback [71] [112] [113] [114] in the domain of document and image retrieval systems. However, the direct application of either explicit or implicit feedback mechanisms for inferring database ranking functions has several drawbacks, and to the best of our knowledge have not been addressed in literature. To address this challenge in a limited context, we proposed a preliminary probabilistic learning method for capturing attribute preferences for Web queries. Our approach is similar to the use of learning methods, such as linear regression and Bayesian classifier, for deriving ranked lists and has been studied extensively in Machine Learning and Image Processing [45].

Our results show that within the framework that we tested, our proposed model performed better than existing Bayesian or regression models. There exists a number of sophisticated learning models that can aid in inferring a ranking function, but comparing them has not been the major focus of this work.

6.2.5 Workloads For Ranking Frameworks

Similar to the workload established in this work, recommendation systems in the form of content filtering [32] [33] [34] as well as collaborative filtering [28] [29] [30] [31] employ a similar user-item rating matrix. A significant difference that separates our work is the information contained in the matrix. While each cell in the rating matrix of recommendation systems contains information in the form of a simple rating given by an user to an object (e.g., rating a particular movie), the information contained in each cell of our workload is more complex in terms of a preference to individual tuples within the results of a query. While it is significantly easier, in terms of time and effort, to obtain a rating for an item, obtaining a ranking function from a user incurs considerable time and effort (since the user needs to browse through the query results and then make individual choices towards the relevance/non-relevance of the tuples).

Furthermore, since the goal of recommendation systems is to recommend items to user based on past data, the more ratings obtained by this matrix, the better it serves these systems. Although this is applicable even in the case of our workload for ranking in Web databases, owing to the cost in obtaining individual functions, the number of functions that can be obtained will be much lesser than the ratings obtained for a recommender system. Consequently, deciding the exact pairs for whom the functions need to obtained (for ensuring a good quality of subsequent ranking) is much more vital than determining the objects for which ratings can be obtained.

To conclude, this dissertation presents a unique framework that blends a novel similarity model with a sophisticated workload (obtained using a heuristic approach) to assist ranking in the context of Web databases; and to the best of our knowledge, has not been addressed in literature.

CHAPTER 7

CONCLUSIONS AND FUTURE DIRECTIONS

The principal contribution of this work is to provide a new insight to the paradigms of – *intent specification* (for focused information retrieval on the Web) and *ranking* (for providing a user- and query-specific ordering of results). In particular, we have made the following original contributions.

7.1 Contributions

An Approach For Query Formulation On The Web – We presented Query-By-Keywords (or QBK), a preliminary approach for query specification over the unstructured Web. This approach translates an user intent, expressed as a collection of keywords, to a structured query that can be processed over the disparate Web sources. We demonstrated the feasibility of this approach when supported by an adequate Knowledge Base, comprising of a Knowledge Repository of semantic information and a Workload Repository made up of statistical data. We detailed the steps involved in the processes of keyword resolution, ranking alternative query intents, generating appropriate query skeletons, and finally, transforming these skeletons into full-fledged queries via meaningful user interaction.

A Similarity-based Ranking Model For Web Databases – Given the limitations of current frameworks, in terms of providing a personalized model of ranking, we advanced a unique Similarity-based ranking framework for user- and query-dependent ranking of query results in the context of Web databases. Specifically we presented two comprehensive models – *Query Similarity* and *User Similarity*, that were merged to form a holistic ranking framework for Web databases. The model of Query Similarity is a novel advancement to the notion of establishing similarity between any pair of arbitrary SQL queries posed (in the form of SPJ semantics) on a database. Likewise, the component of User Similarity is an innovative amalgamation of static information (in the form of user profiles) and dynamic user behavior (in the form of their browsing choices and ranking preferences over query results). The applicability of this ranking model, in terms of the quality of ranked results as well as the efficiency for real-time situations, was established via an extensive experimental evaluation with the assistance of real users from Amazon Mechanical Turk over two distinct Web databases provided by Google Base.

An Algorithmic Approach For Workload Design – The similarity-based ranking framework, presented in this work, relies on the availability of a workload of ranking functions, collected from several users asking various queries on the Web database. Thus, given that the quality of the final ranking depends on such a workload, we argued that this workload needs to be established using an algorithmic design, instead of a random process. Toward that, we presented a *Workload Filling* technique for determining a "good" set of users and queries to represent a workload for assisting *user*- and *query*-dependent ranking on Web databases. In order to quantify the notion of a "good" workload, we advanced a novel metric of *Workload Goodness*. We further demonstrated that finding an optimal workload is intractable in practice; and to overcome this challenge, we proposed a suite of heuristic algorithms. Further, we showed the applicability of our approach in determining the requisite workload in a *static* as well as *dynamic* environment. We analytically explained the effectiveness of our proposal and validated it experimentally over the two Web databases of Google Base coupled with real users from Amazon Mechanical Turk.

A Learning Model For Inferring Ranking Functions – The workload employed by the similarity-based ranking framework, in turn represents a collection of distinct ranking functions for several user-query pairs. Each ranking function depicts the preferences of a specific user towards the results of a distinct query. The ranking model eventually avails of these functions, for performing ranking at the time of query; hence, appropriately capturing these user preferences over query results and translating them into appropriate functions becomes a pertinent challenge. In this dissertation, we put forward novel technique for obtaining ranking functions based on user preferences. Unlike relational databases, the nature of Web database applications allow users to browse and select the results that match their preferences (through an interaction with the Web pages containing the result records). Hence, although an user's explicit ranking preferences over the results of a query are not available, our proposal avails of the results *implicitly* chosen by an user that, in turn, indicate his/her ranking preferences. In order to translate these choices into formal mathematical ranking functions, we advanced a novel technique, that relies on the principles of Machine Learning, called *Probabilistic Data Distribution Difference*. We compared our proposal with existing learning mechanisms such as Naive Bayes and Linear Regression, and showed its effectiveness with respect to these models via an experimental evaluation.

7.2 Future Work

In this dissertation, we presented a framework for the QBK approach. Although we outlined the different components of its resulting framework as well as identified the various types of information to be associated with the Knowledge Base, an important task is to test the practicality of this approach. Given that the approach adheres itself to query formulation over the Web, determining its applicability in the context of Web users specifying intents over multiple disparate domains on the Web becomes mandatory. Correspondingly, investigating (semi-automated) techniques for constructing the Knowledge Base, that supports this approach, is necessary. Furthermore, since the given intent is eventually transformed into a complete formal query, extending the semantics of a query language (such as SQL) to include constructs for domains, Web sources, temporal/spatial conditions is vital.

Finally, checking the applicability of this approach for focused retrievals in the domains (beyond the Web) such as biology, bioinformatics, and other sciences and related fields would be interesting.

The primary contribution of this thesis is a holistic framework for user- and query-dependent ranking over Web databases. Although we covered most of the bases in this problem i.e., a comprehensive similarity model, an appropriately designed workload, and a learning model for inferring functions; certain challenges need to be further explored.

In the ranking framework, the current model of *query similarity*, although capable of handling most SPJ queries, needs to be further extended to cover all aspects for SQL, namely – queries involving GROUP-BY conditions, HAVING clauses, and Aggregate operators. In addition, application of the query similarity model to different applications such as query processing and optimization, analysis of query logs and keyword-based searching in databases needs to be investigated. Incorporating these models as part of commercial databases like Oracle, MY-SQL, and others, to support recommendation of similar queries would be worth exploring into. Similarly, analyzing different techniques so as to adopt the most suitable one for combining the static and dynamic components of the *user similarity* would be meaningful.

In the context of workload design, determining an appropriate size in terms of the user-query pairs needs further attention. Although pragmatic in nature, we believe a good estimate of the workload size can further enhance the ranking model. Likewise, Web databases who plan on using this framework need to analyze different mechanisms (e.g., surveys, opinion polls, etc.) for obtaining the actual preferences from users. Furthermore, our current representation of a ranking function is a linear weighted-sum model; however, further analysis that yields the exact nature of a function that maps closely to user preferences would be worthwhile. Finally, determining the applicability of the ranking framework to other repositories such as the surface Web, document collections, traditional databases and recommender systems would be beneficial.

To conclude, we presented novel solutions to the two most important problems in the context of the Web as known today i.e., intent specification, and relevant result retrieval. As the scale and spread of the Web continues to rise, we believe that this dissertation could act as a stepping stone for spawning new threads of research in these areas, which in turn, would lead to the development of the next generation of sophisticated personalized frameworks for information retrieval.

REFERENCES

[1] G. Salton, J. Allan, and C. Buckley, “Approaches to Passage Retrieval in Full Text Information Systems,” in *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR)*, 1993, pp. 49–58.

[2] S. Miike, E. Itoh, K. Ono, and K. Sumita, “A Full-Text Retrieval System with a Dynamic Abstract Generation Function,” in *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR)*, 1994, pp. 152–161.

[3] M. Narita and Y. Ogawa, “The Use of Phrases from Query Texts in Information Retrieval,” in *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR)*, 2000, pp. 318–320.

[4] M. Shokouhi and J. Zobel, “Federated Text Retrieval from Uncooperative Overlapped Collections,” in *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR)*, 2007, pp. 495–502.

[5] B. Katz, J. J. Lin, and D. Quan, “Natural Language Annotations for the Semantic Web,” in *Proceedings of the Conference on Cooperative Information Systems (CoopIS)*, 2002, pp. 1317–1331.

[6] S. Agrawal, S. Chaudhuri, and G. Das, “DBXplorer: A System for Keyword-Based Search over Relational Databases,” in *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2002, pp. 5–16.

[7] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, and S. Sudarshan, “BANKS: Browsing and Keyword Searching in Relational Databases,” in *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2002, pp. 1083–1086.

[8] J. Melton and A. R. Simon, *Understanding the New SQL: A Complete Guide*. Morgan Kaufmann Publishers, 1993.

[9] L. Liu, C. Pu, and Y. Lee, "An adaptive approach to query mediation across heterogeneous information sources," in *Proceedings of the Conference on Cooperative Information Systems (CoopIS)*, 1996, pp. 144–156.

[10] J.-O. Lee and D.-K. Baik, "SemQL: A Semantic Query Language for Multidatabase Systems," in *Proceedings of the Conference on Information and Knowledge Management (CIKM)*, 1999, pp. 259–266.

[11] M. K. Bergman, "The Deep Web: Surfacing Hidden Value," *Proceedings of the Journal of Electronic Publishing*, vol. 7, no. 1, 2001.

[12] K. C.-C. Chang, B. He, C. Li, M. Patil, and Z. Zhang, "Structured Databases on the Web: Observations and Implications," *Proceedings of the ACM Special Interest Group On Management Of Data (SIGMOD)*, vol. 33, no. 3, pp. 61–70, 2004.

[13] M. M. Zloof, "Query-by-Example: A Database Language," *IBM Systems Journal*, vol. 16, no. 4, pp. 324–343, 1977.

[14] ——, "Design Aspects of the Query-by-Example Data Base Management Language," in *Proceedings of the International Conference on Data and Knowledge Bases (JCDKB)*, 1978, pp. 29–55.

[15] Z. Nie, S. Kambhampati, and T. Hernandez, "BibFinder/StatMiner: Effectively Mining and Using Coverage and Overlap Statistics in Data Integration," in *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2003, pp. 1097–1100.

[16] W. W. Cohen, "A Demonstration of WHIRL," in *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR)*, 1999.

[17] B. He, Z. Zhang, and K. C.-C. Chang, “Towards Building a MetaQuerier: Extracting and Matching Web Query Interfaces,” in *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2005, pp. 1098–1099.

[18] C. A. Knoblock, S. Minton, J. L. Ambite, N. Ashish, I. Muslea, A. Philpot, and S. Tejada, “The Ariadne Approach to Web-Based Information Integration,” *Proceedings of the International Journal on Cooperative Information Systems*, vol. 10, no. 1-2, pp. 145–169, 2001.

[19] D. Braga, S. Ceri, F. Daniel, and D. Martinenghi, “Optimization of Multi-domain Queries on the Web,” *Proceedings of the International Conference on Very Large Databases (VLDB)*, vol. 1, no. 1, pp. 562–573, 2008.

[20] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives, “DBpedia: A Nucleus for a Web of Open Data,” in *Proceedings of the International Semantic Web Conference*, 2007, pp. 722–735.

[21] A. Telang, S. Chakravarthy, and C. Li, “Query-By-Keywords (QBK): Query Formulation Using Semantics and Feedback,” in *Proceedings of the International Conference on Conceptual Modeling (ER)*, 2009, pp. 191–204.

[22] A. Telang, S. Chakravarthy, and Li, “Querying for Information Integration: How To Go From An Imprecise Intent To A Precise Query?” in *Proceedings of the International Conference on Management of Data (COMAD)*, 2008, pp. 245–248.

[23] D. L. Lee, H. Chuang, and K. E. Seamons, “Document Ranking and the Vector-Space Model,” *Proceedings of the IEEE International Journal on Software Systems*, vol. 14, no. 2, pp. 67–75, 1997.

[24] N. Fuhr, “A Probabilistic Relational Model for the Integration of IR and Databases,” in *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR)*, 1993, pp. 309–317.

[25] Ricardo Baeza-Yates and Berthier Ribeiro-Neto, *Modern Information Retrieval*. ACM Press, 1999.

[26] S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," in *Proceedings of the International World-Wide Web Conference (WWW)*, vol. 30, no. 1-7, 1998, pp. 107–117.

[27] W. W. Cohen, "Recognizing Structure in Web Pages using Similarity Queries," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 1999, pp. 59–66.

[28] T. Hofmann, "Collaborative Filtering via Gaussian Probabilistic Latent Semantic Analysis," in *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR)*, 2003, pp. 259–266.

[29] J. Basilico and T. Hofmann, "A Joint Framework for Collaborative and Content Filtering," in *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR)*, 2004, pp. 550–551.

[30] D. Billsus and M. J. Pazzani, "Learning Collaborative Information Filters," in *Proceedings of the IEEE International Conference on Machine Learning (ICML)*, 1998, pp. 46–54.

[31] P. W. Foltz and S. T. Dumais, "Personalized Information Delivery: An Analysis of Information Filtering Methods," *Proceedings of the ACM Communications*, vol. 35, no. 12, pp. 51–60, 1992.

[32] M. Balabanovic and Y. Shoham, "Content-Based Collaborative Recommendation," *Proceedings of the ACM Communications*, vol. 40, no. 3, pp. 66–72, 1997.

[33] C. Basu, H. Hirsh, and W. W. Cohen, "Recommendation as Classification: Using Social and Content-Based Information in Recommendation," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 1998, pp. 714–720.

[34] S. Gauch, M. Speretta, A. Chandramouli, and A. Micarelli, "User Profiles for Personalized Information Access," in *Proceedings of the The Adaptive Web*, 2007, pp. 54–89.

[35] G. Koutrika, "Database Query Personalization," in *Proceedings of the International Conference on Extending Database Technologies (EDBT)*, 2005, pp. 147–152.

[36] G. Koutrika and Y. E. Ioannidis, "Constrained Optimalities in Query Personalization," in *Proceedings of the ACM Special Interest Group on Management of Data (SIGMOD)*, 2005, pp. 73–84.

[37] M. Ortega-Binderberger, K. Chakrabarti, and S. Mehrotra, "An Approach to Integrating Query Refinement in SQL," in *Proceedings of the International Conference on Extending Database Technologies (EDBT)*, 2002, pp. 15–33.

[38] K. Werner, "Foundations of Preferences in Database Systems," in *Proceedings of the International Conference on Very Large Databases (VLDB)*. VLDB Endowment, 2002, pp. 311–322.

[39] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song, "RankSQL: Query Algebra and Optimization for Relational Top-k Queries," in *Proceedings of the ACM Special Interest Group on Management of Data (SIGMOD)*, 2005, pp. 131–142.

[40] A. Marian, N. Bruno, and L. Gravano, "Evaluating Top-k Queries over Web-accessible Databases," *Proceedings of the ACM Transactions of Database Systems (TODS)*, vol. 29, no. 2, pp. 319–362, 2004.

[41] H. Yu, S.-w. Hwang, and K. C.-C. Chang, "Enabling Soft Queries for Data Retrieval," *Proceedings of the International Journal on Information Systems*, vol. 32, no. 4, pp. 560–574, 2007.

[42] S. Agrawal, S. Chaudhari, G. Das, and A. Gionis, "Automated Ranking of Database Query Results," in *Proceedings of the Conference on Innovations in Database Research (CIDR)*, 2003.

[43] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum, "Probabilistic Information Retrieval Approach for Ranking of Database Query Results," *Proceedings of the ACM Transactions of Database Systems (TODS)*, vol. 31, no. 3, pp. 1134–1168, 2006.

[44] W. Su, J. Wang, Q. Huang, and F. Lochovsky, "Query Result Ranking over E-commerce Web Databases," in *Proceedings of the Conference on Information and Knowledge Management (CIKM)*, 2006, pp. 575–584.

[45] H. Yu, Y. Kim, and S. won Hwang, "RV-SVM: An Efficient Method for Learning Ranking SVM," in *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2009, pp. 426–438.

[46] G. Koutrika and Y. E. Ioannidis, "Personalization of Queries in Database Systems," in *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2004, pp. 597–608.

[47] R. Agrawal, R. Rantzau, and E. Terzi, "Context-Sensitive Ranking," in *Proceedings of the ACM Special Interest Group on Management of Data (SIGMOD)*. New York, NY, USA: ACM, 2006, pp. 383–394.

[48] Seung-Won Hwang, "Supporting Ranking For Data Retrieval," Ph.D. dissertation, University of Illinois, Urbana Champaign, 2005.

[49] A. Telang, C. Li, and S. Chakravarthy, "One Size Does Not Fit All: Towards User- and Query-Dependent Ranking For Web Databases," *Proceedings of the IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2011, (To Appear).

[50] ——, "One Size Does Not Fit All: Towards User- and Query-Dependent Ranking For Web Databases," University of Texas at Arlington, Tech. Rep. 6, 2009.

[51] A. Telang, S. Chakravarthy, and C. Li, "Establishing SQL Query Similarity For Ranking and Other Applications," in *Proceedings of the Conference on Information and Knowledge Management (CIKM)*, 2011, (Under Review).

[52] ——, “Fill-In-The-Functions: Toward Establishing A Workload For Ranking in Web Databases,” *Proceedings of the IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2011, (Under Review).

[53] ——, “Fill-In-The-Functions: Towards Establishing A Workload For Ranking in Web Databases,” University of Texas at Arlington, Tech. Rep. 3, 2011.

[54] A. Telang, S. Chakravarthy, and Y. Huang, “Information Integration Across Heterogeneous Sources: Where Do We Stand and How to Proceed?” in *Proceedings of the International Conference on Management of Data (COMAD)*, 2008, pp. 186–197.

[55] A. Telang, R. Mishra, and S. Chakravarthy, “Ranking Issues For Information Integration,” in *Proceedings of the IEEE International Conference on Data Engineering (ICDE) Workshop (DBRank)*, 257-260, Ed., 2007.

[56] A. Telang and S. Chakravarthy, “Information Integration across Heterogeneous Domains: Current Scenario, Challenges and the InfoMosaic Approach,” University of Texas at Arlington, Tech. Rep., 2007.

[57] Z. Zhang, B. He, and K. C.-C. Chang, “Light-weight Domain-based Form Assistant: Querying Web Databases On the Fly,” in *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2005, pp. 197–208.

[58] E. Chu, A. Baid, X. Chai, A. Doan, and J. Naughton, “Combining Keyword Search and Forms For Ad Hoc Querying Of Databases,” in *Proceedings of the ACM Special Interest Group on Management of Data (SIGMOD)*, 2009, pp. 349–360.

[59] G. A. Miller, “WordNet: A Lexical Database for English,” *Proceedings of the ACM Communications*, vol. 28, no. 11, pp. 39–41, 1995.

[60] J. F. Allen, “Maintaining Knowledge About Temporal Intervals,” *Proceedings of the ACM Communications*, vol. 26, no. 11, pp. 832–843, 1983.

[61] F. Fonseca, M. Egenhofer, P. Agouris, and G. Camara, “Using Ontologies for Integrated Geographic Information Systems,” *Proceedings of the ACM Transactions in Geographic Information Systems*, vol. 3, 2002.

[62] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach, “Scalable Semantic Web Data Management Using Vertical Partitioning,” in *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2007, pp. 411–422.

[63] R. Waldinger, D. E. Appelt, J. Fry, D. J. Israel, P. Jarvis, D. Martin, S. Riehemann, M. E. Stickel, M. Tyson, J. Hobbs, and J. L. Dungan, “Deductive Question Answering from Multiple Resources,” in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2004.

[64] H. N. Gabow and E. W. Myers, “Finding All Spanning Trees of Directed and Undirected Graphs,” *Proceedings of the Society For Industrial and Applied Mathematics (SIAM)*, vol. 7, no. 3, pp. 280–287, 1978.

[65] A. Agresti, *Building and Applying Logistic Regression Models.* Wiley, August 2006, no. 2, ch. 5.

[66] J. Adlrich, “R.A. Fisher And The Making Of Maximum Likelihood,” *Proceedings of the Journal of Statistical Sciences*, vol. 12, no. 3, pp. 1912–1922, August 1997.

[67] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum, “Probabilistic Ranking of Database Query Results,” in *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2004, pp. 888–899.

[68] J. L. Myers and A. D. Well, *Research Design and Statistical Analysis.* Lawrence Erlbaum Associates, 2003.

[69] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, “Rank Aggregation Methods for the Web,” in *Proceedings of the International conference on World Wide Web (WWW).* New York, NY, USA: ACM, 2001, pp. 613–622.

[70] T. Kanungo and D. Mount, “An Efficient K-Means Clustering Algorithm: Analysis and Implementation,” *Proceedings of the IEEE Transactions of Pattern Analysis in Machine Intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[71] B. He, “Relevance Feedback,” in *Proceedings of the Encyclopedia of Database Systems*, 2009, pp. 2378–2379.

[72] F. Nielsen and S. Boltz, “The Burbea-Rao And Bhattacharyya Centroids,” *Proceedings of the Computer Research Repository (CoRR)*, 2010.

[73] W. J. Reed, “The Pareto, Zipf and Other Power Laws,” *Proceedings of the Economic Letters*, vol. 74, no. 1, pp. 15–19, December 2001.

[74] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, “Methods and Metrics for Cold-Start Recommendations,” in *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR)*, 2002, pp. 253–260.

[75] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan, “Time Bounds for Selection,” *Proceedings of the Journal of Computer and System Sciences*, vol. 7, pp. 448–461, August 1973.

[76] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison-Wesley, 2005.

[77] J. L. Rodgers and A. Nicewander, *Thirteen Ways to Look at the Correlation Coefficient*, ser. The American Statistician. Mendeley, 1988, vol. 42.

[78] A. Agresti, *Categorical Data Analysis*, ser. Wiley Series in Probability and Statistics. Wiley, 2002.

[79] MathWorks Inc., “Weighted Correlation Matrix,” http://www.mathworks.com/matlabcentral.

[80] M. Petropoulos, A. Deutsch, and Y. Papakonstantinou, “CLIDE: Interactive Query Formulation for Service Oriented Architectures,” in *Proceedings of the ACM Special Interest Group on Management of Data (SIGMOD)*, 2007, pp. 1119–1121.

[81] S. M. zu Eissen and B. Stein, “Analysis of Clustering Algorithms for Web-Based Search,” in *Proceedings of the International Conference on Practical Aspects of Knowledge Management (PAKM)*, 2002, pp. 168–178.

[82] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu, “Web-Scale Data Integration: You can afford to Pay as You Go,” in *Proceedings of the Conference on Innovations in Database Research (CIDR)*, 2007, pp. 342–350.

[83] C. A. Knoblock, “Planning, Executing, Sensing, and Replanning for Information Gathering,” in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995, pp. 1686–1693.

[84] S. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom, “The TSIMMIS Project: Integration of Heterogeneous Information Sources,” in *Prcoeedings of the ACM Transactions on Computer Vision and Applications (IPSJ)*, 1994, pp. 7–18.

[85] W. W. Cohen, “Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity,” in *Proceedings of the ACM Special Interest Group on Management of Data (SIGMOD)*, 1998, pp. 201–212.

[86] S. Kambhampati, U. Nambiar, Z. Nie, and S. Vaddi, “Havasu: A Multi-Objective, Adaptive Query Processing Framework for Web Data Integration,” Arizona State University, Tech. Rep., 2002.

[87] V. Subrahmanian, “A HEterougeneous Reasoning and MEdiator System,” http://www.cs.umd.edu/projects/hermes/.

[88] M. Michalowski and C. A. Knoblock, “A Constraint Satisfaction Approach to Geospatial Reasoning,” in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2005, pp. 423–429.

[89] A. Y. Levy, “Information Manifold Approach to Data Integration,” *Proceedings of the IEEE Journal on Intelligent Systems*, pp. 1312–1316, 1998.

[90] D. Florescu, A. Levy, and A. Mendelzon, “Database Techniques for the World-Wide Web: A Survey,” *Proceedings of the ACM Special Interest Group On Management Of Data (SIGMOD)*, vol. 27, no. 3, pp. 59–74, 1998.

[91] A. Motro, “VAGUE: A User Interface to Relational Databases that Permits Vague Queries,” *Proceedings of the ACM Transactions of Information Systems*, vol. 6, no. 3, pp. 187–214, 1988.

[92] J. Fan, H. Khatri, Y. Chen, and S. Kambhampati, “QPIAD: Query processing over Incomplete Autonomous Databases,” Arizona State University, Tech. Rep., 2006.

[93] O. M. Duschka and M. R. Genesereth, “Infomaster: An Information Integration Tool,” in *Proceedings of the International Conference on Intelligent Information Integration*, 1997.

[94] Z. G. Ives, D. Florescu, M. Friedman, A. Levy, and D. S. Weld, “Adaptive Query Processing for Internet Applications,” in *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 1999, pp. 19–26.

[95] R. M. MacGregor, “Inside the LOOM Description Classifier,” *Proceedings of the ACM Special Interest Group On Artificial Intelligence (SIGART)*, vol. 2, no. 3, pp. 88–92, 1991.

[96] P. M. D. Gray, A. D. Preece, N. J. Fiddian, W. A. Gray, and et. al., “KRAFT: Knowledge Fusion from Distributed Databases and Knowledge Bases,” in *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA)*, 1997, pp. 682–691.

[97] C.-N. Hsu and C. A. Knoblock, “Reformulating Query Plans for Multidatabase Systems,” in *Proceedings of the Conference on Information and Knowledge Management (CIKM)*, 1993, pp. 423–432.

[98] S. Decker, M. Erdmann, D. Fensel, and R. Studer, “Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information,” in *Proceedings of the Database Semantics: Semantic Issues in Multimedia Systems*, 1999, pp. 351–369.

[99] H. Stuckenschmidt and H. Wache, “Context Modelling and Transformation for Semantic Interoperability,” *Proceedings of the Knowledge Representation Meets Databases (KRDB)*, 2000.

[100] B. He, M. Patil, K. C.-C. Chang, and Z. Zhang, “Accessing the Deep Web: A Survey,” University of Illinois at Urbana-Champaign, Tech. Rep., 2004.

[101] K. C.-C. Chang, B. He, and Z. Zhang, “Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web,” in *Proceedings of the Conference on Innovations in Database Research (CIDR)*, 2005, pp. 44–55.

[102] S. Amer-Yahia, A. Galland, J. Stoyanovich, and C. Yu, “From del.icio.us To x.qui.site: Recommendations In Social Tagging Sites,” in *Proceedings of the ACM Special Interest Group on Management of Data (SIGMOD)*, 2008, pp. 1323–1326.

[103] A. Penev and R. K. Wong, “Finding Similar Pages In A Social Tagging Repository,” in *Proceedings of the International conference on World Wide Web (WWW)*, 2008, pp. 1091–1092.

[104] K. Razikin, D. H.-L. Goh, E. K. C. Cheong, and Y. F. Ow, “The Efficacy of Tags in Social Tagging Systems,” in *Proceedings of the International Conference on Asian Digital Libraries (ICADL)*, 2007, pp. 506–507.

[105] T. C. Zhou, H. Ma, M. R. Lyu, and I. King, “UserRec: A User Recommendation Framework in Social Tagging Systems,” in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

[106] S. Shen, “Database Relaxation: An Approach to Query Processing in Incomplete Databases,” *Proceedings of the Journal on Information Processing and Management*, vol. 24, no. 2, pp. 151–159, 1988.

[107] J. Shan, D. Shen, T. Nie, Y. Kou, and G. Yu, "An Effective and High-quality Query Relaxation Solution on the Deep Web," in *Proceedings of the International Asia-Pacific Web Conference (APWEB)*, 2010, pp. 68–74.

[108] N. Chomsky, *The Minimalist Program.* MIT Press, 1995, vol. 28.

[109] N. Fuhr, "A Probabilistic Framework for Vague Queries and Imprecise Information in Databases," in *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1990, pp. 696–707.

[110] W. W. Cohen, "Providing database-like access to the web using queries based on textual similarity," in *SIGMOD Conference*, 1998, pp. 558–560.

[111] S. Amer-Yahia, P. Case, T. Rölleke, J. Shanmugasundaram, and G. Weikum, "Report on the DB/IR Panel At SIGMOD 2005," *Proceedings of the ACM Special Interest Group On Management Of Data (SIGMOD)*, vol. 34, pp. 71–74, 2005.

[112] D. Harper and C. V. Rissbergen, "An Evaluation of Feedback in Document Retrieval Using An Evaluation Of Feedback In Document Retrieval Using Co-occurence Data," in *Proceedings of the International Journal of Documentation*, vol. 34, no. 3, 1978, pp. 189–216.

[113] Y. Rui, T. S. Huang, and S. Mehrotra, "Content-Based Image Retrieval With Relevance Feedback In MARS," in *Proceedings of the IEEE International Conference on Image Processing*, 1997, pp. 815–818.

[114] L. Wu, C. Faloutsos, K. P. Sycara, and T. R. Payne, "FALCON: Feedback Adaptive Loop for Content-Based Retrieval," in *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2000, pp. 297–306.

BIOGRAPHICAL STATEMENT

Aditya Telang was born in Mumbai, India in 1981. He received his Bachelor of Engineering degree in Computer Science and Engineering from University of Mumbai, India in July 2003. He received his Masters of Science degree in Computer Science from the University of Buffalo, The State University of New York in February 2005. He received his Doctor of Philosophy in Computer Science and Engineering from The University of Texas at Arlington in August 2011.

He worked as a Software Developer for Talker Inc., New York in 2005 and has served as a Graduate Teaching Assistant, a Graduate Research Assistant, a Faculty Associate and an Assistant Instructor in the Department of Computer Science and Engineering at The University of Texas at Arlington from 2006 till 2011. He is a member of TBP, ACM and IEEE. In addition, he is the recipient of the Cyneta CSE Graduate Teaching Assistant Award in 2009 and the recipient of the STEM Doctoral Fellowship from 2007 till 2011.

Following the completion of his Ph.D., Aditya Telang will begin work as a Research Scientist at IBM India Research Labs in Bangalore, India.