# CORROBORATING INFORMATION FROM MULTIPLE SOURCES

**BY MINJI WU**

**A dissertation submitted to the**

**Graduate School—New Brunswick**

**Rutgers, The State University of New Jersey**

**in partial fulfillment of the requirements**

**for the degree of**

**Doctor of Philosophy**

**Graduate Program in Computer Science**

**Written under the direction of**

**Amélie Marian**

**and approved by**

**New Brunswick, New Jersey**

**October, 2016**

## ABSTRACT OF THE DISSERTATION

# Corroborating Information from Multiple Sources

**by Minji Wu**

**Dissertation Director: Amélie Marian**

Information on the Internet is abundant but often inaccurate. Given a query that has a unique answer (as opposed to a Web query against a search engine), different Web sources might provide multiple conflicting answers. As a result, users are left with the burden of validating the correctness of the answer from each source. In order to tackle this problem, corroboration techniques have been proposed in order to identify the correct answer given a set of candidate answers extracted from the sources. Corroboration is the technique that evaluates the quality of the answers by considering the trustworthiness of the sources from which the answers are extracted. Intuitively, an answer extracted from a trustworthy source is more likely to be the correct answer. In return, the more correct answers it reports, the more trustworthy a source is.

Unfortunately, several challenges arise before we can successfully apply a corroboration technique to find the correct answer to a query. First of all, the prime challenge is how to evaluate the trustworthiness of the sources and henceforth derive the quality of an answer based on the sources reporting it. Secondly, in a case where all the sources agree on a single candidate answer, how to validate the correctness of the answer. Third, in an application in which each source only provides a partial answer and the final answer is a combination of partial answers from multiple sources, how to evaluate the quality of answers and how to efficiently compute the correct answer. This thesis investigates several real world problems and proposes novel corroboration techniques that address each of the challenges presented above.

We first studied the problem of using corroboration for the task of question answering. With many web sources providing conflicting information on the Internet, users often have to rummage through a large number of different sites to both retrieve the information and ascertain the correctness of the retrieved information. While a naive approach that returns the most frequent answer can eliminate outlier answers such as typos, it fails to consider the fact that answers extracted from different pages are rarely equally important in answering the query. By ranking the answers based on the number, relevance and similarity of the web sources reporting them, as well as the prominence of the answers within the sources, our algorithm is able to efficiently identify accurate answers for most queries.

We investigated the problem of verifying the correctness of claims that are unanimously agreed upon among all sources. Intuitively, a claim supported by all the sources must be true, simply because there is no other source rebutting it. However, it might not be the case in real world scenarios since agreeing sources might be out-dated or due to copy/paste. In such a scenario, existing corroboration approaches tend to reach consensus quickly and conclude that all claims are true since there is little conflict among the sources. We studied this problem in a real world scenario (restaurant listings) and proposed a novel corroboration algorithm that evaluates the claims on a gradual basis. More specifically, our approach divides the claims into multiple sets and evaluates each set of claims using a different trust score from each source. Different from existing algorithms that assign a single trust score to each source, our approach computes a set of finer-grained trust scores for each source that is used to evaluate different set of claims.

In real world scenarios there often exist queries in which a single source is insufficient to provide a candidate answer. To answer these queries, users have to fetch and combine information from multiple sources and derive a potential final answer. Such cases are similar to the case of finding air ticket between two points without direct flight, and differs in that there is no centralized source (*e.g.,* Expedia.com) that provides the information of all connecting flights. The process of combining information from two sources is similar to the *join* operation in relational databases and therefore this problem can be viewed as a join query processing over multiple web-accessible databases. The main bottleneck of join query processing is tuple accessing of

web databases, which typically exhibit high and variable latency. In order to find the top-$k$ answers for a join query, a branch-and-bound algorithm has to be developed to avoid computing scores of all candidates exhaustively. Our method efficiently computes bounds for partial query results and determines a good order in which it accesses the tables so as to minimize wasted efforts in the computation of top-$k$ answers.

In summary, this thesis studies real world problems that involve information from multiple sources. We demonstrate that using information from a single source is often of low quality and in some cases insufficient. We discuss the challenges in each individual problem and present novel corroboration algorithms that efficiently compute scores for the answers by taking into consideration of the trustworthiness of the sources.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Recent advancement in the Web 2.0 technologies has fueled the explosion of online data. The huge amount of data has enabled applications in many domains such as business, education, etc. However, the data itself, be it from the Web, or from a database, or from a set of documents, is never certain. By certain, we mean that the correctness of the data cannot be blindly trusted. The is due to that data can be erroneous, misleading, biased and easily plagiarized. For instance, news sites sometimes post conflicting information pertaining to a certain event. Even the data from the most authoritative sources has the risk of being outdated. Therefore in order to assure certainty, data needs to be vetted before it can be used where data quality is of importance. Consider the following user example.

**Example 1:** *Consider a tourist named Joe wants to find a restaurant for dinner at Midtown Manhattan. After consulting with restaurant listing apps on his smartphone, he chose a restaurant named "Danny's Grand Sea Palace" located at '346 West 46th St, New York'. Unfortunately, after walking to the exact address, Joe finds out that the restaurant has already been closed (for good). Although the restaurant is listed at two reputable sources, Citysearch (http://newyork.citysearch.com) and Yellowpages (http://www.yp.com), the reality is that it has already gone out of business, and consequently results in a bad user experience for Joe.*

The quality of information is undoubtedly crucial to the success of web applications. To measure the data quality, an important observation is that multiple sources may provide information on the same topic that is of interest. For instance, for the topic "*Whether the restaurant named Danny's Grand Sea Palace is open*", 2 sources (*Citysearch* and *Yellowpages*) say *yes* and all other sources (*e.g., Yelp!, etc*) report unknown. Given the listing information from multiple sources, the task is to derive the correct information for the above topic.

The availability of data from multiple sources on the same topic has opened the gate of finding the correct information with the presence of low quality data. For the ease of discussion, let us consider the topics of interest as *queries*, to which each source provides a candidate answer. To find the correct answer from a set of candidate answers, the most straightforward way is through voting in which the answer reported by the most sources is selected as the correct answer. Unfortunately, voting based methods suffer from the fact that sources may be untrustworthy, and the correct answer could be outvoted by incorrect ones from a large number of unreliable sources.

To tackle this problem, corroboration techniques have been proposed that identify the correct answer by taking into consideration the trustworthiness of the sources. Intuitively, trustworthy sources are more likely to provide correct answers than unreliable sources. Consequently, answers from reputable sources should carry more weight in determining the right answer compared with the ones from untrustworthy sources. Corroboration techniques work by assigning a score to each source that represents its trustworthiness and use it to compute a score for each candidate answer that represents how likely it is the correct answer. The answer with the highest score is then selected as the correct answer.

This thesis studies the problem of finding the correct answer among a set of candidate answers provided from multiple sources. In particular, we propose novel corroboration algorithms that leverage the trustworthiness of the sources. In the remaining of the chapter, we first list the major challenges (Section 1.1) in designing corroboration algorithms and then give a brief introduction (Section 1.2) to the problems we discuss in the next few chapters. We summarize this chapter in Section 1.3.

## 1.1 Challenges

The two major metrics of a corroboration algorithm is correctness and efficiency. Naturally, the most important challenge is how to design the algorithm such that it can identify the correct answer for as many queries as possible. In the meantime, it is also important to identify the correct answer efficiently, especially in cases in which the accesses to remote sources are costly. We break these two challenges into smaller ones as below

- The most important challenge is how to estimate the trustworthiness of the sources, since an accurate estimation is the basis of computing the probability that an answer is the correct answer. Estimating the trust scores for the sources is difficult, since there is little to no prior knowledge about the sources or the candidate answers.

- Given a set of sources as well as their estimated trust scores, another key challenge is how to compute a score for the candidate answer that represents the likelihood it is the correct one. It is not clear how to combine the scores of the sources to derive the score of the answers.

- Given a set of sources that provide candidate answers to queries, how and in what order to select sources that is to be used for corroboration. Obviously retrieving all sources to process incurs prohibitive costs and could be impossible in cases where the number of sources is unbounded. It is crucial to design a way of selecting sources so that the correct answer can be found efficiently.

- How to design a branch-and-bound algorithm such that it can prune out unnecessary score computations and source accesses as much as possible. Users are interested in the information of the highest quality, our corroboration algorithm should be able to efficiently return the top-$k$ answers.

The problems we are going to discuss in the following chapters present several or all of the challenges listed above. We did not list some additional challenges such as extracting answers from sources, simply because we leveraged existing tools and do not consider it as one of our contributions.

## 1.2 Our contribution

This thesis studies 3 problems with uncertain data and tries to find the correct answer for the queries. These problems cover uncertain data from different types of sources (the Web and databases). As we show in subsequent chapters, each problem presents a set of challenges mentioned before. In particular, for each of the problems, our proposed corroboration method focuses on addressing one of the major challenges. In the following sections, we give a brief description for each of the 3 problems we studied, namely, question answering (Section 1.2.1), unanimous answer assertion (Section 1.2.2), and top-$k$ join query processing (Section 1.2.3).

As the most important challenge, how to estimate the trust scores of the sources is investigated in all 3 problems and is the core focus of the first 2 problems. In particular, we dealt with cases in which there exists prior knowledge that can be leveraged to infer the quality of the sources, as well as cases in which no useful information is available regarding the sources. In the question answering problem (Section 1.2.1), our corroboration algorithm is able to utilize the meta-data of the sources (*e.g.*, the ranking) to estimate how trustworthy a sources is. The problem is also unique in that there is an extra level of uncertainty due to the limitation of the information extraction techniques through which we used to extract candidate answers. In contrast, for the task of asserting unanimous answer (Section1.2.2), estimating the trust scores for the sources becomes more difficult as there is no knowledge about the sources to begin with. Therefore, we have to iteratively compute the scores for the sources and answers based on the answers each source provide until a convergence is reached. To make things more challenging, since most of the answers are unanimous, there is little conflict among the answers and consequently making the iterative method converge to perfect sources. Our proposed method circumvents the challenges by assigning multiple trust scores to the sources and making the decisions on the answers using one of the score values.

We turn our focus to the challenge of designing an efficient corroboration algorithm in the problem of top-$k$ join query processing (Section 1.2.3). For both the question answering and unanimous answer asserting, despite that the focus is to derive the correct answer, we measured the time complexity of our proposed methods and demonstrate that they are adequately efficient. In this problem, the queries are much more complicated in that in order to obtain a candidate answer, it requires to retrieve and combine information from multiple sources. Each source under this problem only hosts partial information of potential answers and the candidate answers are constructed by *joining* partial answers based on a join graph. Top-$k$ join query processing presents significant challenges on the efficiency of algorithms since it is not viable to retrieve information from all sources due to its prohibitive costs and therefore significant pruning is required.

### 1.2.1 Question answering

Question answering is the task of returning a direct answer to a user query instead of a set of Web documents through which the user has to browse in order to find the answer. Such questions could be ‘Who was the first astronaut that orbited the earth?’, or ‘What is the gas mileage of Fusion 2014?’. The huge amount of data on the Internet makes it a perfect knowledge base for question answering. However, it is possible to find different or even conflicting answers for a given query. In such a case, a frequency based approach may not work well, especially in a scenario where a large number of untrustworthy sources are present. As a result, question answering is an application that can directly benefit from corroboration systems. To tackle this problem, we propose a novel corroboration algorithm that correctly and efficiently returns the top-$k$ answers for each query (Chapter 4). The key observation is that answers extracted from different sources are rarely equally important in answering the query. By ranking the answers based on the number, relevance and similarity of the sources reporting them, as well as the prominence of the answers within the page, our algorithm is able to identify accurate answers for most queries [71] [72].

### 1.2.2 Asserting unanimous answers

It is intuitive that for a query with conflicting candidate answers, there exists uncertainty with respect to the correct answer, and hence the need for a corroboration system. However, corroboration also helps when all sources agree on one candidate answer, as illustrated in Example 1. In this scenario, our objective is to identify legitimate restaurants given the listing information from a set of reputable sources. A legitimate restaurant is defined as one that is up and running and in good business shape. Example 1 shows that even if a restaurant is listed at one or multiple reputable sources it might still be illegitimate. In other words, an answer agreed upon by all the sources does not necessarily indicate that the answer is the correct one. For such a task, state-of-the-art corroboration techniques do not work very well due to the fact that they rely on conflicting information to differentiate the trustworthiness of the sources. With little conflicts among the answers, existing methods usually result in a trust score close to 1 for all the sources and answers. To tackle this problem, we propose a novel corroboration algorithm that utilizes a multi-value trust score for each source. Our algorithm incrementally evaluates the restaurants by considering the information entropy of the unknown restaurants and significantly improve the accuracy of the corroboration results [73].

### 1.2.3 Processing top-$k$ join queries

Although we measured the time complexity of our corroboration algorithms for the above two problems, we are more interested in improving the answer accuracy. For the complex problem of top-$k$ join query processing, algorithm efficiency is of more importance and we turn our focus to designing an efficient corroboration algorithm. A join query is a query against a set of database tables as sources whose answer is composed of a tuple from each source, each of which is uncertain with a probability score. The join operation is based on a join graph that specifies the join relations among sources. Different from the first two problems in which a candidate answer can be independently retrieved from a source, a candidate answer for a join query requires expensive sorted and random accesses over the sources and therefore is much more costly. While the number of candidate answers for the first two problems is bounded by the number of sources, the number of candidate answers for a join query can go as much as the product of the number of tuples of all sources. To tackle this problem, we propose a novel branch-and-bound algorithm that maintains a set of partial candidate answers and incrementally determines a good order in which to retrieve tuples from the tables so as to minimize the efforts of data accesses.

## 1.3 Summary

We present the major challenges for a corroboration system and briefly discussed the scenarios this thesis covers. We demonstrate how to address the challenges in each of the scenarios in the following chapters. The rest of the thesis is organized as follows. We discuss related work in Chapter 2. Formal definitions and notations are introduced in Chapter 3. We present our corroboration algorithm for question answering in Chapter 4. We then discuss corroboration of affirmative statements in Chapter 5, followed by our work in corroboration over join queries in Chapter 6. Finally the thesis is concluded in Chapter 7.

# Chapter 2

# Related Work

We discuss related work in this chapter. In order to make the discussion organized, we present the relevant work in literature as follows. We first review the evolution of corroboration techniques (Section 2.1). In particular, we examine frequency based approaches (*e.g.*, voting) that is considered the primitive form of corroboration in Section 2.1.1, and more advanced corroboration methods that are proposed more recently in Section 2.1.2. We also discuss recent advancement in determining the data dependencies among sources (Section 2.2) which could greatly affect the corroboration results. Lastly, since corroboration techniques aim to return the top few best answers, we cover the topic of top-$k$ query processing that are extensively studied in the research community.

## 2.1 Corroboration techniques

We consider corroboration a technique that evaluates the quality of answers by considering the trustworthiness of the sources from which the answers are extracted. A frequency based approach, by definition, selects the most frequent answer as the correct answer. Although it does not explicitly take into consideration the quality of the sources, we consider it a special case of corroboration methods that considers all sources are equally trustworthy. In the following discussion, we first present frequency based approaches before jumping to approaches that differentiate the trustworthiness of the sources.

### 2.1.1 Frequency based approaches

Early works have considered the frequency of an answer as a measure of answer quality [22, 43, 23, 2]. The Mulder system, proposed in [43], uses frequency of answers to increase answer scores. This approach is similar to our page-frequency approach we examined in Chapter

4.5.2. The models proposed in [22, 23, 2] consider answer frequency at the extraction-rule level, *i.e.*, the score of an answer is increased if it is extracted from several high-quality rules or query rewrites, but not at the source level. In other words, an answer repeatedly extracted from an unreliable source using multiple rules receives a high score despite the fact that it is from an untrustworthy source. [22] proposes a probabilistic model to estimate the impact of repeated extraction from the web. The AskMSR system [2, 23] uses several query rewrites and n-gram extraction and considers answer redundancy in its answer scoring strategy. However, the redundancy is based on the number of answers returned by different query rewrites, and does not consider the quality of the sources reporting the answer.

[74] proposes a novel approach containing a set of features of answer context to estimate the answer confidence extracted from documents. However, when applied to a large corpora (*e.g.*, the Web), their approach simplifies to a frequency based approach. [80] proposes a novel approach that assigns scores to answers by comparing the query and the snippets from which the answers are extracted. In particular, for each answer, the snippets from which the answer is extracted are clustered and a bag-of-words feature vector is constructed for the answer. The answer score is then computed using the feature vector of the cluster and the query. However, their approach considers all the source snippets equally helpful and could be ineffective when a large number of low-quality sources are present.

There are also work that has focused on identifying entities from large data collections and answering queries on these entities. In particular, the WISDM system [12] focuses on entity search and proposes a probabilistic framework to rank entities extracted from web pages by considering the context around entities. The NAGA system [38] provides semantic knowledge to improve web search engine results. Neither systems operate on live web data, but on indexes built over web-extracted data. Our corroboration system for question answering bears some similarity with the problem of *entity finding*, which has been studied since the introduction of TREC entity track in 2009. The task of entity finding is, given a query containing an input entity with its name and homepage, to find related entities that are of a target type. Fang et al [27] proposed a hierarchical model for entity finding by considering the similarity between the query keyword and the document as well as the passage, from which entity candidates are extracted.

### 2.1.2 Advanced corroboration

To the best of our knowledge, our techniques in [71, 72] are the first to combine various measures of the trustworthiness of web sources, as well as the quality of the answers within the sources, and the frequency of these answers within the search engine query result. Using corroboration as a measure of answer quality has recently been suggested in non-web scenarios [41] where corroborating information is used to identify good answers across multiple databases in the presence of low quality data. In [5], the authors studied the problem of extracting and ranking numerical quantity answers from snippets. Their algorithm learns to score and rank quantity intervals by combining snippet quantity and text information. However, their snippet scoring is based on the *tf-idf* score as well as lexical proximity features and does not necessarily represent the trustworthiness of the sources.

More recently, a family of iterative corroboration techniques [79, 40, 28, 49, 54, 81, 29, 55] have also been proposed in finding the correct value among a set of conflicting values for an object (see a survey in [46]). An iterative corroboration technique usually assumes no prior knowledge regarding either the sources or the answers. Starting with a default score for each source and each answer, such approaches compute scores for the sources based on their votes on the answers. In return, the scores for the answers are calculated using the computed scores for the sources. The scores for the sources and the answers are then iteratively computed until a convergence is reached. An early example is in [40], in which the author proposes a link-based approach that iteratively computes a hub and authority score that are the sums of one another.

Yin et al. [79] proposed a novel iterative algorithm called TRUTHFINDER that uses Bayesian analysis and finds true facts among conflicting information. In particular, the answer score is calculated based on the quality of the providers. In addition, TRUTHFINDER takes into consideration the similarity among answers and boost the scores of answers if similar answers exist. Galland et al. in [28] proposed a suite of algorithms (namely, COSINE, TWOESTIMATE, THREEESTIMATE) that iteratively estimates the probability that an answer is correct and the trustworthiness of the sources. The COSINE algorithm computes the trust score of the sources as the cosine similarity between the votes of the source and the estimated probability of the answer. The TWOESTIMATE algorithm differs from the COSINE approach by calculating the trustworthiness of a source as the average probability of all the answers it provides. In return, the probability of an answer is computed as the average trust scores of the sources. In order to avoid the algorithm to converge on local optima, TWOESTIMATE normalizes the score of the sources and the answers to the closest value in $\{0,1\}$. In addition, to quickly stabilize the algorithm, the computation uses a linear combination of the non-normalized and normalized value. The THREEESTIMATE improves over TWOESTIMATE by considering a third variable that represents the likelihood a vote on an answer is correct (in other words, how hard an answer is). Under such settings, the more difficult answers a source correctly provides, the more trustworthy it is.

Pasternack et al. [54] approach the fact-finding problem by incorporating prior knowledge and propose a set of algorithms (AVGLOG, INVEST AND POOLEDINVEST). In particular, their framework translates prior knowledge, often derived from common sense as well as known facts into linear program that enforces constraints on candidate answers. The AVGLOG approach tries to mitigate the overestimation of the trustworthiness of sources that provide more answers in [40] by taking the average and logarithm values of the answer scores. In contrast, the INVEST method considers that the trustworthiness of a source is uniformly (instead as a whole) *invested* in computing the score of each answer it provides. The score of an answer is then computed as a non-linear function of all invested trustworthiness from the sources. It then calculates the trustworthiness of a source as the averaged sum of the scores of answers it provides, proportional to its investment in each of the answer score. The POOLEDINVEST algorithm updates the computation for the answer scores by linearly scaling each candidate answer such that the sum of the answer score equals to the total investments from the sources.

As a step further, latest techniques [55, 19] try to improve existing fact-finding approaches by incorporating the consideration of additional uncertainties, such as the ones from answer extraction, entity linkage and schema alignment. In [55] the authors introduce into the fact-finding solutions several uncertainties including the uncertainty in the answer extraction, the uncertainty of the sources, the similarity among the answers and group memberships to which sources may belong that can be used to infer the support for an answer it does not explicit provide. Each of these uncertainties is then quantified as a score in [0, 1] and incorporated into the corroboration algorithms proposed in [54]. Dong et al. [19] proposes the notion of *knowledge fusion* that examines the role of data extractors in computing the probability of each candidate answer. In particular, the authors adapt existing *data fusion* techniques in [17, 20] by considering the pair (Extractor, URL) as the sources. By presenting the limitation of such simple adaptation, the authors propose several refinements that include considering finer granularity of the sources, provenance selection and using the gold set to predict new extracted answers.

In our study of asserting unanimous answers in Chapter 5, we propose a corroboration algorithm that incrementally evaluates sources and answers. Such a method is motivated by the observation that sources may exhibit different trustworthiness over different queries. By evaluating queries using different trust scores from each source, our technique is able to improve both the precision and recall of the answers. Dong et al. [21] investigated a set of state-of-the-art corroboration techniques and concluded that there are possible improvements to information corroboration. In particular, the authors observed that fractions of data from the same source can have different quality and suggested that differentiating source quality for different categories of data could improve corroboration quality.

### 2.1.3 Corroboration over dynamic data

A significant factor that contributes to the data quality is the staleness of data. Indeed, data generated nowadays becomes obsolete fairly fast in a dynamic world and it is important for an information source to keep track of the latest value of an object. Unfortunately, it is often observed that sources could delay, make mistakes or even miss on value updates. We in Chapter 5 investigated a typical case of this problem in which sources provided outdated information such that it leads the sources to incorrectly agree with each other. We proposed an iterative algorithm that significantly improves the quality of the corroboration results.

There are several existing works [26, 52, 18] that have studied the corroboration problems in the presence of stale data. In those studies, it is common assumption that each source provides the full history of its observations on each data object. Fan et al. [26] studied the stale data corroboration problem in databases when multiple rows of the same real world entity consisting outdated values are inserted into the database over time without any timestamps. Those multiple rows of values about the same entity creates conflicts and confusions to the users. To address this issue, the authors modeled the problem as the studying of data currency, to identify the current values of an entity and to answer queries with the current values in the absence of timestamps. The idea here is to infer the currency order between database tuples by leveraging additional currency relationships (denoted as denial constraints) as well as the copying relationship among tables. The data currency problem is very similar to the one we investigated in Chapter 5 in that the data values are missing timestamps and we also try to uncover the current value of each listing. However, in our scenario the dataset is more limited in that we only possess a snapshot of data values. The fact that each data object has one value would render the techniques in [26] unable to infer the currency order of the multiple values of the same object.

Pal et al. [52] presented a study on a similar problem in the context of the Internet in which each source may provide values of an object at different points of its lifetime and the goal is to determine the latent history of the value updates. This particular study differs from other work in that it not only tried to identify the current value of an object, but also looks for its entire update history. Given the assumption that sources may delay and miss value updates it is extremely difficult that existing corroboration techniques can be effectively applied. To address this problem, a generative model in which each observation from a source is considered a noisy version of the true update is proposed. Several algorithms were given to infer the mappings between observations and true updates. However, as we mentioned before, this is another study in which the underlying assumption is that the observation history of each source is known, which is not true in our scenario.

Dong et al. [18] studied the problem of identifying the true values of data objects when the update history of the sources is known. With date staleness in mind, the study considers the quality of sources by the coverage, exactness and *freshness*. Their technique uses a set of Hidden Markov Models to decide whether a source copies from another source and at which historical moment it copies. In addition, the authors developed a Bayesian model to decide when the true value changes and what the new value is. Although the study has a different assumption as ours in the same way as [26, 52] in that it also assumes the history of sources updates is known, it is effective to cases in which the observations among sources are generally in agreement, which is the focus of our study. Due to the similar nature in the underlying data presented in [18], we tested our approach using the same dataset and reported the results in

Chapter 5.

Another interesting study relevant to evolving data was presented in [59] in which Rekatsinas et al. looked into the problem of source selection considering dynamic data sources whose content change over time. The key observations here are that sources which update the most frequently may not be the ones with the most fresh information. In addition, selecting more sources or acquiring more updates from a source might not necessarily increase coverage. Driven by those observations, the authors introduced algorithmic framework using statistic models to describe the quality and update patterns of the sources. While the source selection problem is NP-complete, efficient local search algorithms with theoretical guarantees are presented. While the techniques proposed in this paper is not directly applicable in our scenario, it provides insights of trade-off between the corroboration quality and cost when the price of acquiring sources or updates is not negligible.

The dynamic nature of information generated nowadays has also prompted the study of linking records that point to the same entity of different time with conflicting attribute values, namely temporal record linkage. Often, a curated data source may store the information about the same object over a long timespan. Since there might not always exist a unique key to identify each entity, it could be difficult to reconcile records that point to the same object but have conflicting values. Traditional techniques (see survey in [24]) have focused on addressing value conflict that arise from lexical heterogeneity, such as different data convention or schemas, or even data entry errors. In other words, the assumption that those techniques hold is that the conflict in values among different sources of the same entity is only *lexical*, that those values are the same in principle. However, such assumptions fail to recognize that as time goes by, the value of an entity attribute could indeed change which could make those techniques fall short.

Several works [45, 13] have addressed the problem of temporal record linkage in recent literature. Li et al. [45] proposed the idea of *time decay* which models the likelihood that the value of an entity attribute changes over a time interval. In particular, the authors considered two types of decays, namely, *disagreement decay* that captures the probability with which the values of an entity attribute over time disagree, and *agreement decay* that captures the probability with which the values of two entities at different time interval may be the same. Using a labeled set to learn the decay probabilities, several clustering algorithms were proposed to compute the similarity between records. In comparison, Chiang et al. [13] approached the problem using a different model calculating the probability that a given attribute value reappears over time. This is based on the assumption that attribute values change in a way that is dependent on its previous values. In addition, the proposed methods evaluate sets of records instead of pairs of records which improves robustness and accuracy. In recognizing and modeling sophistically the attribute value change, both works effectively addressed the temporal record linkage problem.

## 2.2 Data Dependency

When considering the sources in answer corroboration, it is important to note that we implicitly mean *independent* sources. However, in the real world it is not uncommon to see sources that are not independent in that they are copying/pasting information from other sources. An independent source provides genuine information independently, while copying sources present plagiarized data from other sources. Our work in question answering addressed this issue by considering the *originality* of the sources (Section 4.3.1) by dampening the trust score of the sources that are detected as copying sources.

Dong et al. [17, 18, 7, 16] investigated the dependence among sources and its role in data corroboration. The dependence among Web sources with respect to data corroboration was first mentioned in [7] and challenges examined. To address those challenges, the authors propose 2 important intuitions to explore and provide some preliminary solutions to discover dependencies. In [17] the authors studied how to detect dependence among sources by using Bayesian analysis. The important underlying assumption is that the more *rare* answers (*e.g.*, incorrect answers) 2 sources share, the more likely there exists dependence among them. Similar as other data corroboration algorithm examined in Section 2.1.2, an iterative algorithm that takes into consideration of copying detection as well as answer similarity is proposed and shown to perform significantly better than existing methods. Different from the analysis in [17] that focuses on a *snapshot* of answers provided from the sources, the authors in [18] investigated the dependency detection and truth finding problem when the update history of the sources is known. In particular, their techniques consider the coverage, exactness and freshness of the sources and use a Hidden Markov Model for copy detection. In addition to pair-wise copying detection based on common errors and update patterns, authors in [16] try to discover the complex dependences such as co-copying, transitive copying and copying from multiple sources between a set of structured sources. The technique works by first identifying pair-wise copying locally and then globally uncovering co-copying and transitive copying. Data dependence has also been studied in querying the deep Web [4]. In [4], the authors investigated the problem of retrieving and ranking deep web results (*e.g.*, tables) and addressed the possible source *colluding* problem. Sources are considered colluding if they copy from one another to artificially boost their relevance score. The solution to detect the source dependence is to compare results from the sources against *general queries*. The intuition is that if two sources return the same results to general queries that have a large number of results, they are likely to be dependent.

## 2.3 Top-$k$ Query Processing

Top-$k$ query processing has been studied extensively in various areas; see, e.g., [25, 48, 66]). In the typical top-$k$ query model, the score of each object is computed based on a number of attributes stored at data sources. The best known top-$k$ algorithm is the threshold algorithm (TA) proposed by Fagin et al. in [25], which requires both sorted and random accesses. The NRA algorithm improves over TA by considering only sorted access, which is cheaper than random access. Marian et al. [48] proposed the Upper strategy for the case when only random access is available. Theobald et al. [66] studied top-$k$ queries with probabilistic guarantees and proposed a series of approximate variants of TA to reduce the run-time cost. However, all these studies assume that a universal ID for each object is available in each source, which is not necessarily true in every scenario. As an instance, in the top-$k$ join query processing scenario (Chapter 6), a join result does not exist before probes are issued to each source.

Algorithms for top-$k$ join query processing have been proposed in [34, 51]). Ilyas et al. [34] introduced a rank-join algorithm that makes use of the individual orders of its inputs to produce join results ordered on a user-defined scoring function. The rank-join algorithm [34] outperforms the $J*$ algorithm [51] by using a score-guided join strategy, effectively reducing the score threshold. However, these two approaches are designed for a single join path, while we consider a more general case of a join graph, and hence cannot be directly applied. In addition, both of their models consider inner join, assuming that each answer in the top-$k$ set meets the join condition and instantiate scores on each data source, whereas in our model, a join result could instantiate a subset of the data sources and still have a high score.

A join result that has an instantiated tuple on every edge of the query graph can be translated into a DNF formula, with one join condition corresponding to each source-to-sink path. In this context, Ré et al. [58] proposed a novel approach for top-$k$ queries in probabilistic databases. The method runs several Luby-Karp simulations [36] in parallel, to approximate the score for each answer. However, their approach requires that all answers be computed a priori, and the goal is to minimize the number of simulations. In our model, pre-computing all answers means accessing all scores in each data source, which simplifies to the naive approach. In fact, our explicit goal is to minimize the number of such source probes. Note, though, that the two approaches are orthogonal: one could combine them in order to minimize both probing and computation costs.

Top-$k$ query processing in probabilistic database is studied in [60, 44, 77]. In probabilistic databases, the rank of an item is decided by its score in combination with its probability. Soliman et al. [60] investigate two top-$k$ semantics (U-Top$k$ and U-$k$Ranks) in uncertain databases and propose new formulations for top-$k$ queries. Yi et al. [77] propose an improved version of algorithms for the same query. Li et al. [44] propose two parameterized ranking functions ($PRF^{\omega}$ and $PRF^{e}$) for top-$k$ query in probabilistic databases and present novel generating function-based algorithms for efficient query processing.

Theobald et al. [64] design the TopX retrieval engine for the top-k query processing for semistructured data. In their work, they adopt the *eager* strategy to join tuples obtained from sources after a round of sorted access, which could be incorrect. In addition, TopX assumes that there exists a unique ID for each document (*doc id*) and it is accessible from each tuple, which makes it not directly applicable to our problem. As such, the *eager* strategy is limited to join tuples from sources that are neighbors of each other.

# Chapter 3

# Preliminaries

We formally define the concepts and notations that are used throughout the thesis in this chapter. Let $\mathcal{S}$ denote a set of sources and let $\mathcal{Q}$ be a set of queries. For each query $q \in \mathcal{Q}$, a source $s \in \mathcal{S}$ may express opinion with respect to the query. Such an expression, which we call a statement, may constitute a candidate answer for the query or provide partial information that is relevant to a candidate answer. We also say a candidate answer can be *extracted* from a source. As an example, a candidate answer can be extracted from a web source for the question answering task. Given a set of queries $\mathcal{Q}$ and a set of sources $\mathcal{S}$ (as well as their statements over $\mathcal{Q}$), a corroboration task is to identify the correct answer for each query $q \in \mathcal{Q}$.

## 3.1 Sources

We consider a source $s \in \mathcal{S}$ as a real-world object that expresses opinions about queries. A source can be in the form of a database table (local or hosted at a remote site) or a document (semi-structured or unstructured). We use $s(q)$ to denote the statements from a source $s$ for a query $q$. As an example, for a query 'The highest mountain in the United States', a source may provide a statement as 'Mountain McKinley'.

### 3.1.1 Extracting Statements

Depending on the nature of the sources, $s(q)$ can be extracted using the following methods.

- *Database tables*: In such cases, $s(q)$ can be extracted simply by issuing a database query. In our model, we consider that both sorted and random accesses are allowed.
- *Semi-structured documents*: In such cases, a rule based extraction tool suffices to extract $s(q)$. For instance, in order to extract restaurant listings from several web sites, we wrote

a crawler that retrieves the listing web pages and used regular expression and a set of simple rules to identify restaurant listings.

- *Unstructured documents*: Identifying answers for natural language queries in free texts is a complex problem and poses significant challenges in a corroboration system. In such cases, a suite of information extraction tools need to be employed to pinpoint the statements from the sources. In our question answering task, we used a rule-based approach coupled with existing linguistic tools such as tokenizer and named entity recognizer to accurately find answers for the queries (see detail in Section 4.2.1).

In addition to extracting answers, another challenge is how to merge similar answers or entities. Often times, the same answer could be presented in different forms from different sources. Record linkage is a non-trivial problem that has been studied in the research community. To solve this problem, we employ empirical techniques (Section 4.2.2 and Section 5.6.2) for answer reconciliation.

### 3.1.2 Scoring Sources

We associate with each source $s$ a measure $\boldsymbol{\sigma}(s)$ that represents its trustworthiness. The trust score is a real number between 0 and 1, with 1 indicating a perfect source and 0 indicating a completely wrong source. We define sources with a trust score between 0.5 and 1 as *positive* sources. In principle, positive sources are the sources that have more correct statements than incorrect ones. Similarly, a *negative* source is defined as a source with a trust score between 0 and 0.5. For convenience, we use $\boldsymbol{\sigma}(\mathcal{S})$ to denote the collective trust scores for all the sources in $\mathcal{S}$.

**Single-value and multi-value trust scores**: Traditionally, corroboration techniques [79, 40, 28, 49, 54, 29, 55] consider the trust score $\boldsymbol{\sigma}(s)$ for source $s$ as a singular numerical value between 0 and 1. This practice works fairly well in a number of applications. However, as we demonstrate in Section 5.4.2 of Chapter 5, using one trust score for each source (coined a single-value trust score based approach) does not work well in some scenarios. To address its limitations, we propose a novel method (Section 5.5) that uses different trust values toward different queries for each source (called a multi-value trust score based approach).

## 3.2 Answers

### 3.2.1 Identifying Answers

In order to identify the correct answer for each query, the first step is to find all candidate answers based on the statements from the sources. As we mentioned in Section 3.1, a statement from a source in itself can be seen as a candidate answer in certain scenario (*e.g.*, 'Mountain McKinley' for the query 'The highest mountain in the US'). As another example, for the query 'Is Danny's Grand Sea Palace open?', the statements 'Yes' from Yellowpages and Citysearch can also treated as a candidate answer.

In some other scenarios, however, the statements from the sources serve as partial information that can be used to infer a candidate answer. For instance, consider the query 'The elevation of the highest mountain in the US', the statement above (*i.e.*, 'Mountain McKinley') only serves as partial information and we need another statement ('Mountain McKinley has a summit elevation of 20,237 feet') to complete a candidate answer.

Let $f$ be a candidate answer and let $C(f) = \{s_1(q), s_2(q), \cdots\}$ be the statements from $s_1, s_2, \ldots$ that are relevant to $f$. In the case where statements are in themselves candidate answers, each $s_i(q)$ in $C(f)$ represents an occurrence of $f$ (Chapter 4 and 5). Otherwise, a mechanism needs to be derived candidate answers (Chapter 6).

### 3.2.2 Scoring Answers

In order to identify the correct answer given multiple candidate answers, we propose techniques to compute a probability $\sigma(f)$ for each candidate answer $f$ that represents the likelihood that $f$ is correct. A corroboration considers the answer with the highest probability score as the correct answer. For convenience, we use $\mathcal{F}_i$ and $\mathcal{S}_j$ to denote the set of answers from source $s_i$ and the set of sources that have statements for answer $f_j$, respectively.

## 3.3 Corroboration

The objective of a corroboration system is to identify the correct answer for each query (based on $\sigma(f)$) as well as to estimate the trust score $\boldsymbol{\sigma}(s)$ for each source. In principle, the relationship between the computation of $\sigma(f)$ and $\boldsymbol{\sigma}(s)$ is intertwined. On one hand, the probability score of $\sigma(f)$ depends on the trustworthiness of the sources. On the other hand, the trust score for a source is decided by the quality of answers it provides. We use $Corrob()$ and $Update()$ to denote the operations that used to calculate $\sigma(f)$ and $\boldsymbol{\sigma}(s)$, illustrated in Equation 3.1.

$$\begin{aligned} \sigma(f_i) &= Corrob(\sigma(\mathcal{S}_i)) \\ \sigma(s_i) &= Update(\sigma(\mathcal{F}_i)) \end{aligned} \tag{3.1}$$

As a simple example, a frequency-based corroboration method computes the score of an answer as the number sources reporting it. In other words, $Corrob()$ can be implemented as $|\mathcal{S}_i|$. Since all the sources are treated equally in such a method, $Update()$ is ignored by the frequency-based method.

In practice, the design of a effective $Corrob()$ and $Update()$ is the key to the performance of a corroboration system. Intuitively, due to the relationship between $\sigma(f)$ and $\boldsymbol{\sigma}(s)$, we need to know one of the scores to calculate the other, as is the case we describe in Chapter 4. However, even without any knowledge with either score, we can still adopt an iterative algorithm to derive estimations for both scores. Essentially, we can start with a default score for each source (or answer) and repeatedly calculates $\sigma(f)$ and $\boldsymbol{\sigma}(s)$ using a designed $Corrob()$ and $Update()$ implementation until convergence is reached, illustrated below.

$$\begin{aligned} \sigma^{(k)}(f_i) &= Corrob(\sigma(\mathcal{S}_i^{(k-1)})) \\ \sigma^{(k)}(s_i) &= Update(\sigma^{(k)}(\mathcal{F}_i)) \end{aligned} \tag{3.2}$$

where $\sigma^{(k)}(f_i)$ is the score for answer $f_i$ at $i$th iteration and $\sigma(\mathcal{S}_i^{(k-1)})$ represents the trust score for $\mathcal{S}$ at the $(i-1)$th iteration.

## 3.4 Cost Model

In addition to returning the correct answer for the queries, another objective of our corroboration system is to compute the answers efficiently. Since our system deals with heterogeneous types of sources, we abstract the cost of corroboration systems using the number of *probes* operation. A probe could be a database access (either sorted or random access), a document retrieval, or a look-up in memory depending on the nature of the sources. In each of the scenario we discuss in the following chapters, we use the same metric when evaluating different corroboration systems.

## 3.5 Summary

We give a formal and detailed description of the concepts and notations in this chapter. Moreover, a brief discussion on how a corroboration system works is provided. In each following application, we address the challenges using the concepts and notations introduced in this chapter.

# Chapter 4

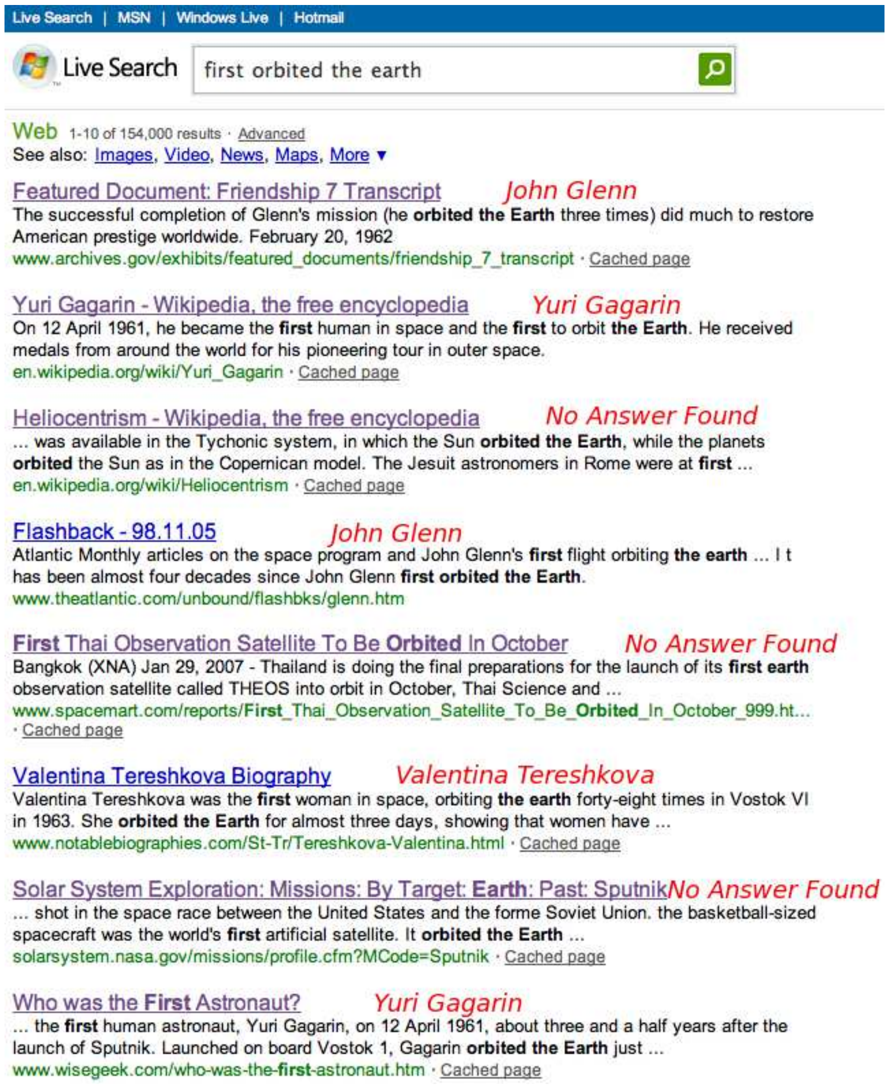
# Corroborating Answers from Multiple Sources

## 4.1 Introduction

We start the technical contribution of this thesis by discussing how corroboration techniques can help the task of question answering. Given a user issued factoid query, the question answering task is to identify the correct answer from the Web. In this thesis, we consider a *factoid* query as one to which there exists a factual answer. For instance, a factoid query could be '*Who is Tom Hanks' wife*', or '*What is the highest mountain in the US*'. To make the description and evaluation of our technique tractable, we focus on factoid queries with succinct answers. For instance, our discussion leaves out queries such as '*Who is Abraham Lincoln?*' and '*How to repair a computer?*'.

Typical web search engines return a list of web pages (or sources) that matches a set of keywords input by users. Web search engines are increasingly efficient at identifying the best sources for any given keyword query, and are often able to identify the answer within the sources. Unfortunately, as we mentioned in chapter 1, many web sources are not trustworthy, because of erroneous, misleading, biased, or outdated information. With many web sources providing similar information on the Internet, users often have to rummage through a large number of different sites to both retrieve the information in which they are interested, and to ascertain that they are retrieving the correct information. In many cases, users are not satisfied with —or do not trust— the results from any single source, and prefer checking several sources for corroborating evidence, as illustrated in the following examples:

**EXAMPLE 1:** *Consider a user interested in buying a car, and considering a specific brand and make (e.g.,* Honda Civic*). One of the criteria influencing the decision is the gas mileage of the car. However, the gas mileage information available on the Internet differs not only based on the year of the car, but also based on the source from which the information is extracted: the*


Live Search | MSN | Windows Live | Hotmail
Live Search
first orbited the earth
Web 1-10 of 154,000 results · Advanced
See also: Images, Video, News, Maps, More ▾

Featured Document: Friendship 7 Transcript — John Glenn
The successful completion of Glenn's mission (he **orbited the Earth** three times) did much to restore American prestige worldwide. February 20, 1962
www.archives.gov/exhibits/featured_documents/friendship_7_transcript · Cached page

Yuri Gagarin - Wikipedia, the free encyclopedia — Yuri Gagarin
On 12 April 1961, he became the **first** human in space and the **first** to orbit **the Earth**. He received medals from around the world for his pioneering tour in outer space.
en.wikipedia.org/wiki/Yuri_Gagarin · Cached page

Heliocentrism - Wikipedia, the free encyclopedia — No Answer Found
... was available in the Tychonic system, in which the Sun **orbited the Earth**, while the planets **orbited** the Sun as in the Copernican model. The Jesuit astronomers in Rome were at **first** ...
en.wikipedia.org/wiki/Heliocentrism · Cached page

Flashback - 98.11.05 — John Glenn
Atlantic Monthly articles on the space program and John Glenn's **first** flight orbiting **the earth** ... I t has been almost four decades since John Glenn **first orbited the Earth**.
www.theatlantic.com/unbound/flashbks/glenn.htm

**First** Thai Observation Satellite To Be **Orbited** In October — No Answer Found
Bangkok (XNA) Jan 29, 2007 - Thailand is doing the final preparations for the launch of its **first earth** observation satellite called THEOS into orbit in October, Thai Science and ...
www.spacemart.com/reports/**First**_Thai_Observation_Satellite_To_Be_**Orbited**_In_October_999.ht... · Cached page

Valentina Tereshkova Biography — Valentina Tereshkova
Valentina Tereshkova was the **first** woman in space, orbiting **the earth** forty-eight times in Vostok VI in 1963. She **orbited the Earth** for almost three days, showing that women have ...
www.notablebiographies.com/St-Tr/Tereshkova-Valentina.html · Cached page

Solar System Exploration: Missions: By Target: **Earth**: Past: Sputnik — No Answer Found
... shot in the space race between the United States and the forme Soviet Union. the basketball-sized spacecraft was the world's **first** artificial satellite. It **orbited the Earth** ...
solarsystem.nasa.gov/missions/profile.cfm?MCode=Sputnik · Cached page

Who was the **First** Astronaut? — Yuri Gagarin
... the **first** human astronaut, Yuri Gagarin, on 12 April 1961, about three and a half years after the launch of Sputnik. Launched on board Vostok 1, Gagarin **orbited the Earth** just ...
www.wisegeek.com/who-was-the-**first**-astronaut.htm · Cached page

Figure 4.1: Results for the query “first orbited the earth” using MSN Search

*official manufacturer web site (up to 51 mpg in the Honda Civic example) has a different value than some other commercial web sites (40 mpg.* `Autoweb.com`*; 30/40mpg,* `Car.com`*). None of these sources, all of which appear in the first page of results for the query “Honda Civic 2014 gas mileage” using* MSN Search *has the “perfect” answer for the query, but they all provide valuable information to the user.*

**EXAMPLE 2:** *Consider a user who wants to find out who the first astronaut who orbited the earth was. Issuing the query “first orbited the Earth” to a commercial search engine returns a list of web sources that are relevant to the query but provide different extracted answers as shown in Figure 4.1. The correct answer, Yuri Gagarin, does not appear in the first search* *engine result; in fact, there are several results that does not contain any valid answer to the user's information need. However, several answer extracted from the search engine result pages are potentially useful to the user as they provide correct answers to refinements of the user's query: Valentina Tereshkova was the first woman who orbited the earth, and John Glenn was the first American to do so.*

A naive approach to try to identify the correct answer would be to return the most frequent answer found among the search engine query result pages. While this method can efficiently eliminate outlier answers such as typos, it fails to consider the fact that answers extracted from different pages are rarely equally important in answering the query, as all pages are not equally trustworthy. In fact, such a frequency-based approach can be viewed as a simplified instance of the corroboration techniques by considering all sources have the same trustworthiness. In addition, it opens the gate to malignant behavior from spammers, who would be tempted to create multiple pages containing similarly erroneous information to boost the score of their chosen answer.

In this chapter, we propose a framework to corroborate query results from different sources in order to save users the hassle of individually checking query-related web sites to corroborate answers. In addition to listing the possible query answers from different web sites, we rank the answers based on the number, relevance, and similarity of the web sources reporting them, as well as the prominence of the answers within the sources. The existence of several sources providing the same information is then viewed as corroborating evidence, increasing the quality of the corresponding information, as measured by a scoring function used to order answers. Our techniques are built on top of a standard web search engine query result, and use existing information extraction techniques to retrieve answers from web pages. Corroborating answers from web sources presents several challenges:

- The main challenge of answer corroboration is the design of a meaningful scoring function. The scoring function should aggregate similar answers and take into account a variety of parameters to identify the best answers.
- Accessing all the pages that match a given web search query to retrieve and compare answers would obviously be very inefficient. We need to select web sources that are

most likely to contain the best answers.

We propose the following contributions to address these challenges:

- *Scoring of corroborated answers.* We propose a framework to score corroborated answers. Our approach considers several factors to score both the relevance of a page to the query and the importance of the query answer within the page. By combining these two factors we can assign a score to each individual answer based on how likely the answer is to be the correct answer. We then aggregate the score of similar answers. To the best of our knowledge, our techniques are the first to consider not only the frequency of the answers in the web search engine result, but also the relevance and originality of the pages reporting the answers, as well as the prominence of the answer within the page (Section 4.3). In particular, our web page relevance score is based on search engine rankings and modeled by a Zipf's Law, an intuition empirically validated using the user clicks from a search engine log.

- *Selecting the web sources from which to retrieve the answers.* By focusing on the pages that are most likely to contain good answers we are able to save on query processing time. This is related to work on top-$k$ query processing and the Threshold Algorithm [25]; however, score bounds information, commonly used in top-$k$ query processing, cannot be used in a corroborative framework. We propose a method to consider a prefix of the search engine query result for information extraction, dynamically deciding the size of this prefix based on the distribution of answers (Section 4.4). We experimentally evaluate the effect of the prefix size and show that our method is effective at reducing the number of pages necessary to corroborate answers.

- *Evaluating the quality of our proposed approach.* We conducted a novel extensive qualitative and quantitative experiments on queries selected from the TREC Question Answering Track [67] and from a log of MSN query searches. Our experimental results show that data corroboration significantly improves the quality of answers (Section 4.5). We also show that for MSN queries, our corroborative answers correlate with user clicks in the MSN search log.

## 4.2 Extracting Answers

Our corroboration system is built on top of an existing search engine. Given a user keyword query, our first step is to extract the candidate answers from the web pages returned by the search engine. The challenge is to efficiently retrieve, and identify, from these web pages the data that qualifies as an answer to the query. For instance, to retrieve answers for the query in Example 1, we need to identify gas mileage values that correspond to a 2014 Honda Civic from the search engine web page results.

In addition, it is often observed that the same answer may appear in different form in different sources. For example, we found two answers (“John Glenn” and “John H. Glenn”) for our Example 2 query. While the text of these two answers is slightly different, it is highly probable they refer to the same answer. Our answer extraction system solves this problem by computing the cosine similarity score between answers and aggregating similar answers if their similarity score is above a certain threshold.

### 4.2.1 Answer Extraction

*Information extraction*, the process of extracting relevant information from documents using text and structure, is a complex problem that has been the focus of many work in the Natural Language Processing and Machine Learning communities [50], among others. Since the focus of our work is on the corroborative aggregation and scoring of answers, we opted to use regular expressions techniques to extract answers from web pages. In addition, our current implementation works on queries with succinct answers, such as the ones illustrated in Example 1 and 2.

It is relatively straightforward to identify numerical answers within web pages. For instance, mileage information will typically be of the form “$x$ mpg,” or “mpg of $x$,” where $x$ is a numerical value. Note that our extraction technique considers multiple units, and does the proper conversions, for each numerical query (e.g., “feet” and “meters” for a length query).

It is more complicated to extract answers for a query that calls for a textual answer. State-of-art IE systems [1, 10, 31, 32, 33, 53, 56] have used a lot of linguistic tools, namely syntactic parser, part-of-speech tagger, named entity tagger, WordNet.

Instead of employing a full featured information extraction system, existing works [57, 35, 62, 75] in the information retrieval community have shown success by using a simplified answer extraction component. Radev et al. apply a part-of-speech tagger to phrases and computes the probability of phrase type to match the query category. Jijkoun and de Rejke pinpoint the answers for a query by looking at Frequently Asked Questions (FAQ) archives. If a question in the FAQ archive is found to match the query, the non-question text block immediately following the question is identified as the answer for the query. The QA system in [62] retrieves answer sentences based on keyword matching. The QUALIFIER system in [75] performs answer selection by matching the expected answer type to the NLP results of the query and returns the named entity in the candidate sentence.

Our answer extraction is similar as the techniques used in[75]. Given a web page, we first apply a HTML parser to obtain the text content for answer extraction. We choose the Jericho HTML parser[1], an effective Java open source HTML parser to obtain the plain text of the web page. We then use a tokenizer to tokenize and tag the plain text from the first step. The Stanford Name Entity Recognizer (NER) is a Java implementation of a Named Entity Recognizer which uses a linear chain Conditional Random Field (CRF) sequence models. In particular, the Stanford NER system segments the plain text into sentences and words. After giving each word a tag, we apply extraction rules for each query by selecting the sentences that contain the extraction rule. The answer is then identified as the words from the sentences that match the expected answer type. The rules for each query are a set of automatically created texts that may appear around the answer. The rules are not query specific but rather are created based on the interrogative word to apply to different types of queries. We give a few examples of such rules for different queries in Table 4.1:

By using rules such as the ones shown in Table 4.1, we can extract answer for each query. As an example, consider the query: “Who is the speaker of the Lebanese Parliament”. We can create rules “is the speaker of the Lebanese parliament” and “the speaker of the Lebanese parliament is” to extract query answers.

While these rules cover a wide variety of scenarios, they are insufficient in some cases.

[1]http://jerichohtml.sourceforge.net/doc/index.html

| | |
|---|---|
| `Who is/was $x` | "is $x", "$x is" |
| `Who VBD $x` | "VBD $x", "$x is/was VBN" |
| `Where is $x` | "$x is" |
| `Where is $x VBN` | "$x is VBN" |
| `Where did/does $x VB $y` | "$x VBZ/VBD $y" |
| `Which $x is/was $y` | "is/was $y", "$y is/was" |
| `Which $x VBZ/VBD/VBN $y` | "$y is/was VBN", "VBZ/VBD/VBN $y" |
| `What $x is/was $y` | "is/was $y", "$y is/was" |
| `What $x does/did/have/has $y VB/VBN` | "$y VBD/VBN $x" |

Table 4.1: Generating rules for queries

Consider the following example shown in the title of a news article:

*"Nabih Berri reelected as the speaker of the Lebanese Parliament"*

Clearly, "Nabih Berri" is an answer to the query, but the two previously created rules cannot extract it. Therefore, in addition to the rules created as above, we introduce a set of *relaxed* rules for each query. The *relaxed* rules include the noun/verb phrase or part of the noun/verb phrase, possibly in combination with a specific adjective (*e.g.*, "current"). We manually create a few specific adjectives that may appear around the extraction rule based on heuristics. For instance, the *relaxed* rules for the above query include "the speaker of the Lebanese Parliament"; "the speaker"; "the current speaker is"; "is the current speaker".

Applying the Stanford NER recognizer will tag "Nabih Berri" as "`PERSON`". Since the answer to this query is expected to be a `PERSON` entity and this sentence matches one of the rules of this query (*i.e.*, "the current speaker is"), "Nabih Berri" is extracted as an answer to this query. Note that it is possible to extract multiple answers from the sentence which matches the extraction rule. We show how we assign scores to each of the extracted answers in Section 4.3.2.

### 4.2.2 Answer Aggregation

After we have extracted answers from different sources, we need to combine similar answers. Due to the poor quality (spelling errors, typographical errors) of a large portion of online content, answers extracted from different sources bearing textural difference may actually point to the same information. A simple example is for numerical queries, different sources may present results using different unit. For instance, for our Example 1 query we may find answers "26 mpg (mile per gallon)" and "11 km/l (kilometers per liter)" from different sources which reflect the same gas mileage information for a Civic in city drive. Our solution to such cases is to convert all extracted answers into the same unit and combine answers that are similar in value. In our implementation, we consider answers that are within 5% difference in values as similar answers.

For queries with textual answers, we are dealing with problems of finding similar strings. Existing work [42, 30, 63] has been proposed using cosine similarity score to effectively handle spelling errors and rearrangement of words when comparing strings. The key idea is to obtain the *tf-idf* score vector for each string and compute cosine similarity score for each pair of strings. In particular, Koudas et al. leverage the semantic similarity if such information is available. In our corroboration system, we apply the *tf*-based[2] cosine similarity method to detect similar answers. The cosine similarity score of two answers is computed using the word frequency vector of the two answers. Given two tokenized answers, we first need to obtain the word frequency vectors for the two answers $W_1$ and $W_2$, the cosine score is then computed as the dot product of the two vectors divided by the square root of the product of the vector dot products of each score vector with itself. For example, assuming we have extracted answers "John Glenn" and "John H. Glenn" for our Example 2 query, the word frequency vectors $W_1$ and $W_2$ for the two answers are (1, 0, 1) and (1, 1, 1). The cosine similarity score is therefore computed as the dot product of $W_1$ and $W_2$ (which is 2) divided by the square root of product of each vector with itself (which is $\sqrt{2} \cdot \sqrt{3}$). The cosine similarity score (0.82) is then compared against a user-defined threshold to test if the two strings can be classified as similar answers. In our implementation, we use the threshold of 0.8 and we recognize "John Glenn" and "John H. Glenn" as similar answers.

## 4.3 Scoring Answers

Once we have extracted answers from the web pages returned by the search engine, we need to identify the best answer to the query. For this purpose, we need to assign scores to each extracted answer. We then aggregate the score of similar answers to identify the best corroborative answer.

[2]We did not use the *idf* score since the document corpus for each query is limited by the number of documents returned for each query.

In the absence of any knowledge regarding the trustworthiness of the sources, it is difficult to infer the correctness of the answers. As a result, an iterative corroboration (Equation 3.1) needs to be employed to identify the correct answer. However, since we are leveraging the web results from a search engine as the information sources, we are able to model their quality from which the answers are extracted.

In order to assign a score to each extracted answer, we first assign to each web page a score that represents the likelihood that an answer found within the page is the correct answer. Then, for each answer found within a web page, we assign a score that represents the probability that this answer is the correct answer within the page. Finally, scores of similar answers are aggregated to provide a corroborative answer score.

We compute the score of an answer $f$ extracted from a given web page $e$ as the product of the score of the page ($\sigma(s)$) and the score of the answer within the page ($\sigma(f|s)$).

$$\sigma(f,s) = \sigma(s) \cdot \sigma(f|s) \tag{4.1}$$

In the rest of this section, we detail our scoring approach. We explore several scoring components that can be used in conjunction, or separately. We will discuss the impact of each of the components, and evaluate them experimentally in Section 4.5.2. In Section 4.3.1, we propose a scoring method of individual web pages. We then propose techniques to estimate the score of an answers within a page in Section 4.3.2. The corroborative score of an answer among all search engine query result pages, and taking into account each of our proposed component, is given in Section 4.3.3.

### 4.3.1 Scoring Web Pages

The score of an answer depends on the quality of the web page from which the answer is extracted. We consider two factors in measuring the quality of a web pages: the relevance of the page (Section 4.3.1), and the originality of the page (Section 4.3.1).

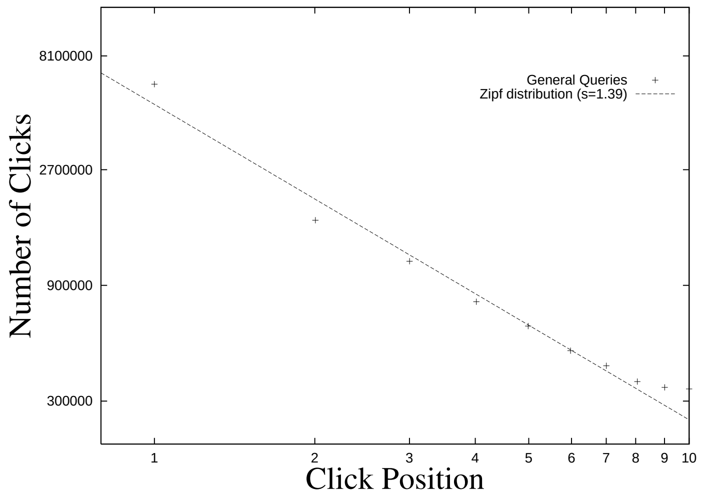


Figure 4.2: The log-log plot for the Number of user clicks as a function of the position of the page in the search engine query result

**Web Page Relevance**

Search engines rank web pages according to (among other factors) the relevance between the page and the query keywords. An intuitive method for web page scoring is to use the page rank score of web pages as given by the search engine. Unfortunately the search engine does not provide its internal score along with the ranking. The *tf-idf* score has been widely used in the information retrieval community to compute the relevance of a document with respect to a keyword query, but our system is built on top of a search engine; as such we do not have access to indexes of the whole web. Therefore we use the individual ranks of web pages in the search engine result list as a measure of relevance.

As we traverse the search engine query results, the quality of the match decreases. We consider that a page ranked highly by the search engine is more likely to provide good information than a page with a lower rank. Therefore an answer found within a higher ranked page has a higher probability than one found within the lower ranked page to be the correct answer. To model this decrease of the relevance of web pages as we go down the search engine result we use the Zipf's Law distribution function [83], commonly used in Natural Language Processing to model the distribution of words in languages where very common words have very high frequencies.

Previous work [8] refers to the commonness of Zipf-like distribution in web access patterns. For instance, there is evidence that the popularity of web pages follows a Zipf distribution. In this work, we are interested in the importance of web pages as we go down a search engine query results. We investigated the click patterns of a query log of about 15 million MSN queries. Figure 4.2 shows the log-log plot for the distribution of user clicks depending on the position of the page in the search engine query result, for the first 10 results and for all queries in the log. We only report on the first 10 results here since the number of user clicks shows a sharp decrease after the tenth result (and similarly after the twentieth, thirtieth, etc.) as search engines return results 10 at a time and users are reluctant to go beyond the first page of result.

We used curve-fitting techniques[3], based on a linear regression in the log-log plot to approximate the distribution of the user clicks per position. Our results show that the distribution can be approximated to a power law distribution, with an exponent parameter ($e$) of 1.39, and constant close to 1.

These results show that the Zipf distribution is adequate to approximate the decrease in page relevance. We then define the relevance score of a page $e$ as:

$$\sigma(s) = \frac{1/r(s)^e}{\sum_{i=1}^{N} 1/i^e} \tag{4.2}$$

where $N$ is the total number of pages considered (i.e., the estimated size of the search engine query result), and $r(p)$ is the rank of page $p$ in the search engine query result. We normalize the score so that the sum of all page scores are summed to 1.

The $e$ exponent parameter has an impact on the slope of the score distribution. It quantifies how quickly the quality of answers degrades when we traverse the search engine result list. In our system, varying $e$ has an impact on both the quality of answers and the number of pages retrieved to identify the best answer. We explore the effect of $e$ experimentally in Section 4.5.2.

### Web Page Originality

In addition to the web page relevance, we take into account the originality of the pages to reduce the effect of similar information coming from sources in the same domain, or sources

[3] `http://www.fast.u-psud.fr/ezyfit/`

that seem to be mirror (or copy-paste) information from each other. These pages tend to show some strong correlation as they reflect information coming from the same real-world sources. As an example, many web pages directly copy material from high-profile web sources such as *Wikipedia*. Errors in the original web site are therefore propagated to independent web pages. Our motivation for using corroboration is to confirm evidence from several autonomous sources. Considering sources that mirror each other as completely separate sources for corroboration would then artificially increase the weight of the original real-world sources (such as *Wikipedia*), and would open the door to some possible malignant behavior from content providers in order to boost the ranking of a given answer. However, discarding the redundant source is not a good option either, as its existence leads some more credence to the information, although not as much as if it were a source containing original information.

Since we aim at corroborating answers from different web pages, duplicated pages which tend to have similar content should not have as much weight in the corroboration as pages that contain original content. However, they should still be taken into account as corroborative evidence, as there is a possibility they reflect similar information coming from independent sources. Our solution is to dampen the score of a page each time a duplication is detected. Our implementation detects suspected copy-paste as duplicated text around answers. Detecting near duplicate web pages is a non-trivial task and has been the focus of a large body of research. Often, two pages sharing the same core content may be classified as independent sources using byte-wise comparison due to different framing, advertisements, and navigational banners. Among those existing works, the SpotSigs technique proposed in [65] proves to be an effective way to detect such near duplicate pages. The key idea in SpotSigs technique is to create a *robust* document signature with a natural ability to filter out noisy components of Web pages. In our corroboration system, we implemented the SpotSigs technique to detect copy/paste pages. Our current implementation is Boolean, a page is a copy or it is not, but we could easily extend our system to use more complex copy detection tools such as [9, 6] to identify different degrees of duplication, and use this information to provide a finer granularity of web page originality scores.

We introduce a parameter $\beta$ to quantify the redundancy of duplicated pages. When traversing a search engine result list, we check whether each page contains original content, or whether it is similar to a page with higher rank. If for a page $e$ there exists $d_m(s)$ higher-ranked pages sharing the same domain and $d_c(s)$ higher-ranked pages generated from copy/paste information, we adjust the relevance of the page as follows:

$$\sigma(s) = \frac{1/r(s)^e}{\sum_{i=1}^{N} 1/i^e} \cdot (1-\beta)^{d_m(s)+d_c(s)} \tag{4.3}$$

The first occurrence of a page from any given domain (or from duplicated web pages) will therefore be assigned its full score, only subsequent similar pages will have their scores dampened. This ensures that we are taking into account all original information, and limits the weight of redundant information in the scoring.

### 4.3.2 Scoring Answers within Web Pages

It is common for several, possibly different, answers to be extracted from a single web page. This can be due to some error or uncertainty in the answer extraction process, or to the fact that the web page does contain several answers. To identify the best answer, we are then faced with the challenge of scoring these multiple answers. If only one answer $f$ is extracted from a page $e$, it gets a score of 1 for that page ($\sigma(f|s) = 1$). A simple way to score multiple answers stemming from the same page would be to assign each of them a score of $1/N(s)$, where $N(s)$ is the number of answers extracted from page $e$. One problem of the above method is that all the answers extracted from the same page are rarely equally helpful in answering queries. Consider the following text from which we can extract two answers from a single web page for the query in Example 2 "first orbited the earth".

**EXAMPLE 3**. *"Now you could have asked, even at the time, what was so special, so magical, about John Glenn, since a year before him a Russian, one Yuri Gagarin, was the first human to orbit the Earth in space and four months after him another Russian did it 17 times".*

By applying the answer extraction techniques (Section 4.2) we could extract two answers, namely, "Yuri Gagarin" and "John Glenn", which are underlined in the above example. Unfortunately, due to the limitation of the simple information extraction techniques we are using, it is difficult to figure out which one the correct answer is to the query. Our solution is to consider the *prominence* of each answer extracted within the page. We define the *prominence* $M(f, s)$

of an answer $f$ from page $e$ as the inverse of the distance $dis(f)$ between this answer and the extraction rule (*i.e.*, “the first human to orbit the earth” in this case).

$$M(f, s) = \frac{1}{dis_{min}(f)} \tag{4.4}$$

We compute $dis(f)$ as the number of tokens plus 1 between the extraction rule and answer $f$, with a minimum distance of 1. As mentioned in Section 4.2, it is possible for multiple answers to be extracted via an extraction rule. The closer an answer is to the extraction rule, the more prominent it should be to answer the query. This approach is based on the assumption that in most cases, relevant answers will be close to the extraction rule in the web pages. We compute the score of the answer within the page as its normalized prominence. As for example 3, we have $dis$(”Yuri Gagarin”) = 2 and $dis$(”John Glenn”) = 12. Therefore the prominence scores for the two answers are 0.86 and 0.14 respectively. Formally, if an answer $f$ is extracted from page $e$ out of $N(s)$ answers, we define the score of answer $f$ to be as follows:

$$\sigma(f|s) = \frac{M(f, s)}{\sum_{i=1}^{N(s)} M(f_i, s)} \tag{4.5}$$

Our simple method to compute prominence scores leads to improvements in answer quality (Section 4.5.2). It is possible that the use of more refined information extraction techniques [50, 22] that return answers with an associated confidence score from which we could derive our prominence score would result in further improvements.

Finally, some web pages may provide several answers; as for Example 1 the `Car.com` web site gives two answers for our query: 30 mpg (city), and 40 mpg (highway). Our current implementation penalizes answers that are not unique in their sources. In some cases, as in the city/highway example, multiple answers in a given web page may be due to different context. We plan to add context to our scoring approach in the future, possibly enabling our techniques to output different answers for different contexts.

### 4.3.3 Corroborating Answers

Taking into account the scores of web pages as well as the scores of answers within web pages, we can assign the score of an answer $f$ from a page $e$ as:

$$\sigma(f, s) = \frac{1/r(s)^e}{\sum_{i=1}^{N} 1/i^e} \cdot (1-\beta)^{d_m(s)+d_c(s)} \cdot \frac{M(f, s)}{\sum_{i=1}^{N(s)} M(f_i, s)} \tag{4.6}$$

The *frequency* of the answers in the set of pages is considered in our corroboration approach. Intuitively, an answer that appears in 10 pages is more likely to be the correct answer than an answer that appears in one page, unless those 10 pages have very low scores. Formally, if $\sigma(f, s_i)$ is the score of answer $f$ from page $s_i$, the corroborative score of answer $f$ is given by:

$$\sigma(f) = \sum_{i=1}^{n} \sigma(f, s_i) \tag{4.7}$$

where $n$ is the number of pages we consider from the search engine query result.

## 4.4 Retrieving Pages

Finding and computing the scores of answers are not the only challenges we face when corroborating answers from search engine query results; another challenge is to select the set of result pages from which we extract information. As we go from higher-ranked to lower-ranked pages, we have to decide how deep we should go in the search engine query result. Accessing all the pages that match a given web search query to retrieve and compare answers would obviously be impractical and inefficient. In addition, lower-ranked pages, which show little correlation with the query tend to give "noise" instead of useful information. Work on top-$k$ query processing algorithms have focused on adaptively reducing the amount of processing done by query evaluation techniques by ignoring data that would not be useful to identify the best answers to a query [25, 47]. However, these techniques cannot be directly applied to our scenario as they rely on query models where the score upper bounds of query results are known, and use this information during query processing. In contrast, in our corroborative model, the score of an answer can potentially grow every time a new page is retrieved.

We adapt ideas from work on top-$k$ query processing to the answer corroboration problem by adaptively selecting a subset of search results from which to extract answers, based on the current scores of the retrieved answers. As was suggested in [41], we focus on estimating the maximum possible score increase that an answer could receive from pages that have not been retrieved, in the absence of score upper bound information. Succinctly, we process web pages in the search engine result order, and stop retrieving new pages when the score of newly discovered answers would not be high enough to impact the overall corroborative answer score ranking.

We use our page relevance score (Section 4.3.1) to decide when to stop retrieving new pages from the search engine result. As we traverse the search engine result list, the relevance score of new pages we encounter decreases following a Zipf distribution. We stop retrieving new pages when the sum of the relevance scores of the unretrieved pages is too small for any answer extracted from new pages to cause any significant change to the corroborative answer list.

The maximum possible score increase is defined as the sum of all scores of unseen pages. Based on Equation 4.2, this score $I_{Max}$ is defined as:

$$I_{Max} = 1 - \sum_{i=1}^{r} \sigma(s_i) \tag{4.8}$$

where $r$ is the rank of the last retrieved page, and $s_i$ is the page retrieved at rank $i$. The value of the maximum possible score increase $I_{Max}$ constantly decreases as we traverse the search engine query result.

During corroboration processing, we maintain a threshold variable $T$, which represents the value required to cause significant changes to the current result list. Our current implementation considers $T$ to be the current difference in score between the top-1 answer and the top-2 answer. That is, we stop retrieving new pages when the current top-1 answer cannot change, i.e., when $T \geq I_{Max}$.

Although we dynamically retrieve pages and stop as soon as the remaining unretrieved pages will not make significant changes to the current answer list, we may still end up retrieving a large number of pages. This is due to our Zipf's Law model where a large number of lower-ranked pages may add up to a high score, as shown in Figure 4.3, which shows the decreasing value of $I_{Max}$ as we traverse the search engine result, up to page 50, for various values of $N$ (the estimated total number of pages in the query result).

To address this, we limit the maximum number of pages our system will retrieve. This limit

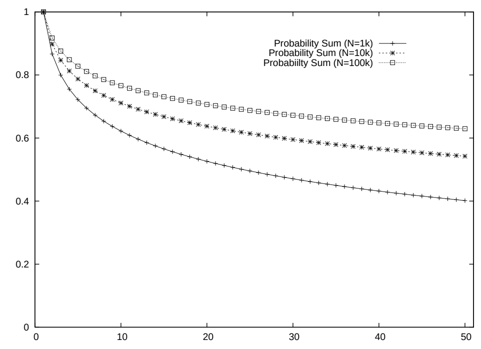


Figure 4.3: Decrease in value of $I_{Max}$ as we traverse the search engine query result.

can be user-defined. By limiting the maximum number of pages retrieved, we are ignoring a subset of the web pages as well as the corresponding score increase they may bring to the corroborated answer scores. This unused potential score increase depends both on the maximum number of pages retrieved, and on the expected total number of page considered (as estimated by the search engine query result size). The normalization factor of Equation 4.2 is adjusted to consider the maximum number of pages $maxPage$, as follows:

$$\sigma(s) = \frac{1/r(s)^e}{\sum_{i=1}^{maxPage} 1/i^e} \tag{4.9}$$

We use deterministic bound information to decide when to stop retrieving new pages. An interesting direction to investigate would be the use of probabilistic bound information instead, in the manner of [66]. We are planning to investigate this approach in future work.

## 4.5 Evaluation

In this section, we present our in-depth experimental evaluation of our corroboration approach for web searches. We describe our experimental setup in Section 4.5.1. Section 4.5.2 focuses on queries taken from the TREC Question Answering Track and derives the best setting of $e$, *maxPage* and $\beta$. Results for real-user queries from a MSN query log are given in Section 4.5.3.

### 4.5.1 Setup

Our system is implemented using Java SDK 1.5 and built on top of the MSN Search SDK. We use MSN search as our backbone search engine. We ran our experiment on machines running

Fedora 6 with 4 2.8G CPU and 2G RAM.

### Evaluation Queries

We reported preliminary experiment results on numerical queries in [71]. In this paper, we extend our experimental evaluation to a broader range of queries, including queries with textual answers. We consider queries from two sources: the TREC Question Answering Track and a MSN Live Search query log that contains real-user queries that were evaluated by MSN in the Spring of 2006.

**TREC Question Answering Track:** We extracted 42 numerical queries and 100 factoid queries from the TREC Question Answering Tracks from TREC-8 (1999) to TREC 2006. We use key-words to extract numerical (*e.g.*, “length of”, “height of”) and factoid queries (*e.g.*, “who”, “where”) from the TREC QA track. Example numerical queries include: “diameter of the earth”, “average body temperature”, “length of Columbia River”, “height of the tallest red-wood”. Example factoid queries include: “Who is the Speaker of the Lebanese Parliament”, “Who is the manager of Manchester United”, “Where is Merrill Lynch headquartered”, “Where was Hitchcock born”.

We use the TREC original answers to evaluate the quality of our corroborated answers. Note however that we are running our queries on top of a web search engine and therefore use the web as a data corpus rather than the TREC documents. Note also for some TREC queries, more than one answers are recognized as the correct answer (For example, for the query “Where did Jay-Z grow up”, both “brooklyn” and “New York” are correct answers according to TREC judgement). For such queries, we evaluate the corroborated answer with the highest ranked correct answer.

**MSN Live Search Query Log:** We selected 38 numerical queries and 100 factoid queries from the MSN search log using similar patterns as the TREC QA track. We manually pick queries to filter out non-factoid queries (For example, “Who wants to be a millionaire”) and only considered queries which yielded at least one user click on the search engine query result.

### Evaluation Metrics

We report on the following evaluation measures.

- **Percentage of Queries Correctly Answered (PerCorrect):** For the TREC dataset, we compare our top corroborated answers with the original TREC answers. We derive the percentage of queries with correct answers at different rank of our corroborated answer list (top-1 to top-5).
- **Mean Reciprocal Rank (MRR):** The mean reciprocal rank of the first correct answer (MRR) is generally used for evaluating the TREC Question Answering Tasks. If the query does not have a correct answer in our top-5 corroborated answer list, its reciprocal rank is set to 0. We report the MRR values for experiments over the TREC queries.
- **Similarity between User Clicks and Corroborated Answer:** Unlike TREC queries, queries extracted from the MSN query log do not come with an original answer. To evaluate the quality of our corroborated answers, we therefore compare pages that generate top corroborated answers with user clicks.
- **Time Cost:** We report the time needed to return the corroborated answer list. This time cost is divided into retrieval time, which is the time spent accessing the web pages, and corroboration time, which is the time spent by our system for answer extraction and scoring.
- **Number of Pages Retrieved:** As discussed in Section 4.4, we do not retrieve every page from the search result but dynamically stop when we have identified the top-1 corroborated answer. We report on the actual number of web pages needed to reach an answer.

### 4.5.2 TREC Queries

We now report our results on queries from the TREC QA Track. In this section, we first discuss the quality of our answers (Section 4.5.2). In particular, we show the benefit of each individual scoring components on the answer quality (Section 4.5.2). We then compare the performance between our approach and previous question answering techniques (Section 4.5.2). We will show the number of web pages needed to return an answer (Section 4.5.2), and the time cost of our techniques (Section 4.5.2).

**Answer Quality**

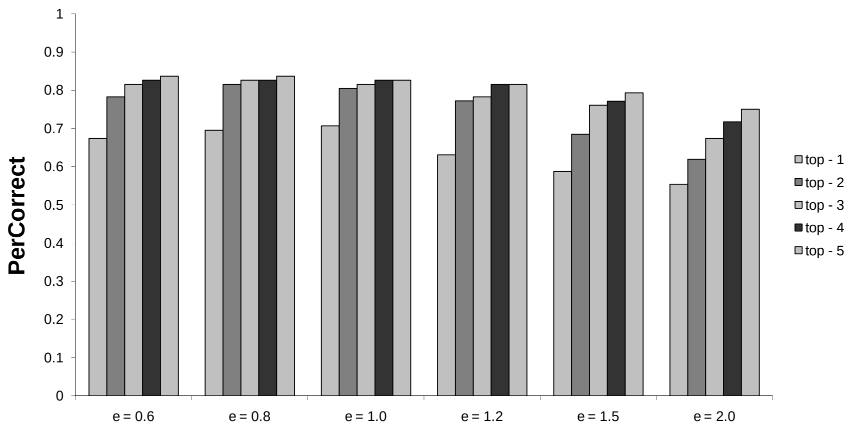


Figure 4.4: Impact of Parameter $e$ on PerCorrect for TREC queries for `CORROB`

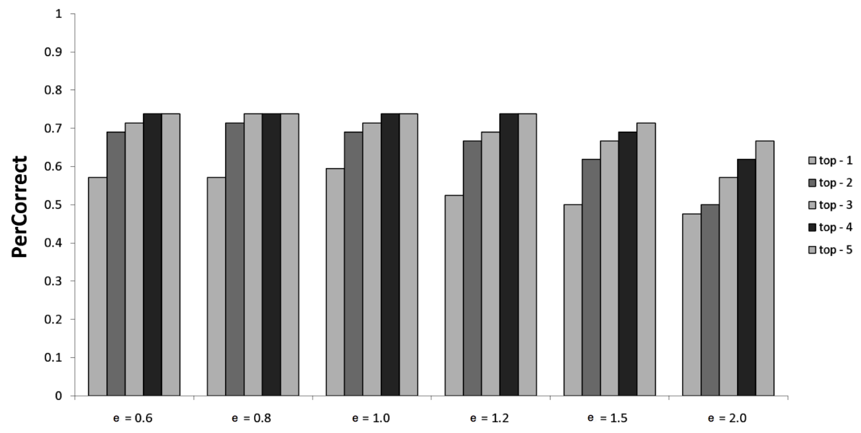


Figure 4.5: Impact of Parameter $e$ on PerCorrect for numerical queries for `CORROB`

We first evaluate the quality of our corroborated answers by analyzing the percentage of correct answers (compared to the TREC-supplied original answers) for the 142 queries we extracted from TREC. Note that as shown in Equation 4.6 there are three parameters which may affect the answer score: $e$, $\beta$ and $maxPage$. In the following we evaluate the effect of each of these parameters.

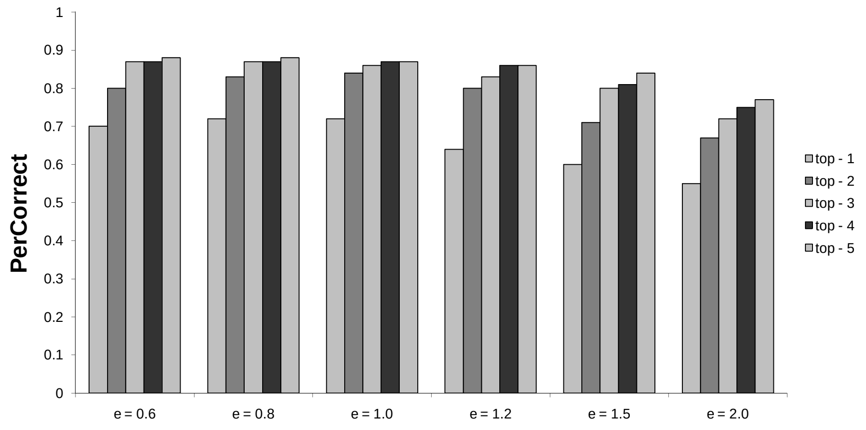


Figure 4.6: Impact of Parameter $e$ on PerCorrect for factoid queries for `CORROB`

We first fix $\beta$ to 0.25 and $maxPage$ to 50 and test the effect of $e$, which is the parameter in the Zipf's distribution. Figure 4.4 shows the PerCorrect values for our top-1 to top-5 corroborated answers for different values of the $e$ parameter of our Zipf corroboration scoring method (`CORROB`) of Section 4.3. Overall, an $e$ value of 1 provides the best results at top-1. Interestingly, the distribution of user click per result position for the subset of the MSN queries that only considers queries expecting a numerical answer has a slope which is not as steep as the one for general queries (Figure 4.2) as these queries yield more clicks, which end up increasing the probability of clicks for positions higher than 1. Using the same curve-fitting techniques as we did in Section 4.3.1 we can approximate the click distribution of these numerical query to a power law distribution with an exponent parameter ($e$) of 1.01 (Figure 4.7(a)). In addition, the curve-fitting yields an exponent parameter of 0.92 for the click distribution of the factoid queries (Figure 4.7(b)), which have the best PerCorrect and MRR values with a $e$ value of 0.8 and 1.0. An $e$ value of 1.0 yields a slightly better PerCorrect value at top-2 answer but slightly worse PerCorrect value at top-5 answer compared with a $e$ value of 0.8. Both of these results validate our choice of Zipf distribution to model the originality of pages.

As we increase $e$, the quality of our corroborated answers drops since top search engine result pages are given the most relevance weight, and fewer pages are considered in the corroboration. Inversely, for lower values of $e$, many pages are considered for corroboration, but their

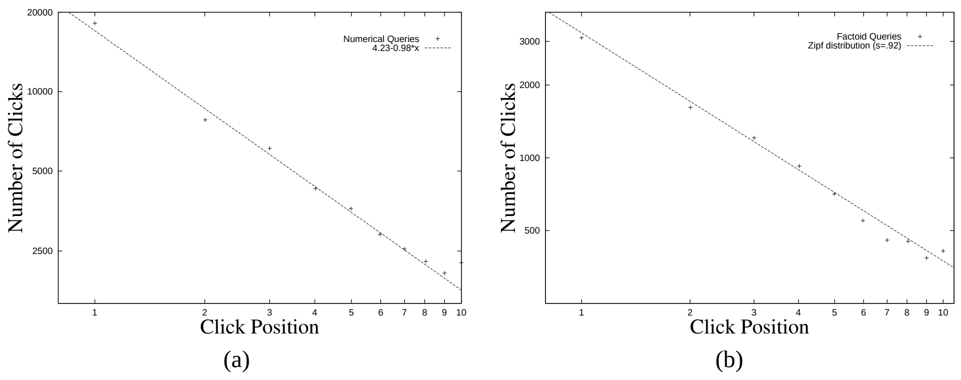


Figure 4.7: The log-log plot for the number of user clicks as a function of the position of the page in the search engine result

relevance scores tend to be similar and answers from high-ranked search engine query result pages are not given more weight.

Figure 4.5 and 4.6 plot the PerCorrect values for numerical and factoid queries respectively. As shown, both results are consistent with the overall PerCorrect values.

| | Combined | Numerical | Factoid |
|---|---|---|---|
| `CORROB` (e = 0.6) | 0.752 | 0.645 | 0.797 |
| `CORROB` (e = 0.8) | 0.765 | 0.651 | **0.813** |
| `CORROB` (e = 1.0) | **0.767** | **0.657** | **0.813** |
| `CORROB` (e = 1.2) | 0.704 | 0.615 | 0.741 |
| `CORROB` (e = 1.5) | 0.664 | 0.586 | 0.697 |
| `CORROB` (e = 2.0) | 0.610 | 0.533 | 0.642 |

Table 4.2: Answer Quality for TREC queries

Table 4.2 reports the MRR value for the `CORROB` method presented in this paper. We list the MRR values for all 142 TREC queries and for the two separate types of queries in each of the three columns. We obtain a MRR score of 0.767 for the `CORROB` method with an $e$ value of 1.0. This means that on average, the correct TREC answer is found within the 2 best corroborated answers. For comparison, the web question answering system presented in [23] has a best MRR score of 0.507; however they report results on a different subset of TREC queries that contains a broad variety of queries, which are possibly harder to answer. In the rest of our experiments, we will use $e = 1$ as our default parameter value.

Another factor that affects the performance of our `CORROB` method (Equation 4.3) is the

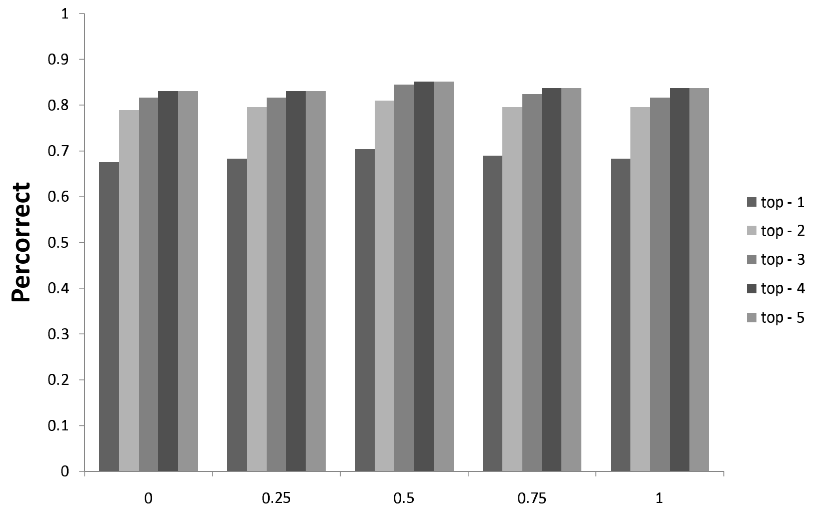


Figure 4.8: Impact of $\beta$ on PerCorrect for TREC queries

| $\beta$ | MRR |
|---|---|
| 0 | 0.766 |
| 0.25 | 0.767 |
| 0.50 | 0.772 |
| 0.75 | 0.756 |
| 1.0 | 0.752 |

Table 4.3: Impact of $\beta$ on MRR for TREC queries

parameter $\beta$ that we use to quantify the decrease of duplicate page score. Figure 4.8 plots the PerCorrect values for the top-1 to top-5 corroborated answers, and Table 4.3 shows the MRR values of the `CORROB` method with a $\beta$ value of 0, 0.25, 0.5, 0.75 and 1 and with a fixed $e$ value of 1.0 and $maxPage$ value of 50. We discussed in Section 4.3.1 that duplicated pages should not have as much weight as page that contains original information but still need to be taken into consideration. The results shown in Figure 4.8 and Table 4.3 confirm our claim. As shown, both PerCorrect and MRR values show improvements compared with the case that no originality is considered ($\beta$ = 0). However, if we completely remove the effect of duplicated pages by setting $\beta$ = 1, the answer quality is not as good as the case for which no originality is considered. Overall, the `CORROB` method has the best answer quality when we set $\beta$ = 0.5. In the following experiment, we set 0.5 as the default value for $\beta$.

As discussed in Section 4.4, limiting the number of pages retrieved using the $maxPage$ parameter may speed up query evaluation, but could affect answer quality as fewer web pages

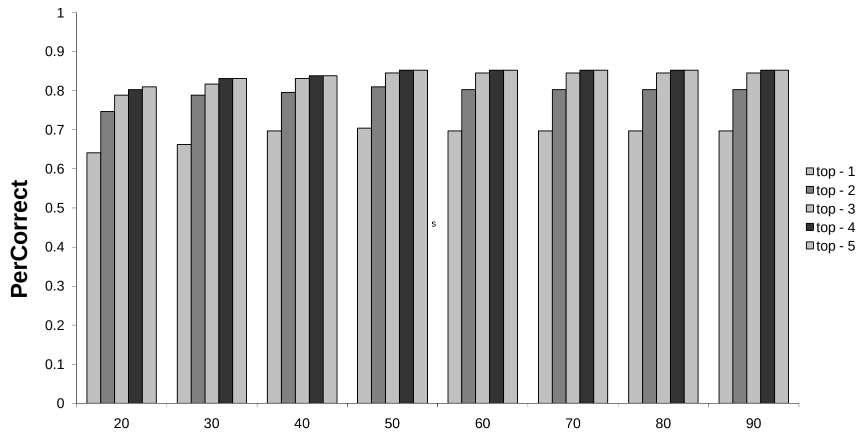


Figure 4.9: Impact of $maxPage$ on PerCorrect for TREC queries

will be used for corroboration. Figure 4.9 shows the impact of different $maxPage$ value (from 20 to 90) on the percentage of correct answers. The answer quality increases as $maxPage$ goes from 20 to 50, but remains the same when more pages are considered. As expected, considering too few pages for corroboration decreases answer quality. Our approach retrieves fewer than 50 pages (on average 34 pages for $e$ = 1, shown in Figure 4.12) to return a corroborated answer, therefore increasing $maxPage$ above 50 has little impact on the quality of the corroborated answers. In addition, high values of $maxPage$ lead to retrieving more pages, resulting in higher query processing times. Therefore, in the rest of our experiments, we set the value of $maxPage$ to 50.

Our CORROB technique finds the correct answers within the top-5 corroborated answers for 85.2% of the 142 queries we extracted from TREC. For 21 of the TREC queries, we were not able to identify the correct answer within the top-5 corroborated answers. An in-depth analysis of these 21 queries shows that 4 of them are multi-answer queries (e.g., "width of Atlantic Ocean") for which our top-5 corroborated answer list contain at least one correct answer (but not the TREC original answer); we were not able to extract web answers for 10 of the queries (e.g., "the first director of the World Food Program"); we answered incorrectly 5 queries; and finally, 2 of the queries have wrong (or possibly outdated) answers within TREC: "Lifetime of hermit crabs," which has a TREC answer of 70 years, and a correct answer of 30 years, and

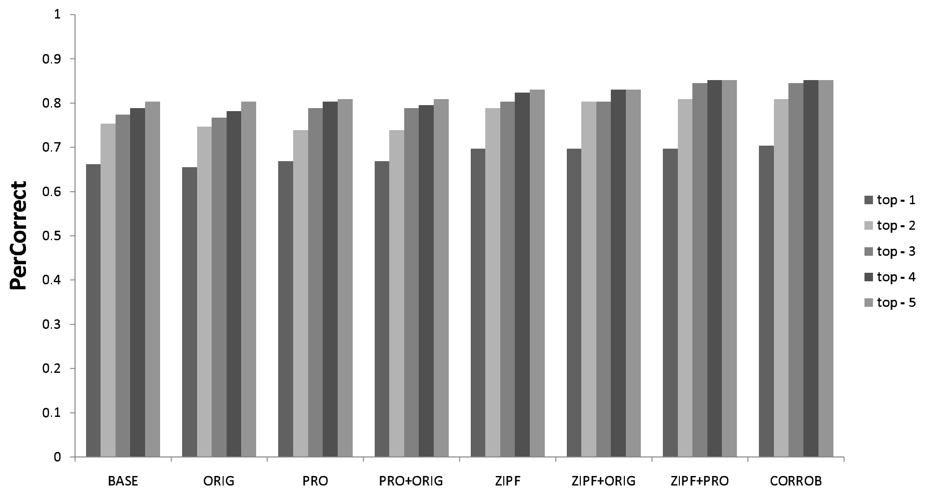


Figure 4.10: Impact of Scoring Components of TREC Queries

"highest recorded temperature in San Antonio, TX," with a TREC answer of 107F and a correct answer of 111F. Out of the 142 queries, 22 of them were time sensitive (For instance, "who is the manager of Manchester United"). However, at the time of our experiments, the correct current answers were the same as the ones provided by the TREC corpus.

## Impact of Scoring Components

In this section, we show the benefit of each individual scoring components we present in Section 4.3. In the following comparison, we denote the baseline `BASE` as the scoring approach in which the score of pages are the same, and the score of each answer is the score of the page divided by the number of answers within the page. We use `ZIPF` to denote the method in which only Zipf's page relevance is considered (Section 4.3.1). We use `ORIG` to denote the baseline approach with the addition of web page originality (Section 4.3.1). We use `PRO` to denote the baseline approach with the addition of answer prominence within web page (Section 4.3.2). The `CORROB` approach, which combines `ZIPF`, `ORIG` and `PRO`, refers to the comprehensive scoring approach, as described in Equation 4.6.

Figure 4.10 shows the PerCorrect values for our top-1 to top-5 corroborated answers, and Table 4.4 reports the MRR values for different combinations of the scoring components. In most cases, the answer quality improves as we incorporate each of the scoring component, the `ZIPF` component provides the best improvement. The only exception is when adding the `ORIG` component, the PerCorrect value at top-1 slightly decreases, but combined with `ZIPF` components, the originality provides a small increase in answer quality. The `CORROB` scoring approach, which combines all three components, has the best result, with an MRR value of 0.772, outperforming the `BASE` case by 6.9%.

| | Combined | Numerical | Factoid |
|---|---|---|---|
| `BASE` | 0.722 | 0.602 | 0.773 |
| `ORIG` | 0.717 | 0.590 | 0.770 |
| `PRO` | 0.727 | 0.610 | 0.776 |
| `ORIG+PRO` | 0.726 | 0.612 | 0.775 |
| `ZIPF` | 0.757 | 0.624 | 0.812 |
| `ZIPF+ORIG` | 0.760 | 0.625 | 0.817 |
| `ZIPF+PRO` | 0.766 | 0.657 | 0.812 |
| `CORROB` | 0.772 | 0.657 | 0.820 |

Table 4.4: Answer Quality of Scoring Components

**Comparison with Existing Question Answering Techniques**

In this section, we compare our approach with existing question answering techniques. Previous works in question answering have been using answer frequency as the evidence of answer quality [43, 23, 78]. Summarizing previous work as frequency-based approach is a reasonable simplification. We could not exactly reproduce the actual question answering techniques of these works because we do not have access to the detail of their implementations, such as their information extraction and query rewriting techniques. Our corroboration approach could be used in conjunction with these tools to further improve answer quality.

In a frequency based approach, the score of an answer is a function of the frequency of the answer among all the pages. In particular, we implement two types of frequency based approaches: page-frequency `(P-FREQ)` and answer-frequency `(A-FREQ)`. In `P-FREQ`, the score of an answer is based on the number of pages from which this answer was extracted. In `A-FREQ`, the score of an answer is based on the number of time the answer was extracted from all pages.

In addition, we also implemented `TOP-PAGE` approach, in which we only extract answers

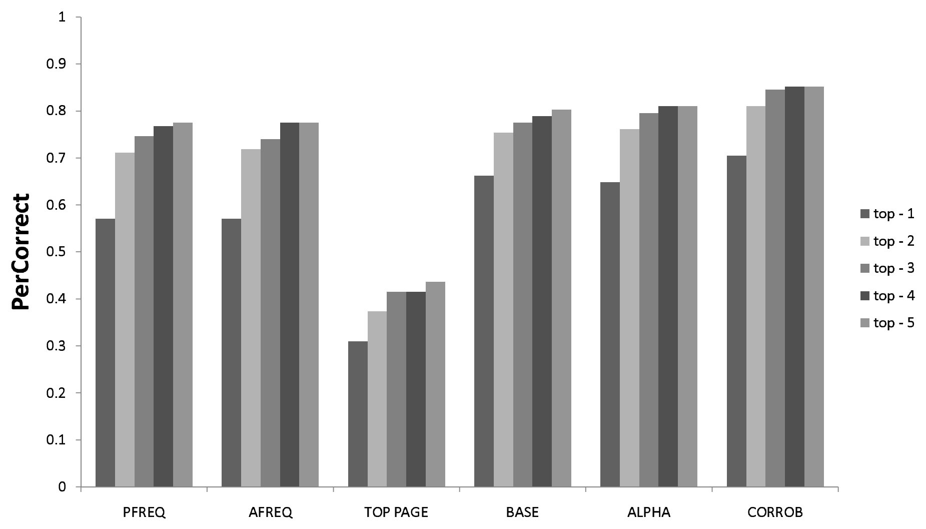


Figure 4.11: PerCorrect Comparison with Existing Question Answering Techniques

from the first page and rank answers based on their frequency (A-FREQ). We tested the performance of `TOP-PAGE` simply based on the observation that most users only look into the first page returned from the search engine.

| | MRR |
|---|---|
| `P-FREQ` | 0.663 |
| `A-FREQ` | 0.664 |
| `TOP-PAGE` | 0.360 |
| `BASE` | 0.722 |
| `ALPHA(`$\alpha$`=0.05)` | 0.722 |
| `CORROB` | 0.772 |

Table 4.5: MRR Comparison with Existing Question Answering Techniques

Figure 4.11 and Table 4.5 shows the PerCorrect and MRR values of the two frequency based approaches, the `TOP-PAGE` approach, the `BASE` approach, the `ALPHA` method [71], and our `CORROB` approach. As shown, `P-FREQ` and `A-FREQ` achieve a PerCorrect value of 0.57 at the top-1 corroborated answer and 0.77 at top-5 corroborated answers. In addition, `P-FREQ` and `A-FREQ` have MRR values of 0.663 and 0.664 respectively, both of which are smaller than the `BASE` approach, showing that even a simple corroboration-based approach outperforms the frequency-based techniques dramatically. We compute the statistical significance of our results using the one-tailed Wilcoxon Signed-Rank test [69]. The difference between `CORROB` and

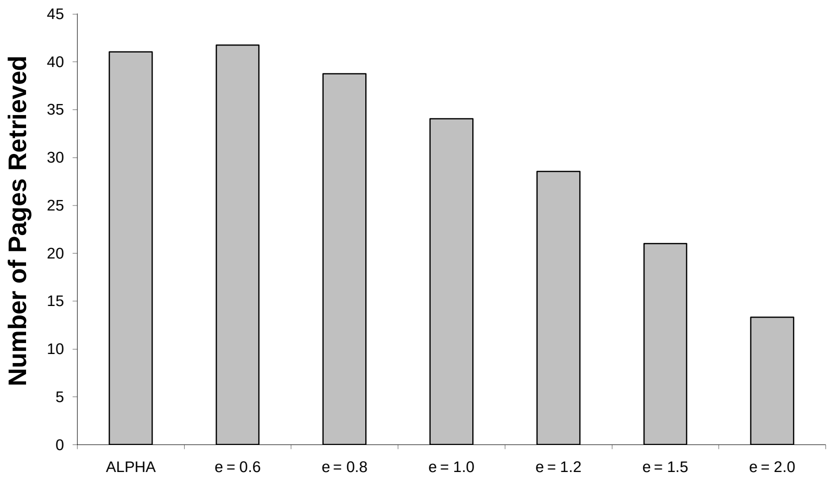


Figure 4.12: Average Number of Pages Retrieved for TREC Queries

A-FREQ and between CORROB and P-FREQ are both statistically significant with a *p-value* of 0.003. These results confirm that a corroboration-based approach outperforms frequency-based approaches. In addition, the advantage between the CORROB method and the BASE approach is statistically significant with a *p-value* of 0.034. The TOP-PAGE approach, performs surprisingly bad compared with all other techniques, with an MRR value of 0.36. This suggests that looking into only the first page from the search engine result is not sufficient for the users to get the correct answer in most cases.

We proposed the corroborative ALPHA method in [71]. ALPHA uses a simpler scoring strategy for the page relevance, dropping the score of a page $p$ based on a parameter $\alpha : s(p) = (1-\alpha)^{r(p)-1}$, where $s(p)$ is the score of page $p$ and $r(p)$ is the rank of the page as returned from the search engine. Our new CORROB strategy outperforms ALPHA, with a *p-value* of 0.011. By using a Zipf's law to model the decrease in page relevance, the CORROB method is able to return higher quality answers than ALPHA.

**Number of Pages Retrieved**

We dynamically retrieve pages as we corroborate answers (Section 4.4). Figure 4.12 shows the average number of pages retrieved when answering queries extracted from TREC for the

ALPHA method and the CORROB technique with the $e$ parameter ranging from 0.6 to 2.0. Increasing the value of $e$ in our CORROB method gives more weight to pages ranked higher in the search engine query result therefore reducing the number of pages needed to identify the top corroborated answers. In contrast, the ALPHA method has the highest average number of page retrieved except for $e = 0.6$.

**Time Cost**

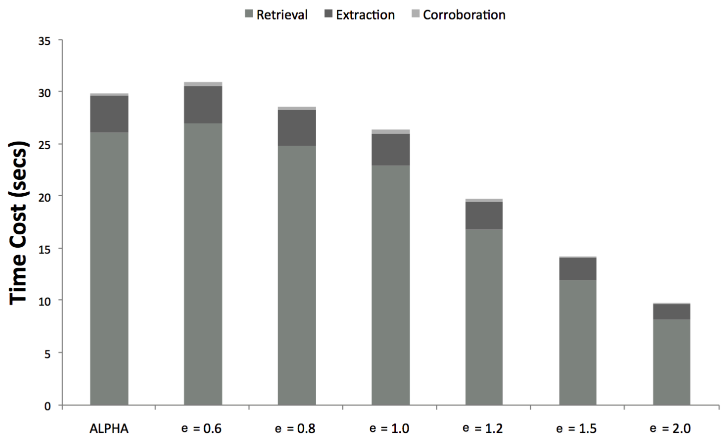


Figure 4.13: Time Cost of different scoring strategies for TREC queries

The query evaluation time for the ALPHA and CORROB corroboration techniques is shown in Figure 4.13. In particular, we divide the time cost into three parts: web page retrieval, answer extraction and answer corroboration. The first part is the time cost for retrieving cached web pages from the search engine server, and the second and third part is the time cost for answer extraction and corroboration, respectively. As expected, based on the number of pages retrieved (Figure 4.12), the CORROB method outperforms ALPHA method for all values of $e$ but $e = 0.6$. The CORROB method takes a bit more time than ALPHA with an $e$ value of 0.6 due to more pages being retrieved.

While the overall query answering cost can be high, most of the time is spent on retrieving pages from the web. Our query evaluation time is reasonable (3 seconds for answer extraction and 0.4 second for corroboration on average for each query), and user may find it acceptable to wait for corroborated answers if it saves them the hassle of manually checking the pages themselves. In addition, our implementation does not have direct access to the search engine indexes and cache but must access these through an interface which incurs web retrieval costs. If our techniques were implemented within a search engine, they would provide much faster query response time.

### 4.5.3 MSN Queries

We now report on experimental results over the queries extracted from the MSN Live Search query log. We set the default values of $e$, $\beta$, $maxPage$ to 1, 0.5, 50 respectively as they provided the best quality results for a reasonable time cost over TREC queries.

#### Comparison with User Clicks

We do not have a list of correct answers to the queries we extracted from the MSN Live Search query log. To evaluate the quality of our answers, we compare user clicks with pages that we used to generate the top corroborated answers. The internet being dynamic, by the time we performed our experiments, much of the information in the log was obsolete. Many pages on which users clicked were not available in the search engine (SE) query result anymore. Among the 331 pages that appeared in user clicks, only 93 pages were still in the search engine result when we run the experiment. In addition, the position of these pages in the search engine result has changed greatly: while they had high positions when the query was issued (with average rank reverse of 0.5), they were ranked much lower in the current search engine result (with average rank reverse of 0.081). Overall, out of 138 queries in the MSN query set, for 81 of them the user clicks are no longer in the current search engine result.

Despite these limitations, we found that our corroborated answers correlate with user clicks. Figure 4.14 shows, for each of the 57 queries that have at least one user click in the current search engine result, the number of user clicks, the number of corresponding pages that were returned in the search engine query result at the time of our experiments, the number of such pages that contained answers to the query, and the number of pages that contained the top corroborated answer. As shown, when a page on which the user clicked appears in the search

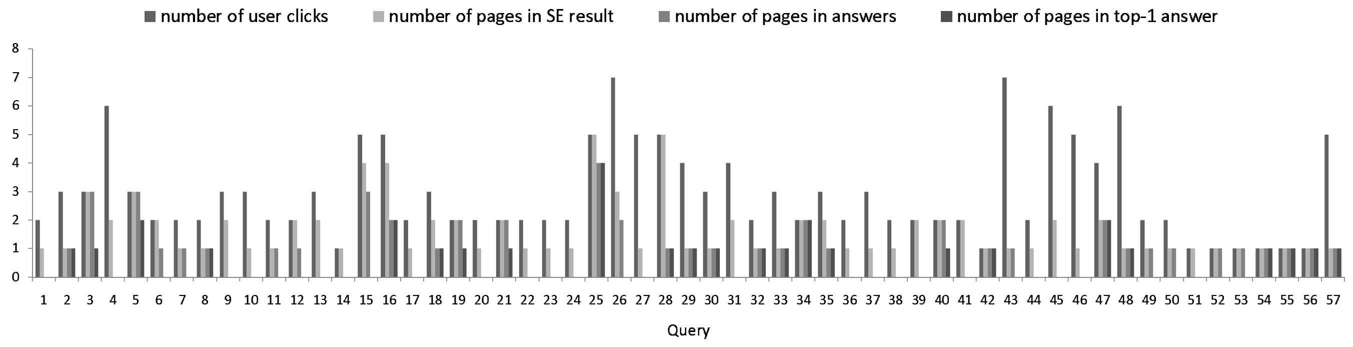


Figure 4.14: Comparison between User Clicks and Corroborative Answers for MSN queries

engine query result, it contains one of the top-5 corroborated answers for 61% of the queries, and the top-1 answer for 42%. The results show that the corroboration approach is good at identifying answers that are of interest to the user, as many user clicks correlate with top corroborated answers. We looked into the remaining 22 queries for which there is no overlap between the user clicks and the pages we use to generate top corroborated answers. We found that for 14 of them we could not extract answers from these user clicks, for 5 of them our corroboration terminated before reaching these user clicks because they were ranked much lower by the current search engine. For only 3 of these queries, our corroborated answers do not include answers extracted from user clicks. We believe that with a more recent search log, our corroboration method would show better similarity with the user clicks.

### Number of Pages Retrieved

We also tested the number of pages retrieved for each MSN query. Our corroboration techniques dynamically retrieve web pages. On average, we retrieve 36.1 pages for each MSN query, which is lower that our 50 $maxPage$ limit. The retrieval stopped dynamically before the $maxPage$ value for 88% of the queries tested; in some cases fewer than 10 pages were enough to identify the best corroborated answer.

**Time Cost**

Figure 4.15, 4.16 and 4.17 shows the time cost for MSN queries that have at least one click in the current SE result and that have no clicks in the current SE result, respectively (In order to display properly, we break the figure for the queries that have no clicks in the current SE result into two subfigures). As with TREC queries, the retrieving time is the dominant factor in the total time cost.

We also compare this time cost with the time the user spent on browsing the web pages. The MSN query log comes with an accurate timestamp for both the time when the query was issued by the user and the time when the user clicked on the web page. We calculate the user time cost as the duration between the time the query was issued and the time of the last user click. Of course, this is an approximation of the time the user took to find an answer to the query as it does not take into account the time the user spent loading and reading the corresponding pages nor does it guarantees that the user found an answer within the clicked pages. On average, the user time cost (107.3 seconds) is about 4.1 times more than the cost of our corroboration approach (26.4 seconds). This indicates that our corroboration method is efficient in identifying the answers and can save users a great amount of time. Figure 4.15, 4.16 and 4.17 show that we do have a few queries for which our corroboration techniques take longer to return answers than the user time cost (*e.g.*, Query 38, 39 in Figure 4.15; Query 21, 35 in Figure 4.16 and Query 41, 70, 76 in Figure 4.17). For these queries our corroboration needs to retrieve more pages while the corresponding number of user clicks is relatively small. On average, the number of pages retrieved for MSN queries set is 36.1 and the number of user clicks for each query is 2.4. For the queries that our corroboration techniques take longer than the user time cost, the average number of pages retrieved is 38.3 and the average number of user clicks is 1.7. Typically, the queries that require expensive corroboration times are the ones for which multiple correct answers exist. For instance, for the query “who invented the television”, both “Farnsworth” and “Zworykin” are correct answers. The existence of multiple correct answers leads our corroboration techniques to go deeper into the search engine results to find the top-1 answer, therefore resulting in more pages being retrieved.

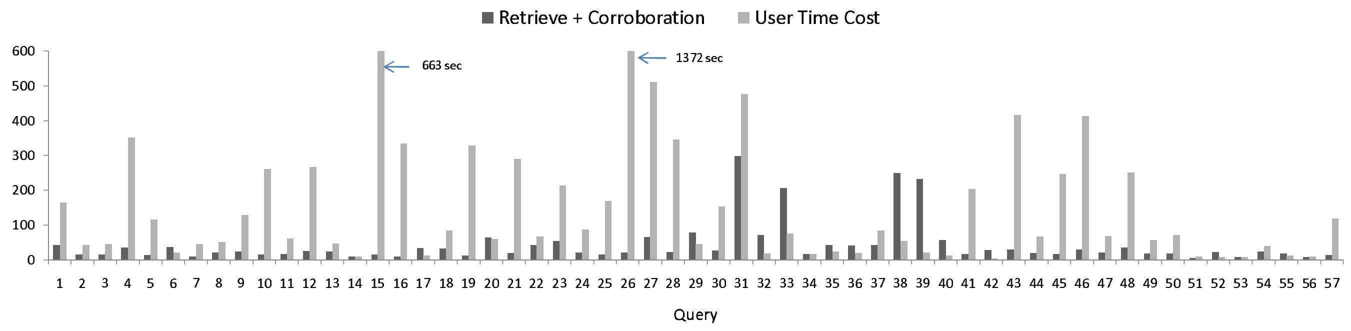


Figure 4.15: Time Cost for MSN queries that have at least one click in the current SE result

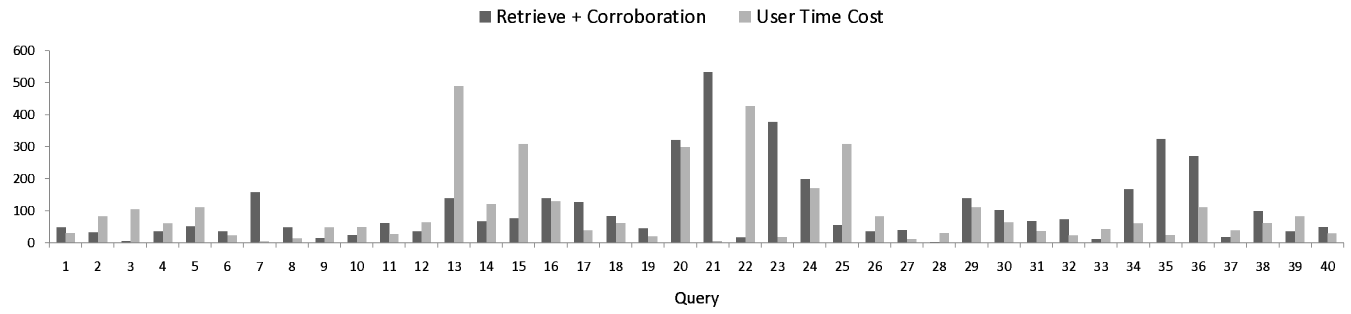


Figure 4.16: Time Cost for MSN queries that have no clicks in the current SE result (Part 1)

## 4.6 Conclusions

In this chapter, we presented an approach to corroborate answers from the web to improve the accuracy for the question answering task. Our techniques use information extraction methods to identify relevant answers from a search engine result. We assign scores to each answer based on the frequency of the answer in the web search engine result, the relevance and originality of the pages reporting the answer, as well as the prominence of the answer within the pages. Our experimental evaluation on queries extracted from the TREC Question Answering Track shows that a corroboration-based approach yields good quality answers. In addition, by comparing our approach to user-click behavior on a sample of queries from a MSN query log, we show that our techniques result in faster answers as they prevent users from having to manually check several sources. The work presented in this chapter was published in [71] and [72].

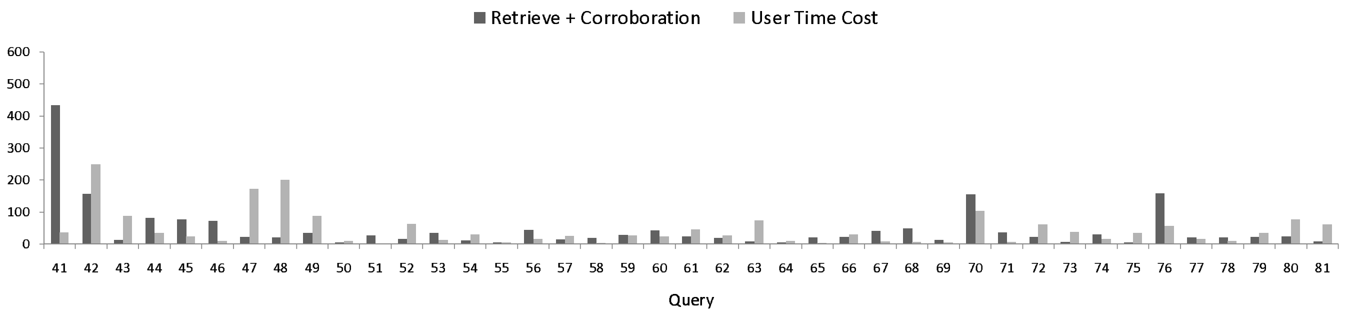


Figure 4.17: Time Cost for MSN queries that have no clicks in the current SE result (Part 2)

Our work on corroborating answers was first published in 2007, during which time the mainstream technology in information corroboration was frequency based approaches. By using a meaningful scoring function for the extract answers, our techniques were among the first to consider the trustworthiness of the sources and stand out from existing methods. Our results have motivated works that were proposed since the publishing of [71] in 2007 not only in information corroboration [17, 18, 5, 82] but also in other topics such as entity search [11] and attributes extraction [3, 61]. In particular, Banerjee et al. [5] studied the Quantity Consensus Queries (QCQs) whose answer is a tight quantity interval extracted from thousands of snippets. QCQ queries present a unique challenge in that it considers the closeness among numerical answers and existing methods including ours that only consider exact matches would be unable to combine the scores of close but not identical quantities. The authors proposed two algorithms that learn to score and rank quantity intervals by leveraging the snippet quantity and snippet text information. The experiments show significant improvement over methods that consider fixed numerical answers but not quantity intervals.

Over the past few years, research in information corroboration [79, 40, 28, 49, 54, 81, 29, 55] has shifted towards more principled ways of evaluating the trustworthiness of the sources (see a survey in [46]), as opposed to using external metadata such as the rankings from the search engine . Such methods usually work by iteratively estimating the trustworthiness of the sources and the probabilities of the answers until convergence is reached. The computation of the scores of the sources and the answers is based on probability theory or learned from graph theories. We present a case study of such algorithms in Chapter 5.

# Chapter 5

# Corroborating Affirmative Statements from Unreliable Sources

## 5.1 Introduction

In the previous chapter, we discussed how corroboration could help the task of question answering. It is evident that by leveraging the quality of the sources as well as the prominence of the answers within the sources, it is possible to significantly improve both the precision and recall of the answers. Moreover, as we observe in Table 4.4 in Chapter 4, the quality of the sources is a bigger contributor in improving the answer quality (as `Zipf` is the component that provided the biggest boost to both PerCorrect and MRR). In many cases, it is possible to infer the quality of the sources by using some prior knowledge. For instance, we leveraged the rank of the sources in the search engine result and approximated the quality of the sources using the Zipf distribution. Unfortunately, in many other applications, such information regarding the sources might not be readily available and we need to explore other avenues of techniques in order to carry out corroboration.

The problem of lacking prior knowledge of the sources have been significantly relieved thanks to the recent advance in corroboration techniques (see a survey at [21]). The key idea of those methods is to iteratively estimate the probability of the answers and the trustworthiness of the sources until a convergence is reached (*i.e.*, the scores stabilize). Of all the methods proposed, the intuition is that they rely on the conflicting answers to differentiate the trustworthiness of the sources. Sources that provide more correct answers (computed by the algorithms) is more trustworthy than sources with more incorrect answers.

Unfortunately, although such an intuition sounds reasonable and indeed work effectively, it does not bode well in a scenario where there are little conflicting answers. For each query, a lack of conflicting answers means that all the sources return the same answer. Intuitively, it looks as if no corroboration is needed since there is no suggestion of an alternative answer.

However, this is not always the case, as illustrated in the following example.

EXAMPLE 1: *Consider we want to identify a list of restaurants that are up and running in a certain region. There exist several web sources that provide valuable information for this task. For instance, local search engines such as* `Yellowpages` *and* `Citysearch` *provide business listings including restaurants. Social web sites such as* `Yelp` *and* `Foursquare` *allow users to check-in at dining venues. Most of the restaurant listings on these web sources are hints that the restaurants exist (except for those listed as 'CLOSED'). However, the fact that a restaurant is listed at one or several of these web sources is not definite evidence that it is still open. As an example, consider a restaurant named 'Danny's Grand Sea Palace' located at '346 West 46th St, New York', which is backed by both* `Yellowpages` *and* `Citysearch`*. A follow-up check*[1] *revealed that the restaurant is no longer in business and that the listing was inaccurate.*

In this chapter, we investigate the corroboration problem of identifying the veracity of facts in the presence of mostly affirmative statements. A fact is either true or false, and an affirmative statement indicates support from a source that the fact is true. Intuitively, for a fact with only affirmative statements, there should be no ambiguity that it is true, since there is no suggestion from any source that the fact may be false. However, as we see in Example 1, this is not necessarily the case. Although there are two affirmative statements for the fact *'Danny's Grand Sea Palace is open'* , it is still factually false. In addition to the above example, we also observe similar cases in other domains. For instance, technology blogs usually provide claims regarding major product releases, each of which could be viewed as facts with only supportive statements.

Note that in Chapter 3, we defined our corroboration model as identifying the correct answer given a set of candidate answers for a query. For the ease of discussion, we in this chapter use *facts* as the object for which we want to corroborate instead of answers. In fact, a *fact* considered in this chapter can be viewed as a *query* with 2 candidate answers $\{$*true, false*$\}$. Our objective is to estimate for each fact its correct value (*i.e., true* or *false*). Note also that by modeling the corroboration problem this way, we are not limiting the applicability of our technique to problems with binary candidate answers. For a query with multiple candidate answers, our

[1] *The restaurant is one of a set of restaurants that we checked in person.*

model can be adapted by considering each candidate answer as a fact. For instance, consider a query $q$ with a set of candidate answers $\{a_1, a_2, \ldots, \}$, our model can handle such cases by creating a fact $f_i$ for each answer $a_i$ where $f_i$ is a new query '$a_i$ is the correct answer for $q$'.

The main difficulty is to correctly identify false facts, since facts with only affirmative statements appear to be true. In principle, a false fact could be revealed if its affirmative statements are from sources with low trustworthiness. Unfortunately, in a scenario where most facts have affirmative statements only, it is hard to compute the correct trustworthiness of the sources. We listed below the challenges of the problem we focus in this chapter.

- **Quality of sources** Information on the Internet is fast changing and goes out-dated fast. For a certain task, there might not exist a source that is fresh and yet with good coverage. A serious website such as `Yelp` which allow users to post authentic reviews contains erroneous restaurant listings. This is different from applications for which existing corroboration techniques have been successful. For instance [81], `imdb` is a near-perfect source for the *movie* dataset. Assessing the quality of the source is critical to derive the correctness of the facts it reports.

- **Apparent consensus** The principle of corroboration is to differentiate the sources' quality, hence treating the information from each source differently. Existing corroboration techniques work well in tasks with conflicting statements because they can dampen the trust scores for the sources that have incorrect statements. Unfortunately, this is not the case in our scenario since sources provide mostly same statements (*i.e.*, affirmative statements). As a result, it is extremely difficult to identify any errors from the sources.

To tackle these challenges, we propose a novel corroboration algorithm that uses a multi-value trust score for the sources. Each fact is evaluated using one of the trust values from each source. Unlike with a single trust score for the sources where all facts would have the same corroboration result in a scenario where most or all facts have only affirmative statements, we can correctly identify false facts by considering a lower trust score for the sources reporting them. Intuitively, a source may have different trustworthiness on different sets of facts, and our algorithm leverages such observation to improve corroboration quality. We summarize our contributions as follows.

- We investigate the problem of corroborating facts in the presence of mostly affirmative statements and demonstrate the limitations of state-of-the-art methods.
- We propose a novel corroboration method that adopts a multi-value trust score for each source; each fact is evaluated using one set of source trust values.
- Our corroboration algorithm incrementally selects facts by considering the information entropy in the unprocessed facts and updates the trust scores for the sources.
- We conduct experiments over synthetic and real world datasets and show that our algorithm significantly outperforms existing approach on precision and accuracy.

To the best of our knowledge, our corroboration algorithm is the first to consider different trust scores from the same source for different sets of facts. In the following discussion, we show that a multi-value trust score is not only effective, but necessary in the corroboration problem considered in this chapter. The rest of the chapter is organized as follows. A detailed motivating example is shown in Section 5.2. We formally define the corroboration problem in Section 5.3 and introduce the multi-value trust score strategy in Section 5.4. We present our incremental algorithm in Section 5.5. Experiment results are shown in Section 5.6. Finally, we conclude the chapter in Section 5.7.

## 5.2 A Motivating Example

We use an instance of Example 1 as the motivating example to illustrate the limitation of existing methods. Consider a scenario with 5 sources $\{s_1, s_2, s_3, s_4, s_5\}$ and 12 restaurant listings $\{r_1, ..., r_{12}\}$. For the ease of discussion, we use `T` and `F` votes to refer to affirmative and disagreeing statements. The votes from the sources are shown in Table 5.1, where in the last column we list if each restaurant is actually open (ground truth). A source can vote either *for* (`T`) (*e.g.,* by listing the restaurant) or *against* (`F`) a restaurant (*e.g.,* by listing the restaurant as `CLOSED`). A '-' indicates that a source does not list the restaurant. As shown, all the sources cast votes only for a subset of restaurants. In addition, most restaurants (except for $r_6$ and $r_{12}$) receive `T` votes only. If we know the correct result for each restaurant (as shown in the last column) a priori, it could be computed that the global trust scores for all the sources are $\{$1, 0.8, 1, 0.5, 0.625}, respectively. In the following, we examine the performance of 2 state-of-the-art corroboration techniques.

| | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | *correct value* |
|---|---|---|---|---|---|---|
| $r_1$ | - | T | - | T | - | true |
| $r_2$ | T | T | - | T | T | true |
| $r_3$ | T | - | T | - | T | true |
| $r_4$ | - | - | - | T | T | false |
| $r_5$ | - | T | - | T | T | false |
| $r_6$ | - | - | F | T | - | false |
| $r_7$ | T | - | - | T | - | true |
| $r_8$ | - | T | - | T | T | true |
| $r_9$ | - | - | T | - | T | true |
| $r_{10}$ | - | - | - | T | T | false |
| $r_{11}$ | - | - | T | T | T | true |
| $r_{12}$ | - | F | F | T | - | false |

Table 5.1: A scenario with 5 sources and 12 restaurants

### 5.2.1 The TwoEstimate Algorithm

Galland et al. [28] introduced a set of iterative algorithms that are very related to our corroboration task. Among those, the `TwoEstimate` algorithm is directly applicable to our scenario[2]. The `TwoEstimate` works by iteratively estimating the probability that each restaurant is open and the trustworthiness of the sources until convergence is reached. A direct application of the `TwoEstimate` algorithm on the motivating example yields a result of *true* for all the restaurants except for $r_{12}$, and a trust score of {1, 1, 0.8, 0.9, 1} for the 5 sources, respectively.

Although the `TwoEstimate` algorithm has a recall of 1, the precision and accuracy are only 0.64 and 0.67 respectively. The reason for the result can be explained as follows. First, since the majority of the restaurants only have `T` votes, the only possible corroboration outcome for restaurants other than $r_6$ and $r_{12}$ is *true*. In addition, consider $r_6$ with a `T` vote from $s_4$ and an `F` vote from $s_3$. Although there is one `F` vote, the `T` vote is from $s_4$ which has more correct votes for other restaurants than $s_3$. In a sense, the `F` vote is 'outvoted' by the `T` vote since $s_4$ has a higher trust score than $s_3$. Second, in order to guarantee convergence, the `TwoEstimate`

[2]Note that although the `ThreeEstimate` algorithm has shown better performance, it calculates a measure using the number of `T` and `F` votes for each fact. Since for most restaurants there are `T` votes only, the `ThreeEstimate` algorithm essentially simplifies to the `TwoEstimate` algorithm in this scenario.

normalizes the probability of a restaurant or the trustworthiness of a source to 1 if it is greater than or equal to 0.5. This normalization process essentially translates a restaurant with uncertainty into an absolute `T` or `F` and then uses it as feedback for the calculation of its sources. The effect of this *reinforcement* mechanism is greatly amplified in our scenario since, there is little conflict for the vast majority of the restaurants and consequently, sources receive a near-perfect trust score.

### 5.2.2 The BayesEstimate Algorithm

Zhao et al. [81] proposed a Bayesian probabilistic graphical model (termed as `BayesEstimate`) that infer true facts and source quality. Instead of using a single value for the trustworthiness, the `BayesEstimate` algorithm leverages two-sided errors (number of false positive and false negative) of each source. In essence, the `BayesEstimate` algorithm is tailored for real world corroboration tasks in which the algorithm has some prior knowledge about the source quality (*e.g.*, high precision but low recall). In our scenario, although we do have some sources that match such profile, we have other sources with relatively poor precision (*e.g.*, $s_4$). In addition, the `BayesEstimate` algorithm also suffers from the fact that there is little conflict for most of the restaurants, and hence has similar corroboration result as the `IterEstimate` algorithm. Using the `BayesEstimate` algorithm we obtain a result of *true* for all restaurants, which translates to a precision of 0.58 and recall of 1. The reason that `BayesEstimate` did not identify $r_{12}$ as false is because it considers a high-precision-low-recall prior, and therefore giving `F` vote less weight.

### 5.2.3 Our strategy

Consider a simplified version of our strategy, which does not apply to all the restaurants at once. Instead, it divides the corroboration task into 3 sub-tasks that are carried out in 3 rounds, as shown in Figure 5.1. We start our algorithm with a default trust value (*e.g.*, 0.9) for each source and pick restaurants $r_9$ and $r_{12}$ to process. By using the default trust scores, our method computes a corroborated result of `true` and `false` for the restaurants, respectively. In addition, the trust scores for the sources are then computed as $\{$-, 1, 1, 0, 1$\}$. During the second round, we choose $\{r_5, r_6\}$ (the shaded objects in Figure 5.1 refer to the restaurants which have

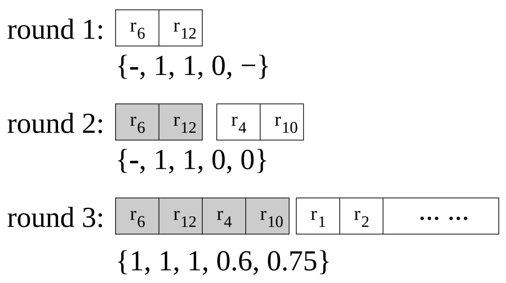


Figure 5.1: Illustration of our strategy

been evaluated), which results in `false` for the two restaurants. Note that although we have `T` votes from $s_4$ for both restaurants, since it has a trust score of 0 from the first round, the corroboration assigns a low score for both restaurants. The trust scores for the sources are updated to $\{$0, 1, 1, 0, 1$\}$. During the last round, the corroboration is applied to the rest of the restaurants and results in *true* for all the remaining restaurants due to the fact that each restaurant is backed by at least one of the "*good*" sources ($s_2, s_3, s_5$). Overall, the sources have a trust score of $\{$0.67, 1, 1, 0.7, 1$\}$ respectively and the corroboration results in a precision of 0.78 and a recall of 1.

The rationale behind considering a multi-round corroboration strategy can be explained as follows. In order to identify as many corrupt listings (restaurants that are no longer open or are not at the address) as possible, we need to have sources with low trust scores. However, in a scenario where all sources are generally good (*i.e.*, with a trust score above 0.5), it is impossible to find bad listings. By applying corroboration in a step-by-step fashion, we are able to obtain a low trust score for some sources over a subset of restaurants. For instance, the above strategy calculates a trust score of 0 for $s_4$ over $\{r_6, r_{12}\}$. During the second round, it aggressively selects all listings that are projected to be corrupt based on the current trust scores of the sources ($\{r_4, r_{10}\}$). In the third round, since all remaining restaurants are projected to be valid, it processes all of them and finish corroboration.

Table 5.2 lists the results of the three methods described above on Example 1, where it shows that our strategy is advantageous compared with two state-of-the-art algorithms. In the following sections, we formally define the corroboration problem (Section 5.3) and present our algorithm with a more sophisticated strategy (Section 5.4 and 5.5) that seeks to further improve

| | Precision | Recall | Accuracy |
|---|---|---|---|
| `TwoEstimate` | 0.64 | 1 | 0.67 |
| `BayesEstimate` | 0.58 | 1 | 0.58 |
| `Our strategy` | 0.88 | 1 | 0.92 |

Table 5.2: Results of the strategies

the corroboration quality.

## 5.3 Problem Formulation

We consider a problem that consists a set of sources $\mathcal{S} = \{s_1, s_2, \cdots\}$ and a set of facts $\mathcal{F} = \{f_1, f_2, \cdots\}$. A fact could be either *true* (meaning it is correct) or *false* (erroneous). For instance, a fact could be "*A restaurant called 'M Bar' located at 12 W 44th St is a legitimate restaurant*".

### 5.3.1 Sources

We consider a source $s \in \mathcal{S}$ as a real-world object that expresses opinions about facts. A source may agree or disagree with a fact in the form of casting a true (`T`) vote or false (`F`) vote. For instance, a source may disagree with the legitimacy of a restaurant by listing it as `CLOSED`. We use $s(f)$ to denote the vote of source $s$ over a fact $f$, illustrated below.

$$s(f) = \begin{cases} T, & \text{if } s \text{ agrees with } f \\ F, & \text{if } s \text{ disagrees with } f \\ -, & \text{if } s \text{ has no knowledge about } f \end{cases} \tag{5.1}$$

Note that in certain scenarios, an unknown vote (*i.e.*, the '-' vote) from a source may slightly indicate that it disagrees with the fact (*e.g.*, a source may delete a restaurant listing after it went out of business), we cannot differentiate such cases from cases where the source has no knowledge about the object.

We associate with each source $s$ a trustworthiness score $\sigma(s)$ which represents its precision. The trust score is a real number between 0 and 1, with 1 indicating a perfect source and 0 indicating a completely wrong source. We define sources with a trust score $\sigma(s)$ between 0.5 and 1 as *positive* sources. In principle, positive sources are the sources that have more correct votes than incorrect ones. Similarly, a *negative* source is defined as a source with a trust score between 0 and 0.5.

### 5.3.2 Facts

A fact $f \in \mathcal{F}$ is an expression over a real-world object that is of interest to the users. A fact is either true or false. In order to estimate the correct value (*i.e.*, true or false) of a fact $f$, we propose techniques to compute a probability $\sigma(f)$ which represents the likelihood that $f$ is true. A fact with a probability of 1 (or 0) is a true (or false) fact. A corroboration algorithm determines the value of a fact $f$ if $\sigma(f)$ is greater than a certain threshold. In this chapter, we use 0.5 as the threshold value, shown below.

$$f = \begin{cases} true, & \text{if } \sigma(f) \geq 0.5 \\ false, & \text{if } \sigma(f) < 0.5 \end{cases} \tag{5.2}$$

**Entropy of unknown facts**: In information theory [14], the entropy is a measure of uncertainty of a random variable. Since we consider $\sigma(f)$ the probability of a fact $f$ being true, we can calculate the entropy $H(f)$ of the unknown fact $f$ as follows.

$$H(f) = -\sigma(f) \cdot \log \sigma(f) - (1 - \sigma(f)) \cdot \log(1 - \sigma(f) \tag{5.3}$$

It is easy to see that a fact $f$ has an entropy of 0 if its probability is 1 or 0 (*i.e.*, no uncertainty) and has the highest entropy of 1 if its probability of 0.5. Intuitively, we expect a fact to have an entropy between 0 and 1 given the votes from the sources. We discuss how to iteratively select facts by leveraging the fact entropy in the following section.

### 5.3.3 The corroboration problem

Given a set of sources $\mathcal{S}$ and a set of facts $\mathcal{F}$, the corroboration problem is to identify the correct value of each fact and estimate the trustworthiness of each source. In this chapter, we focus on a corroboration problem in a specific scenario in which most facts in $\mathcal{F}$ only receive affirmative statements. More formally, let $\mathcal{F}^*$ be a subset of $\mathcal{F}$ such that for each fact $f \in \mathcal{F}^*$ there are only `T` votes only. We focus on a corroboration problem in which most facts in $\mathcal{F}$ are in $\mathcal{F}^*$ (*i.e.*, $|\mathcal{F}^*| \gg |\mathcal{F} - \mathcal{F}^*|$).

## 5.4 Trust Scores of Sources

Our corroboration algorithm is built upon the concept of a multi-value trust score for each source. In the following discussions, we first formally define the single-value trust score and multi-value trust score (Section 5.4.1). We then demonstrate the limitation of an algorithm using a single-value trust score (Section 5.4.2). We finally present our method of implementing multi-value trust scores (Section 5.4.3).

### 5.4.1 Definition

Traditional corroboration techniques [72, 49, 28, 37, 81, 78] usually calculate a trust score for each source that is used to evaluate facts. In such a setting, the same trust score of each source is used to evaluate *every* fact. There are exceptions in which a technique may consider more than one trust score for each source. For instance, the `BayesEstimate` method considers a two-sided errors for each source which capture both the false positive and false negative rates. However, the same measures for a source are used to evaluate *every* fact for which the source casts a vote.

Formally, we define a *single-value trust score* that is used in existing techniques as *one* measure $\sigma(s)$ that is computed for each source $s$. The measure $\sigma(s)$ is used to evaluate each fact $\{f|s(f) \in \{T, F\}\}$ that $s$ casts vote. Note that such measure $\sigma(s)$ could contain more than one value (*e.g.*, `BayesEstimate`). In this following, we focus our discussion on the case where $\sigma(s)$ is a single value.

In contrast, we propose to use a *multi-value trust score* in our corroboration algorithm. A *multi-value trust score* is defined as a group of values assigned to each source, as shown below.

$$\boldsymbol{\sigma}(s) =< \sigma_1(s), \sigma_2(s), \cdots, > \tag{5.4}$$

where we call $\sigma_i(s)$ one of the trust values of source $s$. In such a setting, each fact $f$ is evaluated using one of the trust values of $\boldsymbol{\sigma}(s)$ of sources that have voted for $f$. Consider 2 facts $f_1, f_2$

for which a source $s$ has a `T` vote. A single-value based algorithm evaluates both facts use the same measure $\sigma(s)$, while a multi-value based algorithm may use different trust values of $\boldsymbol{\sigma}(s)$.

As an example, assume we adopt the scoring used in the `TwoEstimate` [28] to compute the probability for facts. Formally, let $f_i \in \mathcal{F}^*$ and $\mathcal{S}_i^+$ be the set of sources that have a `T` vote for $f$. We use $p(\boldsymbol{\sigma}(s))$ to denote the function that picks a trust value from $\boldsymbol{\sigma}(s)$. A multi-value trust score based algorithm computes the probability of $f$ as follows.

$$\sigma(f_i) = \frac{\sum_{s \in \mathcal{S}_i^+} p(\boldsymbol{\sigma}(s))}{|\mathcal{S}_i^+|} \tag{5.5}$$

### 5.4.2 Single-value trust score

A single-value trust score based algorithm works by iteratively estimate the probability of facts and the trust score for the sources. As shown in Equation 5.7, the probability of facts is calculated using the trust score for the sources from the previous iteration (the `Corrob` method). In return, the trust score for the sources is updated using the probability of the facts (the `Update` method). The algorithm terminates once convergence is reached.

$$\sigma^{(k)}(f_i) = Corrob(\sigma(\mathcal{S}_i^{(k-1)})) \tag{5.6}$$

$$\sigma^{(k)}(s_i) = Update(\sigma^{(k)}(\mathcal{F}_i)) \tag{5.7}$$

In a regular corroboration task in which there exist conflicting votes (*i.e.*, both `T`s and `F`s), a single-value trust score often works because incorrect votes can be identified based on the corroborated result of each fact. For instance, consider a fact $f$ with a `T` vote from $s_1, s_2$ and an `F` vote from $s_3$. Assume that the right result (*true*) for $f$ has been derived by the corroboration algorithm. Since $s_3$ has the incorrect vote, its trust score is discounted and is reflected in the corroboration of other facts. However, in a scenario where most facts have `T` votes only, it is difficult to identify any incorrect votes.

Let $A$ be an iterative corroboration algorithm using a single-value trust score. Let us simulate the procedure of $A$ and explain that after applying $A$, all the facts $f \in \mathcal{F}^*$ have the same corroboration result and all the sources have near perfect (or completely wrong) trust scores. Suppose the algorithm starts with an initial trust score $\lambda$ (*e.g.*, 0.9) for each source. $A$ computes the probability for each fact using the $Corrob()$ operation. Since for each $f \in \mathcal{F}^*$ there are

$\mathsf{T}$ votes only and sources have a high initial trust score, facts in $\mathcal{F}^*$ receive a high probability. This is based on the assumption that if a fact is vouched by a number of accurate sources, it is likely to be true. In return, $A$ then updates the trust score for each source based on the calculated probabilities of the facts. Recall that most of the facts are in $\mathcal{F}^*$, *i.e.*, $|\mathcal{F}^*| \gg |\mathcal{F} - \mathcal{F}^*|$. Therefore the computation of $\sigma(s)$ is dominated by the probabilities of facts in $\mathcal{F}^*$. Since $A$ considers that each source has the correct vote for each fact $f \in \mathcal{F}^*$, it assigns a high trust score to each source. This is based on the assumption that the more correct votes a source has, the more trustworthy it is. In addition, in order to avoid converging to a local optima (*i.e.*, all sources have a trust score of 0.5), a common fix is to use a normalization process that converts the probability to 1 (or 0) if it is above or equal to (or less than) 0.5. Once $A$ converges, it results in $true$ for each fact $f \in \mathcal{F}^*$ and a trust score close to 1 for each source.

From the information entropy perspective, such result indicates that the entropy of all unknown facts is 0 since for each fact $f$ the probability is 1, and therefore we have $H(f)$ = 0. In other words, a single-value based method dismisses the uncertainty of facts and considers them true with a probability of 1. This result is counter-intuitive as we expect that in real life scenarios, each source has a trust score between 0 and 1, and each fact has a level of uncertainty quantified by its entropy $H(f)$.

### 5.4.3 Multi-value trust score

Since a method that uses a single measure to evaluate all facts does not work well for our scenario, we now resort to a strategy that processes unknown facts separately. However, we have to address two fundamental challenges: 1) how do we calculate the trust values for each source; and 2) for each fact, how do we select the trust values from each source (*i.e.*, the $p(\boldsymbol{\sigma}(s))$ function) to compute its correct value.

Both challenges above can be addressed by incrementally evaluating facts and updating the trust score for the sources. We repeatedly select a subset of the facts to process and update the trust value that represents the trust score over the facts that have been evaluated. During each round, we use the latest trust value for the sources and leverage heuristics in selecting unevaluated facts. We formally define the incrementally calculated trust score as follows.

**Definition 1 (Incrementally calculated trust score)** *Consider $\{t_0, t_1, \cdots, t_m\}$ be a set of finite time points. We define an incrementally calculated trust score for source $s$ as $\boldsymbol{\sigma}(s) = \{\sigma_0(s), \sigma_1(s), \cdots,\}$ where $\sigma_i(s)$ is the trust score of $s$ at time $t_i$. At each time point $t_i$, a subset of facts $F_i$ is selected for evaluation. The facts in $F_i$ are evaluated using the trust scores $\sigma_i(\mathcal{S})$=$\{\sigma_i(s_1), \sigma_i(s_2), \ldots,\}$. In return, we update the trust score of the sources to $\sigma_{i+1}(\mathcal{S})$ by incorporating the corroboration result of the facts in $F_i$. Let $t(f)$ denote the time point at which $f$ is selected. In essence, the trust score $\sigma_i(\mathcal{S})$ at time $t_i$ represents the trustworthiness of the sources over the facts $\{f|t(f) < t_i\}$ that have been evaluated up to $t_i$. When the algorithm terminates at $t_m$, the probability $\sigma(f)$ is used to determine the corroborated result of each fact.*

The advantage of using an incrementally calculated trust score is two-fold. First, it enables us to incrementally calculate the trust values for each source. Second, the function to choose a trust value from the sources for facts $p(\boldsymbol{\sigma}(s))$ can be set to $\sigma_i(\mathcal{S})$ at time $t_i$, By incrementally calculating the trust score for each source, the challenge is now how to select facts at each time point so that the correct result could be computed for as many facts as possible. We detail this process in the following section.

## 5.5 Corroboration

In this section, we investigate strategies of selecting facts at each time point and present our corroboration algorithm (denoted as `IncEstimate`). In the following discussion, we assume the scoring of the `TwoEstimate` algorithm (Equation 5.5) is used. We first introduce our fact selecting strategy (namely `IncEstHeu`) in Section 5.5.1. We then present the `IncEstimate` algorithm int Section 5.5.2. We analyze the complexity of `IncEstHeu` in Section 5.5.3.

### 5.5.1 Selecting facts

Recall that in Section 5.4 we showed the main limitation of existing algorithms is that they use a universal trust score for each source and incorrectly dismisses the uncertainty of facts. In order to uncover the correct value of unknown facts, the key challenge is to evaluate each fact $f$ at a point $t_i$ such that $\sigma(\mathcal{S})$ is a more accurate measure for the facts.

Let $\bar{\mathcal{F}}_i \subseteq \mathcal{F}$ be the set of facts that have not been evaluated at $t_i$ and $\sigma_i(\mathcal{S})$ be the trust values for the sources. It is yet difficult to decide a set of facts $F_i \in \bar{\mathcal{F}}_i$ such that $\sigma_i(\mathcal{S})$ is accurate for $F_i$. Now let us again look at the problem from the information entropy perspective. Since we know that the entropy for an unknown fact is 0 if its probability is 1 (or 0), we model the fact selection problem as a problem to maximize the collective entropy $H(\bar{\mathcal{F}}_i)$ of unknown facts.

One possible greedy strategy is to select facts with the highest entropy at each $t_i$. However, such a strategy does not necessarily maximize $H(\bar{\mathcal{F}})$ since the selected $F_i$ would impact the trust values $\sigma(\mathcal{S})$ for the sources. In turn, the updated trust values would affect the entropy of the remaining facts $\bar{\mathcal{F}}_i - F_i$. For instance, suppose we select $r_1$ (which has the highest entropy of 1) at round 2 in the motivating example. Such a selection results in $\sigma(\mathcal{S}) = \{-, 1, 1, 0.5, -\}$ which would decrease the entropy of remaining facts. As a consequence, we would be unable to identify the false facts $r_4$ and $r_{10}$.

Given a set of unknown facts $\bar{\mathcal{F}}_i$, it is easy to see that there exist $2^{|\bar{\mathcal{F}}_i|}$ ways of selecting facts at $t_i$. However, it is computationally expensive to explore all possibility so as to maximize $H(\mathcal{F})$. We approach the problem by selecting a set of facts such that the updated trust values $\sigma_{i+1}(\mathcal{S})$ is unlikely to decrease the entropy of the remaining facts. Consider below 2 cases of trust values $\sigma_{i+1}(\mathcal{S})$ after evaluating $F_i$.

- $\sigma_{i+1}(s_j) > \sigma_i(s_j)$ for each $j$ or
- $\sigma_{i+1}(s_j) < \sigma_i(s_j)$ for each $j$

For simplicity, let us assume that all the remaining facts $\bar{\mathcal{F}}_i$ have $\mathsf{T}$ votes only ($\bar{\mathcal{F}}_i \subseteq \mathcal{F}^*$). Consider the case in which by selecting $F_i$ the trust value increases for each source (*i.e.*, case 1). Since the probability of facts is calculated as the average trust scores of its sources (Equation 5.5) and a higher trust value for the sources would increase the probability of the facts, the entropy decreases for facts with a probability above 0.5 (recall a fact has the highest entropy if it has a probability of 0.5). On the other hand, the updated trust value increases the entropy for the facts with a probability smaller than 0.5 and $\sigma_{i+1}(\mathcal{S})$ brings its probability closer to 0.5 than $\sigma_i(\mathcal{S})$. Similarly, a smaller trust value for the sources would bring down the entropy for facts with a probability smaller than 0.5 and could raise the entropy for facts with a probability greater than 0.5.

Now let us examine the relationship between facts and the trust value changes. We observe that if the evaluation results of $F_i$ are true (or false), the trust value for the sources increase (or decrease). This is based on the intuition that the more correct votes a source has, the more trustworthy it is. Let $\sigma_{i+1}(s)$ be the trust value for source $s$ after evaluating $F_i$ and for each $f \in F_i$ the corroboration result is true, $\sigma_{i+1}(s)$ can be calculated as follows.

$$\begin{aligned}
\sigma_{i+1}(s) &= \frac{\sum_{f_j \in \hat{\mathcal{F}}_i} \sigma(f_j)}{|\hat{\mathcal{F}}_i|} \\
&= \frac{\sum_{f_j \in \hat{\mathcal{F}}_{i-1}} \sigma(f_j) + \sum_{f_j \in F_i} \sigma(f_j)}{|\hat{\mathcal{F}}_{i-1}| + |F_i|} \\
&> \frac{\sum_{f_j \in \hat{\mathcal{F}}_{i-1}} \sigma(f_j)}{|\hat{\mathcal{F}}_{i-1}|} = \sigma_i(s)
\end{aligned} \tag{5.8}$$

where $\hat{\mathcal{F}}_{i+1}$ refers to the facts that have been evaluated up to $t_i$. Note that the above calculations consider the probability to be 1 for true facts.

Based on the discussions above, we now have a viable strategy. We first group unevaluated facts based on the sources of the votes. Facts in the same group receive votes from the same set of sources. The intuition behind is that facts with the same votes should have the same corroboration result. As an instance in the motivating example, $r_5$ and $r_8$ are grouped together since they have the same votes. We then calculate a score $\Delta H(\bar{\mathcal{F}})$ for each fact group $FG$ that represents the entropy change for the remaining facts if $FG$ is selected. We rank fact groups in decreasing order of their $\Delta H(\bar{\mathcal{F}})$ scores and pick the one with the highest score.

$$\Delta H(\bar{\mathcal{F}})_{FG} = \sum_{FG' \in \bar{\mathcal{F}} - FG} (H_{i+1}(\bar{\mathcal{F}})_{FG'} - H_i(\bar{\mathcal{F}})_{FG'}) \tag{5.9}$$

There is one special case in which given $\sigma(\mathcal{S})$, all remaining facts have a probability above (or below) 0.5. In this case, the facts would be evaluated to be true (or false) which is equivalently as having a entropy of 0 (recall true facts have a normalized probability of 1). Such a scenario could be caused if `IncEstHeu` repeatedly selects facts that evaluated to be true facts. To avoid this effect, we slightly modify our strategy as follows. During each time point $t_i$, we divide fact groups into positive part (fact groups with probability above 0.5) and negative part (fact group with probability below 0.5). We then pick one fact group from each part with the highest $\Delta H(\bar{\mathcal{F}})$ score. In addition, we require that the same number of facts are selected from each group. Let $FG_i^+$ and $FG_i^-$ denote the positive and negative fact group, and we use

$size(FG)$ to denote the number of facts of group $FG$. `IncEstHeu` selects $n$ facts from each group where $n$ = $min\{size(FG_i^+), size(FG_i^-)\}$. The rationale behind is that as $FG_i^+$ and $FG_i^-$ become extremely disparate in sizes, the updated trust scores are dominated by the larger fact group.

### 5.5.2 The Algorithm

**Algorithm 1** Incremental Estimate (`IncEstimate`)

**Input**: $\mathcal{F}$: a collection of facts; $\mathcal{S}$: a set of sources

**Output**: $\sigma(\mathcal{S})$: estimations for the sources, $\sigma(\mathcal{F})$: estimations for the facts

```
 1: Initialize $\sigma_0(\mathcal{S})$, $\sigma(\mathcal{F})$
 2: while $|\bar{\mathcal{F}}| > 0$ do
 3:     $F_i \leftarrow Select\_Facts(\bar{\mathcal{F}}, \sigma_i(\mathcal{S}))$
 4:     $\bar{\mathcal{F}} \leftarrow \bar{\mathcal{F}} - F_i$
 5:     for all $f \in F_i$ do
 6:         $\sigma(f) \leftarrow Corrob(f, \sigma_i(\mathcal{S}))$
 7:         $\mathcal{W} \leftarrow \mathcal{W} \cup f$
 8:     end for
 9:     $\sigma_{i+1}(\mathcal{S}) \leftarrow Update\_Trust(\mathcal{W})$
10: end while
11: return $\sigma(\mathcal{S}), \sigma(\mathcal{F})$
```

Algorithm 1 demonstrate the overall flow of our `IncEstimate` algorithm. Our `IncEstimate` takes a set of facts and a set of sources as input, and output the estimations of the trust scores of the sources as well as the probabilities of the facts. At first, we initialize $\sigma_0(\mathcal{S})$ and $\sigma(\mathcal{F})$ with a default value (*e.g.*, 0.9). Our algorithm then repeatedly selects a new subset of facts in each round. During each round (line 2-10), the new set of facts are selected using the $Select\_Facts(\bar{\mathcal{F}}, \sigma_i(\mathcal{S}))$ function (line 3, defined in Algorithm 2). The corroboration calculates the probability for each selected fact (line 6) and inserts it into a set $\mathcal{W}$ that contains facts which have been evaluated (line 7). The trust scores of the sources are then updated incorporating the results of facts that have been selected. Our algorithm terminates when all facts have been evaluated and returns $\sigma(\mathcal{S})$ and $\sigma(\mathcal{F})$.

We briefly discuss the `IncEstHeu` strategy defined in Algorithm 2. At first, the `IncEstHeu` strategy initializes a set $\mathcal{W}$ which is the set of facts that are to be selected, and $\mathcal{P}$ and $\mathcal{N}$ which represents the set of positive and negative fact groups respectively (line 5). The set $\mathcal{P}$ and $\mathcal{N}$

**Algorithm 2** $Select\_Facts(\bar{\mathcal{F}}, \sigma(\mathcal{S}))$

```
IncEstHeu:
    $\mathcal{W} \leftarrow \emptyset, \mathcal{N} \leftarrow \emptyset, \mathcal{P} \leftarrow \emptyset$
    $\mathcal{P} \leftarrow$ all fact groups $FG$ in $\bar{\mathcal{F}}$ s.t. $\sigma(FG) > 0.5$
    $\mathcal{N} \leftarrow \bar{\mathcal{F}} - \mathcal{P}$
    sort $\mathcal{P}, \mathcal{N}$ in decreasing order of $\Delta H(\bar{\mathcal{F}})$
    $FG^{+} \leftarrow peek(\mathcal{P})$  $FG^{-} \leftarrow peek(\mathcal{N})$
    $n \leftarrow min\{size(FG^{+}), size(FG^{-})\}$
    for $i = 1 \rightarrow n$ do
        $\mathcal{W} \leftarrow \mathcal{W} \cup peek(FG^{+}) \cup peek(FG^{-})$
    end for
    return $\mathcal{W}$
```

are then filled with positive and negative facts that have not been evaluated (line 6 and 7). Note that the $peek(\mathcal{P})$ function pops the first elements from $\mathcal{P}$. We then sort the set $\mathcal{P}$ and $\mathcal{N}$ based on their $\Delta H(\bar{\mathcal{F}})$ in decreasing order (line 8). From $\mathcal{P}$ and $\mathcal{N}$, we pick the fact group that has the highest $\Delta H(\bar{\mathcal{F}})$ score, denote as $FG^{+}$ and $FG^{-}$ respectively (line 9). We select $n$ facts from each group where $n$ is the number of facts of the smaller group between $FG^{+}$ and $FG^{-}$ (line 10-13).

### 5.5.3 Complexity analysis

In this section, we analyze the complexity of our `IncEstimate` algorithm. As a comparison, a voting based method simply counts the number of votes for each fact and therefore incurs a cost of $\Theta(|\mathcal{F}|)$. Methods that iteratively computes the scores for the facts and sources have a cost of $\Theta(m(|\mathcal{F}| + |\mathcal{S}|))$, where $m$ is the number of iterations needed for convergence.

Our `IncEstimate` algorithm incurs additional cost when calculating projected scores for unevaluated facts and updating trust scores for the sources at each time point. Let $t_m$ be the number of time points `IncEstimate` needs to evaluate all facts. As we see before, `IncEstHeu` evaluates at least one fact group at each time point, therefore we can bound $t_m$ by the number of fact groups in $\mathcal{F}$. Recall that a vote takes a value from $\{T, F, -\}$, therefore the maximum number of fact groups is $3^{|\mathcal{S}|} - 2|\mathcal{S}| - 1$ (we excluded fact groups with only one vote or no vote). Further, since we focus on scenarios where most facts receive `T` votes only, the bound for the number of fact groups can be reduced to $O(2^{|\mathcal{S}-\mathcal{S}^*|} \cdot 3^{|\mathcal{S}^*|})$ where $\mathcal{S}^*$ is the set of sources that cast `F` votes ($\mathcal{S}^* < \mathcal{S}$).

At each time point, `IncEstHeu` calculates projected scores for unevaluated facts and update trust scores for the sources. In the worse case scenario, each of the fact groups evaluated at $t_1$ through $t_{m-1}$ contains one fact, and the fact groups evaluated at $t_m$ contains $|\mathcal{F}|-2(t_m-1)$ facts. The total cost on calculating projected score is therefore $t_m \cdot (t_m-1)+(|\mathcal{F}|-2t_m+2)\cdot t_m$. In the best case scenario, the majority facts $|\mathcal{F}|-2(t_m-1)$ are evaluated at $t_1$ and 2 facts are evaluated from $t_2$ through $t_m$, which brings the cost to $(|\mathcal{F}|-2t_m+2)+t_m \cdot (t_m+1)$. On the average case where the number of facts in fact groups is uniformly distributed, the cost is $\frac{|\mathcal{F}|(t_m+1)}{2}$. By adding the cost for calculating trust scores for the sources, the total cost for `IncEstimate` can be bounded by $O(|\mathcal{F}| \cdot 2^{\mathcal{S}-\mathcal{S}^*} \cdot 3^{\mathcal{S}^*})$.

While the cost of `IncEstimate` is exponential in the number of sources, in typical cases this number is small. In addition, whether adding more sources results in better corroboration quality is still an open question [21]. Moreover, $t_m$ is also bounded by the number of facts $|\mathcal{F}|$ since at least one fact is evaluated at each time point, and therefore the cost for `IncEstimate` can be bounded by a polynomial term $O(|\mathcal{F}|^2)$. Our experimental results in Section 5.6.2 show the overhead of a more sophisticated algorithm is acceptable in exchange for better corroboration results.

## 5.6 Experiments

In this section, we present our experimental results. Section 5.6.1 describes the experiments setup. We first present our results on the real-world dataset, the restaurant application mentioned before in Section 5.6.2. We then show the experimental results on synthetic datasets in Section 5.6.3.

### 5.6.1 Setup

In this section, we present our experiment setup and evaluation metrics, as well as the algorithms we implemented for comparison.

### Algorithms

**[Baseline methods]**: We provided two baseline approaches named `Counting` and `Voting`. The `Counting` method assigns a $true$ result to each fact if more than half the sources report it true. In contrast, the `Voting` method considers a fact as $true$ if there exist more sources reporting it true than false.

**[Corroboration methods]**: We implemented the strategy `IncEstHeu` of our incremental algorithm introduced in Section 5.5. We used a default trust score $\sigma(\mathcal{S})$ of 0.9 for each source to start our algorithm. We tested other default values and we observed all default value above 0.5 generate the same corroboration result. This is because despite different $\sigma_0(\mathcal{S})$ used, the same facts are selected at $t_0$, and therefore they result in the same trust value at $t_1$. We are also interested in how a different fact selection strategy would impact the performance of `IncEstimate`. To that end, we implemented `IncEstPS`, a simple strategy that selects the fact group with the highest probability at each time point. The rationale behind is that facts with higher probability are more likely to receive correct corroboration results. Compared with a balanced strategy `IncEstHeu` that considers both positive and negative facts, we want to see how a naive greedy strategy performs in the competition. We also implemented the `TwoEstimate` algorithm [28] and the `BayesEstimate` algorithm [81] for comparison. For the `BayesEstimate` algorithm, we used the same assumption as in [81] that sources have low false positive rate but high false negative rate. In particular, we set $\alpha_0$ = (100, 10000), $\alpha_1$ = (50, 50), and $\beta$ = (10, 10).

**[ML-based methods]**: Since our problem can be naturally seen as a classification problem, we also tested machine learning based algorithms using the votes as features. In particular, we tested 2 classifiers using Weka: a SVM classifier (using SMO implementation) and a logistic classifier with default parameter. We report the results using 10-fold cross validation.

### Environment and Metrics

We implemented all the algorithms using Java SDK 6. All the experiments were conducted on a Mac OS 10.8.2 with a quad-core CPU of 3.3 GHz and 8GB Ram. We use the following metrics to evaluate the results of various algorithms.

| Source coverage | YellowPages | Foursquare | MenuPages | OpenTable | CitySearch | Yelp |
|---|---|---|---|---|---|---|
| | 0.59 | 0.24 | 0.20 | 0.07 | 0.50 | 0.35 |
| Source overlap | YellowPages | Foursquare | MenuPages | OpenTable | CitySearch | Yelp |
| YellowPages | 1 | 0.22 | 0.18 | 0.04 | 0.43 | 0.26 |
| Foursquare | 0.22 | 1 | 0.30 | 0.08 | 0.22 | 0.29 |
| MenuPages | 0.18 | 0.30 | 1 | 0.09 | 0.17 | 0.29 |
| OpenTable | 0.04 | 0.08 | 0.09 | 1 | 0.05 | 0.07 |
| CitySearch | 0.43 | 0.22 | 0.17 | 0.05 | 1 | 0.27 |
| Yelp | 0.26 | 0.29 | 0.29 | 0.07 | 0.27 | 1 |
| Source accuracy | YellowPages | Foursquare | MenuPages | OpenTable | CitySearch | Yelp |
| | 0.59 | 0.78 | 0.93 | 0.96 | 0.62 | 0.84 |

Table 5.3: Source coverage and overlap

**Precision, Recall, Accuracy:** We first report standard information retrieval metrics to evaluate the results of all algorithms.

**Mean square error of trust score (MSE):** We use $t(s_i)$ to denote the trustworthiness of source $s_i$ over a sampled golden set, and let $\sigma(s_i)$ denote the computed trust score of $s_i$ by a corroboration algorithm. The mean square error of trust score is computed as follows.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(t(s_i) - \sigma(s_i))^2 \qquad (5.10)$$

### 5.6.2 Real-World Dataset

We report our experiment results over real world datasets in this section.

#### Dataset

We used the restaurant example discussed in Section 5.2 in our experimental evaluation. We used the same example in [49] and reported results of existing techniques on a small sample of restaurant listings. In this study, we expanded our investigation and conducted experiments in a much larger scale. We crawled data from six major sources for restaurant listings[3], namely Yellowpages, Foursquare, Menupages, Opentable, Citysearch, and Yelp. Some of the web sources allow accesses to the list of restaurant listings at a given a location (*e.g.*, Manhattan), while for others we have to do random accesses to probe as many listings as possible. In

[3]We comducted the crawling in Feb 2012.

this particular experiment, we consider restaurant listings in the greater New York City area. Our initial crawling yielded 42969 restaurant listings but contains numerous duplicates due to various presentation of the same listing. In order to clean the data and remove duplicates, we employ the following strategy. We first wrote a rule-based script to normalize the addresses of all listings. Listings that share the same address are then grouped together. We calculate a similarity score between each pair of listings within a group and we consider two listings are the same entities if their similarity score is above a certain threshold. For the purpose of this project, we adopted the cosine similarity score at the term level as well as 3-gram level and used a threshold of 0.8. After removing the duplicates, we recorded 36916 restaurant listings from these 6 sources. Among those, only 654 listings ($<$2%) have F votes from sources. More specifically, F votes come from 3 sources, Foursquare (10), Menupages (256), and Yelp (425).

Table 5.3 reports source coverage (the fraction of the total restaurant listings contained in each source), as well as the source overlap (a measure of how much two sources have in common). As shown, all sources contain only a fraction of the entire listings. Among all sources, 2 sources (*i.e.*, Yellowpages, Citysearch) have significantly more coverage ($>$50%) compared with others.

**Golden set**: In order to evaluate the performance of various algorithms, we must create a golden set of listings for which we know for certain their correct value (true or false). Unfortunately, there does not exist an authoritative source that can provide such information. To that end, we selected restaurant listings from 3 zip codes and investigated their legitimacy in person[4]. Overall, our golden set contains 601 listings, out of which 340 are true and 261 are false. We also calculated the accuracies of all sources in the golden set, listed in Table 5.3. Unsurprisingly, sources that have direct connection with restaurants (*e.g.*, OpenTable and Menupages) have high accuracies ($>$0.9). Interestingly, the two sources with significantly higher coverage (Yellowpages, Citysearch) are also the sources with low accuracy ($\sim$0.6).

Identifying legitimate restaurants is not a trivial task. Before we embarked on designing a corroboration algorithm for this task, we tried to leverage the reviews information from some

[4]The in-person check-up was conducted during Apr 2012.

of the sources to predict whether a restaurant listing is legitimate. In particular, we used a variety of meta data (number of reviews, average interval of review time stamp, length since last review, etc) as well as the review content as features and tested using a SVM classifier. However, the classifier resulted in a less-than-satisfactory accuracy ($< 0.7$).

**Corroboration quality**

| | Precision | Recall | Accuracy | F-1 |
|---|---|---|---|---|
| `Voting` | 0.65 | 1.00 | 0.66 | 0.79 |
| `Counting` | 0.94 | 0.65 | 0.76 | 0.77 |
| `BayesEstimate` | 0.63 | 1.00 | 0.67 | 0.77 |
| `TwoEstimate` | 0.65 | 1.00 | 0.66 | 0.79 |
| `ML-SVM (SMO)` | 0.98 | 0.74 | 0.77 | 0.84 |
| `ML-Logistic` | 0.86 | 0.85 | 0.82 | 0.82 |
| `IncEstPS` | 0.66 | 1.00 | 0.68 | 0.79 |
| `IncEstHeu` | 0.86 | 0.86 | **0.83** | **0.86** |

Table 5.4: Result of real-world dataset

Table 5.4 lists the performance of various algorithms as well as the `Counting` and `Voting` methods. Since for most of the listings there exist only `T` votes, the `Voting` method assigns them a *true* result, thus results in a perfect recall value (1.0) but a low precision (0.65). In contrast, the `Counting` method uses a high threshold to filter out listings with insufficient `T` votes, hence has a high precision (0.94). However, the threshold is high enough to reject a lot of legitimate listings, therefore resulting in a low recall value (0.65). The two non-corroboration based approaches have an accuracy of 0.66 and 0.76, respectively.

Existing corroboration-based approaches do not perform much better than the two baseline approaches. As they cannot leverage conflicting information from the data, the `TwoEstimate` algorithm has almost the same result as the `Voting` method by assigning a *true* result to every listing except for a small set for which there are more `F` votes than `T` votes. The `BayesEstimate` algorithm also has very similar results as `TwoEstimate`. In addition to suffering from the lack of conflicting votes among facts, the `BayesEstimate` algorithm relies heavily on the prior knowledge regarding the sources. The *high-precision low-recall* prior that is used by `BayesEstimate` is clearly different from the actual trustworthiness of the sources, as we see in Table 5.3 that both Yellowpages and Citysearch have poor precisions.

The two machine learning based algorithms perform noticeably better than both baseline

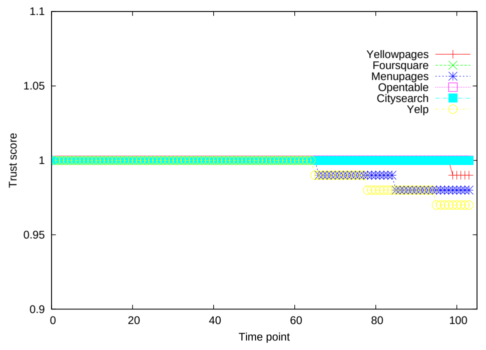
| | YellowPages | Foursquare | MenuPages | OpenTable | CitySearch | Yelp | MSE |
|---|---|---|---|---|---|---|---|
| Source accuracy | 0.59 | 0.78 | 0.93 | 0.96 | 0.62 | 0.84 | - |
| `TwoEstimate` | 1.00 | 1.00 | 0.98 | 1.00 | 1.00 | 0.98 | 0.063 |
| `BayesEsitmate` | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.066 |
| `ML-Logistic` | 0.62 | 0.85 | 0.98 | 0.92 | 0.65 | 0.95 | **0.004** |
| `IncEstHeu` | 0.51 | 0.70 | 0.90 | 0.93 | 0.51 | 0.89 | 0.005 |

Table 5.5: The mean square error of trust score

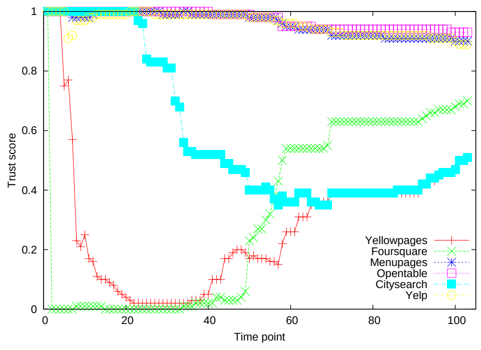
and existing corroboration methods. In particular, the support vector classifier improves on both accuracy (0.77) and recall (0.84). In comparison, the logistic classifier proves to be a better model in this case, with a precision of 0.86 and an accuracy of 0.82. Unsurprisingly, the most discriminating features are the `F` votes from the 3 sources. The performance gain of machine learning algorithms is largely due to the consideration of missing votes among sources. As we discussed earlier, a missing vote could be seen as either a `F` vote or that a source has no knowledge about the fact. By taking advantage of the missing votes, machine learning based algorithms can leverage extra knowledge and therefore improve accuracy of predictions.

Our `IncEstHeu` strategy significantly outperforms the baseline and existing corroboration methods, and slightly improve on accuracy and recall compared with machine learning based algorithms. The improvement is statistical significant for both baseline and existing corroboration techniques (with $p$-value ¡ 0.001). The `IncEstPS` strategy has a similar result as existing approaches and improves on accuracy only marginally. This is due to the way `IncEstPS` selects facts at each time point. We observe `IncEstPS` repeatedly selects facts with high probability which are evaluated to be true. As a consequence, the trust values remain high and most of the facts are evaluated to be true except for a few with more `F` votes than `T` votes. The `IncEstHeu` has a good balance between precision and recall, results in the best value for accuracy and F-1. In particular, `IncEstHeu` results in more true negatives (141 in the golden set). The best machine learning method (`ML-Logistic`) has 137 true negatives.

Although the improvement of our `IncEstHeu` over the machine learning based approaches is not statistically significant, we argue that our method is advantageous in such task. For one thing, our approach does not require any training data, which could be difficult to obtain in certain applications. In addition, the machine learning methods are trained using a small dataset



(a) `IncEstPS`



(b) `IncEstHeu`

Figure 5.2: Multi-value trust score at each time point

and it is unclear whether it would scale to a larger unseen dataset due to the fact the model could be overfitting over the small golden set.

**Mean square error**

Table 5.6.2 lists the corroborated trust scores of the sources of various algorithms as well as their MSEs. For `IncEstHeu` we report the trust scores for the sources at the end of last time point, which reflects their trustworthiness over the entire dataset. Compared with the actual source accuracy over the golden set, the `IncEstHeu` is clearly the best performer (almost identical trust score for Menupages, Opentable), thus results in the smallest MSE (0.005) among all corroboration techniques. This is due to the fact that it adapts its trust value for each fact group. The `TwoEstimate` algorithm, which is unable to identify most illegitimate listings, concludes all the sources as perfect or near perfect sources. Similar as precision and recall, the `BayesEsimate` algorithm assign a trust score to each source similar to `TwoEstimate`. The machine learning method `ML-Logistic` has the best MSE value overall, slightly outperform our best strategy. This is because the machine learning methods are specifically trained using the golden set. In addition, our method reports the trust score at the end of the last round, which represents the trust score over the entire dataset, and therefore it is not surprising that it deviates from the trustworthiness of the sources on the golden set.

### Multi-value trust score

In this section, we illustrate how the trust score for each source changes when using different strategies of the `IncEstimate` algorithm. Figure 5.2 plots, for each strategy, different trust scores that are used for corroboration at each time point.

Since we use an initial trust score $\sigma(\mathcal{S})_0$ for each source, all sources share the same trust score at $t_0$. As the `IncEstimate` algorithm continues, each strategy develops different trust score trajectory for the sources. In particular, the `IncEstPS` strategy (Figure 5.2(a)) chooses the set of facts with the highest probability which are evaluated to be true. In return, the true facts boost the trust score for the sources that cast votes. Since `IncEstPS` favors facts with high probability, the trust scores for the sources remain at 1 until all facts with only `T` votes have been evaluated. It is not surprising that from then on, trust values for sources with `F` votes start to decrease since facts with `F` votes are evaluated to be true. Eventually, `IncEstPS` is only able to identify 2 true negatives, which is similar as existing corroboration techniques.

In contrast, the `IncEstHeu` strategy overcomes the limitation of `IncEstPS` by selecting both positive and negative listings during each round. This results in significantly different trust score change from the `IncEstPS` strategy (Figure 5.2(b)) While after evaluating $F_0$ all sources are positive sources, the trust scores for both Citysearch and Yellowpages begin to dip as more false facts with `F` votes are identified, which effectively makes them negative sources (after $t_{12}$). With the presence of negative sources, `IncEstHeu` is able to uncover false facts from $\mathcal{F}^*$ which explains a significantly higher number of true negatives. `IncEstHeu` then continues to evaluate facts and the trust scores eventually converges to the actual accuracy for the sources.

### Time cost

Inevitably, a more sophisticated corroboration algorithm incurs additional time cost in computation. Our `IncEstimate` algorithm suffers from the overhead of the multiple round corroboration. Table 5.6 lists the time cost of various algorithms over the real world dataset. We used the ‘time’ command to test the time cost of each algorithm and report the *real* part. The two baseline approaches, `Voting` and `Counting`, which only considers the number of `T`

| | Time cost (secs) |
|---|---|
| `Voting` | 0.60 |
| `Counting` | 0.61 |
| `BayesEstimate` | 7.38 |
| `TwoEstimate` | 0.69 |
| `ML-SMO` | 0.99 |
| `ML-Logistic` | 0.91 |
| `IncEstPS` | 1.13 |
| `IncEstHeu` | 1.15 |

Table 5.6: Time cost of various algorithms

| | Number of errors |
|---|---|
| `Voting` | 292 |
| `Counting` | 327 |
| `TwoEstimate` | 269 |
| `ThreeEstimate` | 270 |
| `IncEstHeu` | 262 |

Table 5.7: Results of various algorithms over the Hubdub dataset

and `F` votes, are the fastest ones, with a time cost of 0.6 and 0.61 seconds, respectively. The `TwoEstimate` algorithm, which applies corroboration on all the listings at once, is also fairly efficient, with a time cost of 0.69 seconds. The `BayesEstimate` algorithm requires a burning period before stabilizing and results in the longest time (7.38 secs). The two machine learning based approaches take less than 1 second largely due to the fact that they only run over the golden set. The best strategy of our `IncEstimate` algorithm results in a little more than 1 second, with the best performing strategy having an acceptable time cost of 1.15 secs.

#### The Hubdub Dataset

Our technique focuses on the scenario where most or all facts have only `T` votes. Nevertheless, we do not believe that our incremental algorithm is limited to such cases. To demonstrate the effectiveness of `IncEstimate` in dataset with ample conflicting votes, we use the Hubdub dataset from [28]. The Hubdub dataset was constructed using a snapshot of settled questions from hubdub.com, which contains 830 facts from 471 users on 357 questions.

Table 5.7 report the results of various algorithms on the Hubdub dataset. We did not include the machine learning based methods since this task involves more than two candidate answers.

| Snapshot | Precision | Recall | F-measure |
|---|---|---|---|
| 1 | .843 | .722 | .778 |
| 2 | .843 | .688 | .758 |
| 3 | .873 | .699 | .776 |
| 4 | .878 | .749 | .808 |
| 5 | .843 | .759 | .799 |
| 6 | .744 | .816 | .778 |
| 7 | .873 | .821 | .857 |
| 8 | .840 | .832 | .836 |

Table 5.8: Results of INCESTIMATE over the restaurant dataset in [18]

For comparison, we report the same metric used in [28], the number of errors (the sum of false positive and false negative). The best performance in [28] was from `TwoEstimate`, which recorded 269 errors. Our `IncEstHeu` outperforms all existing methods by reducing it to 262 errors. This proves that `IncEstHeu` is not only suitable for the corroboration problem discussed in this chapter, but also effective in scenarios with conflicting statements.

**The online listing dataset**

In addition to the Hubdub dataset, we also tested our approach on the dataset used in [18] that comprises restaurant listings from a set of web sources. The authors in [18] studied the problem of identifying the true values of data objects when the update history of the sources is known. To that end, a method that takes into consideration the coverage, exactness and freshness of the sources is proposed. To test the effectiveness of their method, [18] used a dataset that contains a list of restaurants scraped from multiple listing websites containing several snapshots over a period of time to determine whether each of the restaurant is still in business. Due to the similarity in nature between this dataset and the one we used in 5.6.2, we like to test if our algorithm is robust enough to handle another dataset.

We obtained a copy of the raw dataset which contains all the 8 snapshots of weekly crawling results of restaurant listings from 12 sources. Similarly as the authors did in [18], we used the name to identify each restaurant and only retain those that appeared in more than one source. We ran our algorithm against all 8 snapshots of data and report their precision, recall and the F-measure in Table 5.8.

As shown in Table 5.8, our INCESTIMATE algorithm performs consistently well against all

| | Precision | Recall | F-measure |
|---|---|---|---|
| NAIVE | .70 | .93 | .80 |
| CEF | .83 | .88 | .85 |
| COPYCEF | .86 | .87 | .86 |
| INCESTIMATE | .84 | .83 | .84 |

Table 5.9: Results of various algorithms over the restaurant dataset in [18]

8 data snapshots. The precision is reasonably good among all runs ($> 0.8$), except for snapshot #6 which took a notable hit in precision. Note that it is possible that two snapshots with minor updates could result in a sizable difference in the evaluation results under INCESTIMATE. This is because that even a slight difference in voting from the sources could cause INCESTIMATE to assign an initial trust score to sources that might deviate from its true trustworthiness over all facts. Since selecting facts at each iteration depends on the trust scores from the previous round, an inaccurate initial estimate of the trust scores might take longer to correct. On the other hand, the recall of INCESTIMATE has seen a steady gain as we evaluate each snapshots in chronicle order. This is hardly surprising due to the fact that the test set was selected as restaurants that later disappeared from the listings. By obtaining more evidence of those disappearing listings, INCESTIMATE is able to uncover more true negatives.

Table 5.9 lists the results of various approaches reported in [18], as well as INCESTIMATE at snapshot #8 which represents the final result of the entire dataset. As shown in the table, both CEF and COPYCEF perform exceptionally well with COPYCEF having the best overall F-measure value. Our INCESTIMATE algorithm also performs quite well, with a precision of 0.84 and a recall of 0.83 that only trails slightly the best-performing COPYCEF method. This minor performance disparity can be explained due to the fact that CEF and COPYCEF were designed to capture not only the correctness and coverage, but also the freshness of the sources. As such, they perform extremely well when a rich history of data values is available. In addition, COPYCEF takes into consideration of the data dependencies among the sources which could effectively rule out copious votes. In contrast, although our approach focus on more on static data, it is generating results with a more than acceptable quality and is very suitable when the history of data values is expensive to obtain.

### 5.6.3 Synthetic Dataset

We present our experiment results on synthetic datasets in this section. We first provide details on how we generate synthetic datasets (Section 5.6.3) and then present the corroborated results (Section 5.6.3).

#### Dataset

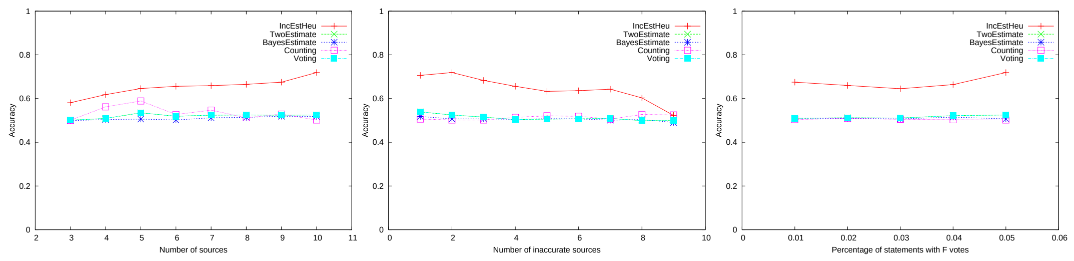
We use the following model to generate synthetic datasets. We consider all the sources are positive sources, *i.e.*, with a trust score of greater than 0.5. For each source $s$, let $\sigma(s)$ and $c(s)$ denote its trust score and coverage. For each fact, we randomly assign a correct value of either true or false. We also consider a factor $\eta$ that determines the percentage of facts that have F votes. The parameters $\sigma(s)$ and $c(s)$ controls whether and how a source $s$ casts votes on facts. Motivated by the observation in the real world dataset, we divide the sources into accurate sources (*e.g.*, Menupages) and inaccurate sources (*e.g.*, Yellowpages). In particular, we create sources as follows.

- **Accurate sources** are created with a trust score uniformly distributed in [0.7, 1.0]. In addition, each accurate source $s$ is associated with a probability $m(s)$ that it casts a F vote for a false fact. We set $m(s)$ to be uniformly distributed in [0, 0.5].
- **Inaccurate sources** are created with a trust score uniformly distributed in [0.5, 0.7]. Inaccurate sources do not cast F votes for any fact.

We generate coverage for each source by following the intuition that inaccurate sources have a higher coverage compared with accurate sources. In particular, the coverage is calculated using Equation 5.11,

$$c(s) = 1 - \sigma(s) + random() * 0.2 \tag{5.11}$$

where $random()$ is a function that generates a random real number in [0, 1]. For each synthetic dataset we generate 20000 facts which are randomly assigned a true or false value.



(a) Varying number of sources (b) Varying number of inaccurate sources (c) Varying percentage of F votes

Figure 5.3: Corroboration results of synthetic datasets

### Results

Figure 5.3 plots the performance comparison of various algorithms in the synthetic datasets. In particular, Figure 5.3(a) illustrates the accuracy of algorithms with a varying total number of sources. In this experiment, we fix the number of inaccurate sources at 2. As shown, our `IncEstHeu` algorithm consistently outperforms all other methods by a large margin. As the number of accurate sources increases, the accuracy of the `IncEstHeu` improves. In contrast, except for the `Counting` method, all methods remain almost flat as the number of sources change. The performance of existing algorithms is not unexpected. Although the majority sources are accurate, their low coverage, coupled with the fact there exist very few conflict votes, renders the state-of-the-art methods incapable of identifying false facts.

Figure 5.3(b) demonstrates the results under a varying number inaccurate sources, with the total number of sources fixed at 10. We are seeing similar results as shown in Figure 5.3(a). Unsurprisingly, as the number of inaccurate sources increases, the accuracy of the `IncEstHeu` decreases and eventually drops to the same level when 9 out 10 sources are inaccurate. Our `IncEstHeu` outperforms all other methods by as much as 37%.

Figure 5.3(c) shows the results with a different percentage $\eta$ of facts that have `F` votes (from 0.01 to 0.05). We fix the number of total and inaccurate sources at 10 and 2 respectively. Again, the `IncEstHeu` algorithm generates significantly more accurate corroboration results than any other methods.

## 5.7 Conclusion

We studied the corroboration problem in a scenario in which there exists little conflicting information. We tackle the problem by proposing an algorithm based on a multi-value trust score for each source. For each source, we use a different trust score when evaluating different sets of statements. We leverage the entropy of unknown facts and derive strategies of choosing facts during each round in our algorithm. We conduct experiments on both synthetic and real-world datasets and demonstrate that our algorithm significantly outperforms state-of-the-art corroboration methods and improves accuracy over machine learning based approaches. The work presented in this chapter was published in [49] and [73].

# Chapter 6

# Corroborating Joined Information over Web-accessible Databases

## 6.1 Introduction

We have demonstrated in the previous two chapters that corroboration techniques could leverage the quality of the sources to improve the quality of the answers (Chapter 4). Moreover, we proposed finer-grained trustworthiness for the sources (*i.e.*, multi-value trust scores) that captures an observation that a source may exhibit different trustworthiness towards different queries (*facts*). Such technique proves to be extremely effective in a situation where the sources agree on the same answer for most queries (Chapter 5).

Although the corroboration methods proposed in the previous two chapters are aimed towards different scenarios, the corroboration problems (*queries*) are both *simple* queries: queries whose candidate answers can be independently extracted from a single source. As an example, in Chapter 4, each candidate answer (*e.g.*, the `MPG` of a car) is independently extracted from a single source. Unfortunately, the fact that an answer can be extracted from a single source is not always true for real world scenarios. Consider the following query.

*Q: what is the elevation of the highest mountain in the US*

| | |
|---|---|
| (1, "Mount Everest", 0.3) | (1, "29029 ft", 0.9) |
| (2, "Mount McKinley", 0.9) | (2, "20322 ft", 0.7) |
| (3, "Mount Vancouver", 0.7) | (3, "18000 ft", 0.6) |
| $s_1$ | $s_2$ |

Figure 6.1: Illustration of Example 1

EXAMPLE 1: *Consider the query "What is the height of the highest mountain in the United States?". It is possible that a single source might provide the answer to this query. But in a* *more probable scenario we may find two sources where $s_1$ provides a list of highest mountain in the US and $s_2$ provides the height of a list of mountains (Figure 6.1).*

EXAMPLE 2: *As another example, consider the query "Show me the upcoming movies that are directed by winners of Oscar best actor". It is unlikely that we could find a source that provides a direct answer. We could however, identify a list of Oscar best actor winners as well as the list of movies that are about to be released from multiple sources.*

For both scenarios, each value provided from the sources is only partial information of an answer to the original queries, and a post processing function similar to the *join* operation in databases is needed in order to generate the final answer to the queries. In addition, each partial information may be correct or incorrect, and corroboration techniques need to be employed in order to ensure the correctness of the final answer.

While search engines and deep Web technologies have been increasingly effective at finding relevant documents and information in a certain domain, they are less capable of automatically combining pieces of relevant information to form a consolidated answer that is of users' interest. In this chapter, we discuss corroborating answers for queries that involve joining information from multiple sources (denoted as *join queries*). The problem of join query corroboration significantly differs from the ones we introduced in the previous two chapters (henceforth denoted as traditional corroboration problems) in two major parts. First, in traditional corroboration problems, each source provides a candidate answer and the correct answer is decided as one of the candidate answers. In this case, each source provides partial information and only by combining the partial information from multiple sources can we construct a candidate answer. The correct answer is then picked from a set of candidate answers, each of which is generated using partial information from the sources. Second, the number of potential candidate answers increases dramatically in join query corroboration. For traditional corroboration problems, the number of candidate answers is bounded by the total number of different candidate answers from the sources (and hence, the number of sources). In contrast, the potential candidate answer space could be as large as the Cartesian product of the cardinality of all sources for the problem of join query corroboration. Therefore, a more pressing challenge is to *efficiently* identify the top answers for join queries.

Motivated by the biggest challenge, we focus on efficiently corroborating answers for

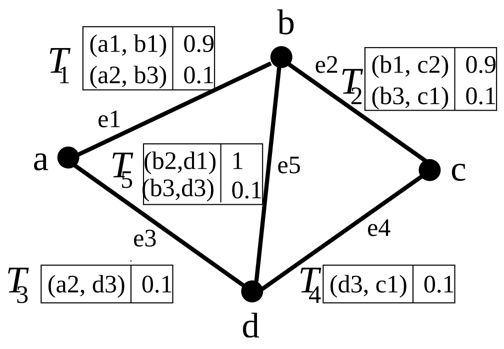


Figure 6.2: An example of join graph depicting the join relations between tables

join queries in this chapter. Formally, let us consider a join query $q$ and a set of sources $\mathcal{S} = \{s_1, s_2, \ldots\}$. Each source $s_i$ provides statements $s_i(q) = \{f_i^1(q), f_i^2(q), \ldots\}$ pertaining to the query $q$ that shed lights on the partial information of a potential answer to $q$. Each statement $f_i^j$ is associated with a score $\sigma$ that represents its quality with respect to the query $q$. We write a candidate answer $f = \{f_1, f_2, \ldots\}$ of query $q$ as a combination of one statement from each source ($f_i \in s_i(q)$) that satisfies the join condition specified in $q$. Given a scoring function $Corrob(f)$ that evaluates the candidate answer $f$ based on the statements from each source, the objective is to efficiently identify a set of answers of size $k$ that have the highest scores. As an example, consider the query $q$ and the sources $s_1$ and $s_2$ in Example 1. We extract statements from each source as $s_1(q) = \{(1, \text{“}MountEverest\text{”}, 0.3)$, $(2, \text{“}MountMcKinley\text{”}, 0.9)$, $(3, \text{“}MountVancouver\text{”}, 0.7)\}$ and $s_2(q) = \{(1, 29029ft, 0.9), (2, 20332ft, 0.7), (3, 18000ft, 0.6)\}$. Based on the statements from $s_1$ and $s_2$, we can construct candidate answers as $(f_1^1, f_2^1)$, $(f_1^2, f_2^2)$ and $(f_1^3, f_2^3)$. Note that although the original query does not explicitly specify the join condition, it is obvious to conclude that, given the two sources, it is a natural join between $s_1$ and $s_2$.

In order to conform to the terminology used in the context of join operations, we from here on refer to the sources as tables and the statements as tuples (*e.g.,* $(1, \text{“}MountEverest\text{”}, 0.3)$). Note that both queries shown in Example 1 and 2 are simple joins in that the tables are joined in a $chain$. In other words, each table joins with two adjacent tables except for the first and last table. Consider a more general join graph shown in Figure 6.1 as an example. Assume that edges represent sources and nodes represent attributes on which two tables join. Intuitively, the score for a join of multiple tuples is computed as the product of the scores of each tuple. Given a value for an attribute (*e.g.*, $a$), a possible join query would be to retrieve the value of another attribute (*e.g.*, $c$) connected to the first attribute via a join path (*i.e.*, $e_1 \rightarrow e_2$). An important observation is that given a source and a destination attribute, there may exist multiple join paths connecting them. We consider that each join path *vouches* for the quality of the generated join result. Therefore, in addition to the attribute value in the destination node, the user may be also interested in the tuples (*i.e.*, edges) that support the join result.

Recent efforts on top-k join query processing have made great strides that are relevant to our corroboration problem. In particular, Ilyas et al. [34] proposed a branch-and-bound algorithm named the rank-join algorithm that efficiently computes the top-k answers for join queries. The key idea is similar to the Threshold Algorithm (TA) proposed by Fagin in [25] that maintains a buffer of candidate answers and threshold of score upper bound of unseen answers and halts when the score of the $k$th answer is above the threshold. The rank-join algorithm would work beautifully for join query corroboration such as the ones mentioned in Example 1 and 2. Despite the performance advantage, the rank-join algorithm suffers from two limitations. First, the join queries considered in [34] involve tables that form only one join path. In contrast, we consider a more general join graph, allowing multiple join paths between the source and destination attributes. The second limitation of rank-join algorithm is that it is essentially considering inner-join, which requires the join answer to have an instantiated value from each source in the join graph. The study in [39] shows that inner join may produce answers with scores that are too low to be of interest.

Note that one important feature of our join query corroboration is that we allow `null` tuples for some of the data sources in a join answer. This is different from the rank-join algorithm which considers only inner-joins. Consider the same join graph in Figure 6.1. Assume we find one complete join answer with score 0.1 on each edge and another partial join answer with score 0.9 on edge $e_1$ and $e_2$ and `null` on all other edges. Clearly the latter join answer has a higher score and therefore is of more interest to the user. Even if the rank-join algorithm could be applied to the join graph considered in our case, it could not produce the latter answer since it contains `null` tuples on some of the join edges. For comparison purposes, we extended the rank-join approach to more general join graphs. Our experimental results show that our approach is significantly better than the rank-join based approach.

We consider the major bottleneck of top-k join query corroboration to be tuple accessing of web-accessible databases. If, for instance, the tables involved in one query are stored on different servers, and can only be accessed via a Web interface, executing a single join between two tables may become very expensive, as Web accesses exhibit high and variable latency. In addition, the query optimizer in one database will generally have no statistics about tables stored at remote sites and thus be unable to offer any improvements over the naive approach.

**Our contributions.** We propose a novel branch-and-bound algorithm for computing the top-k answers for join queries over Web-accessible databases. Rather than computing all the results of the join query, our strategy dynamically retrieves a subset of tuples from each table, and maintains lower and upper scores bounds for the query results that include the retrieved tuples. By ordering the retrieval of table tuples based on the score bounds of the partial results, our algorithm results in significant savings in the number of Web accesses. We make the following contributions:

- We propose a model for scoring answers of arbitrary join graphs based on network reliability. We also develop methods for computing score bounds for partial answers.
- We present a novel branch-and-bound algorithm which aims to minimize the number of Web accesses required for computing the top-k answers.
- We evaluate our algorithms on a variety of queries and data sets and demonstrate the significant benefits they provide.

The rest of this chapter is structured as follows. Section 6.2 presents a real-life example that we use in our experimental study. The example illustrates the concepts that we formally define in Section 6.3. Section 6.4 presents our dynamic probing techniques that efficiently compute the top-k results. We present our experimental study in Section 6.6. We conclude this chapter in Section 5.7.

## 6.2 Illustrative Example

Suppose that a sophisticated marketer wants to design personalized promotional packages for attendees of certain scientific conferences. To optimize his strategy, he would like to find out who are the researchers most likely to attend which conferences, and what are their main reasons. The marketer decides that he could estimate the answer with reasonable accuracy by taking into account the following factors:

F1: Travel cost for each potential attendee to each conference site;

F2: Whether a potential attendee has at least one accepted paper; has a tutorial; is a conference organizer; or is a conference committee member.

F3: How important the conference is in its field.

F4: Whether the attendee is likely to attend in order to meet with a close collaborator such as his Ph.D. advisor; and how likely the collaborator is to attend.

The marketer finds several sites that each contains part of the data he needs. For example, a list of researchers' contributions to various conferences can be obtained from DBLife[1]. The same site also has information on researchers' affiliation, and thus their location. Travel sites return travel costs between any two locations. Conference locations can be obtained from the DBLP website, and IA Genealogy has a fairly large list of researchers' Ph.D. advisors.

Suppose that the following structured data is accessible from these websites.

- Table $Research$ with attributes $\{person, conf, \sigma\}$, where $\sigma$ is the tuple score, normalized between 0 and 1: Tuples connect researchers to conferences. The value $\sigma$ is a measure of the strength of this connection, based on their roles in that conference (author, tutorial giver, organizer etc.). For example, $(A, VLDB09, 0.9) \in Research$ may mean that researcher $A$ will give a tutorial at VLDB09. Intuitively, this means he is very likely to attend VLDB09, so the tuple has a high score. Tuple $(A, ICDE09, 0.5)$ may mean that researcher $A$ has one accepted paper at ICDE09, with another co-author.

[1]http://dblife.cs.wisc.edu/

- Table $Travel$ with attributes $\{person, loc, \sigma\}$: Tuples in this table reflect how cost-effective it is for a researcher to travel to a location. For example, $(A, Shanghai, 0.1)$ means that researcher $A$ has only expensive options for traveling to Shanghai, while $(A, Providence, 0.9)$ means that researcher $A$ has at least one cheap option for going to Providence; e.g., researcher $A$ may live in New Jersey and travel by train.

- Table $People$ with attributes $\{person, advisor, \sigma\}$: Tuples in this table reflect the strength of the professional connection between a person and their advisor. This strength may be measured as, e.g., the percentage of papers a person co-authored with their advisor in the past 5 years; or as the inverse of the number of years since the person graduated.

- Table $Conference$ with attributes $\{conf, loc, \sigma\}$: Tuples contain information on the conference name and location. The value $\sigma$ reflects the importance of the conference in its field.

```
SELECT TOP 100 C.conf
FROM Research R, Travel T, Conference C,
     People P, Research R1, Travel T1
WHERE ((R.conf=C.conf)
   or (R.person=T.person and T.loc=C.loc)
   or (R.person=P.person
       and ((P.advisor=T1.person and T1.loc=C.loc)
           or (P.advisor=R1.person and R1.conf=C.conf))))
   and R.person IN PREDEF-SET
```

Figure 6.3: Query retrieving top 100 conferences that researchers in PREDEF-SET are likely to attend, based on factors F1–F4.

Note that in our model we assume, as in other prior work [58, 15], that the scores of tuples in each table are available. Such scores may be computed based on surveys (e.g., $Conference.\sigma$); by machine learning methods (e.g., examine historical attendance records to learn a model for $Research.\sigma$); or by formulas provided by the query issuer (e.g., the marketer believes that $People.\sigma$ should be computed as $(years)^{-1}$, where $years$ is the number of years since a person's graduation; if table $People$ contains attribute $years$ instead of $\sigma$, then $\sigma$ is computed on the fly). A full discussion on modeling tuple scores is beyond the scope of this chapter. If all tables were stored in a single DBMS, the marketer would issue the SQL query in Figure 6.2.

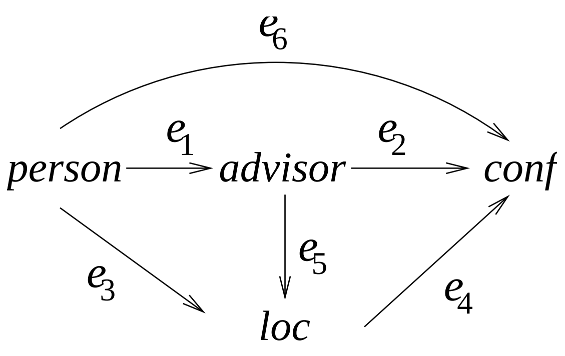


Figure 6.4: Query graph for the example query

**Query graphs** It is easier to visualize this SQL query as the query graph in Figure 6.4. Each edge corresponds to a table, while each node corresponds to an attribute[2]. If two edges share a node, then there is a join on that attribute between the two tables. For example, edge $e_6$ corresponds to table $Research$, and edge $e_3$ to table $Travel$. Edge $e_5$ also corresponds to $Travel$. The reason we represent this table by two edges is that the table appears twice in the query, as $T$ and $T1$.

Nodes connected by a path correspond to a logical 'and' between their corresponding joins. Thus, the path $person$ - $loc$ - $conf$ corresponds to the clause *R.person=T.person and T.loc=C.loc*. Edges emanating from the same node correspond to a logical 'or' between the clauses that start with the corresponding tables. Thus, since edges $e_6$ and $e_3$ start two paths from the same node, the corresponding clauses *(R.conf=C.conf)* and *(R.person=T.person and T.loc=C.loc)* are connected by 'or'.

We use directions on the edges to ensure that certain paths are impossible. For example, the path $person$ - $loc$ - $advisor$ - $conf$ would be a valid path in an undirected graph. However, this would correspond to a clause *(R.person=T.person and T.loc=T1.loc and T1.person=R1.person and R1.conf=C1.conf)* being 'or'-connected to the other conditions. Such a clause breaks the semantics of the SQL query: for $R.person = A$, $T.loc = Shanghai$, and $C.conf = ICDE09$, there are many values $T1.person = B$ that satisfy this clause, because there are many other researchers that are connected to $ICDE09$. However, this should not contribute

[2]We restrict the model to binary tables. Tables with more join attributes can be modeled as multiple binary tables.

to the likelihood that $A$ will attend $ICDE09$. To insure the equivalence between the query semantics and the paths in the query graph, we impose directions on edges. Nevertheless, our methods are directly applicable to undirected graphs, as well.

Finally, in order to fix the source and destination nodes, we use the techniques proposed in [76]. The source attributes are the ones that have selection conditions in the "WHERE" clause, and the destination attributes are the ones that appear on the "SELECT" clause. For instance, the example query above has $person$ in the "WHERE" clause with selection condition and $conf$ in the "SELECT" clause, therefore we fix them as source and destination nodes respectively. For simplicity, we assume there are exactly one source and one destination (otherwise, add new nodes $s$ and $t$; connect $s$ to all sources via edges with scores 1; connect all destinations to $t$ via edges with scores 1).

## 6.3 Definitions

We study join queries of type *SELECT* $\mathcal{L}$ *from* $\mathcal{R}$ *where* $\mathcal{C}$, where $\mathcal{R}$ is a list of tables, $\mathcal{L}$ is a list of attributes from $\mathcal{R}$, and $\mathcal{C}$ is a set of join conditions over attributes from $\mathcal{R}$, connected by and/or operators. For the remainder of this chapter, we assume that the join query is represented as a query graph, as described in the previous section.

Let $G = (V, E)$ be the (directed or undirected) query graph, with *source node* $s$ and *destination node* $t$; $s, t \in V$. Each edge $e \in E$ corresponds to a table accessible via a Web site, and thus has an associated set of tuples denoted $Tup(e)$. For each tuple $\tau$, let $\sigma(\tau) \in [0, 1]$ denotes the score of $\tau$. Similarly, each node $v \in V$ corresponds to an attribute and has an associated domain denoted $Val(v)$. The domain contains all possible values for that attribute, over all the tables that have that attribute. For any edge $e$, if its endpoints are nodes $u$ and $v$, then $Tup(e) \subseteq Val(u) \times Val(v)$.

### 6.3.1 Cost Model

Our goal is to minimize the number of Web accesses necessary to compute the query results. As in [48], we consider two types of probes: *random access probes (RA)* and *sorted access probes (SA)*. We first define them below, and then explain their contribution to the cost function.

In an RA probe, we know the value for at least one position in the tuple, and we ask for all the tuples that match that value, along that edge. An SA probe, on the other hand, returns the tuple with highest score that has not been accessed so far. We use the notations $RA(e)$ and $SA(e)$ to denote random and sorted accesses on edge $e$ respectively.

Whenever a tuple $\tau$ is returned as part of an RA or SA result, we assume that its score $\sigma(\tau)$ is also returned. An RA probe may return more than one tuple. If $k$ tuples are returned, the cost of the operation is $Cost_{RA} + \alpha(k-1)Cost_{RA}$, where $Cost_{RA}$ is the cost of one Web access, and $0 < \alpha < 1$ is a dampening factor. The rationale is that having a Web request processed by a remote site is the main bottleneck, and the number of results returned adds only a small overhead. By contrast, an SA probe only returns one request at a time. However, since these results are accessed sequentially, it is reasonable to assume that multiple results are sent at once, and cached on the query processor's site. Therefore, we assume that $Cost_{SA} = \beta Cost_{RA}$, for some $0 < \beta < 1$.

### 6.3.2 Bindings

We define a *query result* to be a set of tuples, one from each table in the 'FROM' list $\mathcal{R}$, such that the tuples satisfy the conditions in the 'WHERE' clause $\mathcal{C}$. The set of values for the columns in the 'SELECT' list $\mathcal{L}$ can easily be computed from the query result. A brief justification for this definition is provided in Remark 1 at the end of this subsection. This set of tuples induces a binding of all nodes in the graph to some specific values. In addition, it also induces corresponding scores on the edges. Conversely, a binding of nodes to values and edges to scores, if it is consistent with the query conditions, induces a unique query answer (and its score). For the sake of clarity, we therefore refer to query results as *complete bindings*, defined below.

**Definition 2** *Let $G = (V, E)$ be a directed query graph, where $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. A* complete binding *of G is a vector*

$$B = (a_1, \ldots, a_n, \sigma_1, \ldots, \sigma_m),\ a_i \in Val(v_i)$$

*such that, for any edge $e_i = v_j \to v_k$, if the tuple $(a_j, a_k)$ belongs to $Tup(e_i)$ then $\sigma_i = \sigma((a_j, a_k))$; and otherwise, $\sigma_i = 0$. We say that edge $e_i$ is* bound to the tuple *$(a_j, a_k)$, and* *nodes* $v_j$*, resp.* $v_k$*, are* bound to the values $a_j$*, resp.* $a_k$.

Note that we must allow zero-score values on edges in order to model situations in which not all paths can be instantiated. For example, the vector $(A, SIGPOD09, Providence, B, 0, 0.8, 0.9, 0.4, 0.9, 0.7)$ is a complete binding of the query graph in Figure 6.4. Tuple $(A, SIGPOD09)$ is not an instance of table $e_1$. Therefore, $\sigma_1 = 0$. Tuple $(A, Providence)$ is an instance of $e_2$, with score 0.8.

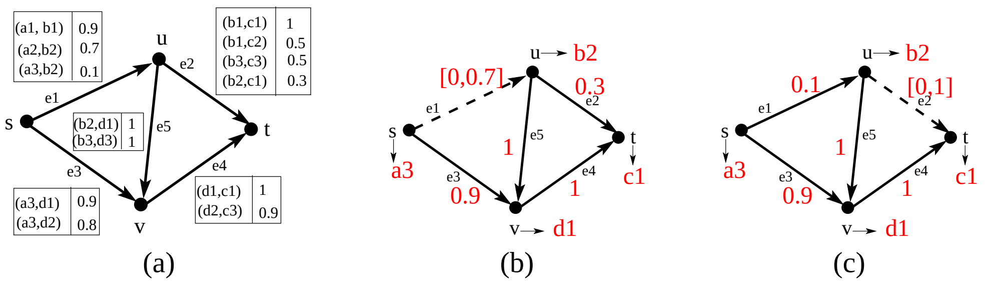


Figure 6.5: Generating bindings for a simplified version of the graph in Figure 6.4 (dashed edges are unbound): (a) the graph and its associated edge tuples and scores; (b), (c) two different partial bindings.

Our branch-and-bound strategy involves exploring and possibly discarding a subset of complete bindings (i.e., complete results) at each step. We represent such subsets as partial bindings (i.e., partial results), defined below.

**Definition 3** *Let* $G = (V, E)$ *be a query graph, where* $V = \{v_1, \dots, v_n\}$ *and* $E = \{e_1, \dots, e_m\}$*. We denote by '*' a new symbol, such that* $* \notin (\cup_{i=1}^{n} Val(v_i))$*. A* partial binding *of* $G$ *is the vector*

$$PB = (b_1, \dots, b_n, [\ell_1, L_1], \dots, [\ell_m, L_m]),\ b_i \in (Val(v_i) \cup \{*\}),$$

*such that for each* $1 \le j \le m$*,* $[\ell_i, L_i] \subseteq [0, 1]$ *and* $[\ell_i, L_i]$ *contains at least one score* $\sigma(\tau)$ *of a tuple* $\tau \in Tup(e_i)$*.*

*For any* $v_i \in V$*, we use* $PB[v_i]$ *to denote the value of* $PB$ *corresponding to* $v_i$ *(i.e.,* $PB[v_i] = b_i$*). Similarly, for any* $e_j \in E$*,* $PB[e_j]$ *denotes the range* $[\ell_j, L_j]$ *corresponding to* $e_j$*.*

Note that, unlike a complete binding, a partial binding allows a node instance $b_i$ to be the new symbol *. This signifies that node $v_i$ has not been bound to any instance from $Val(v_i)$.

For the range of an edge $e_i$, we will only allow two cases: Either $\ell_i = 0 < L_i$, in which case we say that $e_i$ is *unbound*; or $\ell_i = L_i = \sigma(\tau)$, where $\sigma(\tau)$ is the score of a tuple $\tau \in Tup(e_i)$. In the latter case, we say that $e_i$ is *bound* to the tuple $\tau$, and denote it by $e_i \to \tau$.

As we detail in Section 6.4, our algorithm generates new partial bindings $PB'$ from a current partial binding $PB$ using probes on unbound edges $e_i$. In general, in the new partial bindings edge $e_i$ will be bound to one of the tuples $\tau \in Tup(e_i)$ returned by the probe (some exceptions occur for SA probes).

**Executing one edge binding:** We use the notation $PB' = (PB, e_i \to \tau)$ to signify that $PB'$ was created from $PB$ by binding edge $e_i$ to $\tau$. Edge $e_i$ must be unbound in $PB$. More precisely, $PB'$ is computed as follows: $PB'[e_i] = \sigma(\tau)$; if $e_i = v_j \to v_k$ and $\tau = (a, b)$, then $PB'[v_j] = a$ and $PB'[v_k] = b$; all other entries in $PB'$ are the same as in $PB$. This edge binding operation is well-defined only if $\tau$ is *compatible with* $PB$, i.e., $PB[v_j] \in \{a, *\}$ and $PB[v_k] \in \{b, *\}$. In other words, we only execute an edge binding $e_i \to \tau$ if the endpoints of $e_i$ are either unbound, or bound to the same values as in $\tau$.

**EXAMPLE 3:** *Consider the query graph from Figure 6.5(a). A complete binding for this graph is, e.g.,*

$$B = (a_3, b_2, c_1, d_1, 0.1, 0.3, 0.9, 1, 1).$$

*Two partial bindings for the graph are illustrated in Figures 6.5(b) and (c): unbound edges are dashed, while bound ones are solid; ranges/scores are indicated along the edges; and the binding values for nodes are indicated by small arrows. Hence, Figure 6.5(b) illustrates the partial binding*

$$PB_1 = (a_3, b_2, c_1, d_1, [0, 0.7], 0.3, 0.9, 1, 1),$$

*and Figure 6.5(c) corresponds to*

$$PB_2 = (a_3, b_2, c_1, d_1, 0.1, [0, 1], 0.9, 1, 1).$$

*Note that, even though the nodes are bound to the same values in all 3 cases, the bindings are different, because they were generated via different edge bindings. For example,* $B =$

$(PB_1, e_1 \rightarrow (a_3, b_2)) = (PB_2, e_2 \rightarrow (b_2, c_1))$*, but* $PB_1$ *and* $PB_2$ *cannot be generated from each other via edge bindings.*

*An example of invalid edge binding in this figure is* $(PB_1, e_1 \rightarrow (a_2, b_2))$*, since it conflicts with the binding of node* $s$ *to* $a_3$ *in* $PB_1$*.*

Intuitively, a partial binding is a short-hand notation for a subset of complete bindings. It is therefore natural to talk about an inclusion relationship between bindings, as follows.

**Definition 4** *Let* $PB_1$ *and* $PB_2$ *denote two partial bindings, such that*

$$PB_1 = (b_1, \ldots, b_n, [\ell_1, L_1], \ldots, [\ell_m, L_m])$$

$$PB_2 = (c_1, \ldots, c_n, [r_1, R_1], \ldots, [r_m, R_m]).$$

*We say that* $PB_1$ is included in $PB_2$*, and write* $PB_1 \subseteq PB_2$*, if for all* $1 \leq i \leq n$*, either* $c_i = b_i$ *or* $c_i = *$*; and for all* $1 \leq j \leq m$*,* $[\ell_i, L_i] \subseteq [r_i, R_i]$*. If, in particular,* $PB_1$ *is a complete binding and is included in* $PB_2$*, we say that* $PB_1$ belongs to $PB_2$ *and write* $PB_1 \in PB_2$*.*

**Remark 1** *In the example from Section 6.2, there is a unique complete binding for each pair* $(R.person, C.conf)$*. However, this is not usually the case. Suppose that table* $Travel$ *has an extra attribute* $OptionID$*, and that it contains tuples* $t_1$ *and* $t_2$ *as* $(ID1, A, Providence, 0.9)$ *and* $(ID2, A, Providence, 0.88)$*. Then the answer* $(A, SIGPOD09)$ *is obtained via 2 complete bindings* $B_1$ *and* $B_2$*:* $B_1$ *binds edge* $e_2$ *to* $t_1$ *with a score of* $0.9$*, while* $B_2$ *binds it to* $t_2$ *with a score of* $0.88$*. Returning* $B_1$ *and* $B_2$ *as separate results gives the marketer additional information; e.g., he may have airline clients interested in it. Moreover, our algorithms can still be adapted to return just* $(A, SIGPOD09)$*, with score* $score(B_1)$*, i.e., the maximum score of all complete bindings generating the pair.*

### 6.3.3 Computing Scores of Bindings

Let $G = (V, E)$ be a query graph with specified *source node* $s$ and *destination node* $t$; $s, t \in V$. Graph $G$ can be seen as a communication network, in which $s$ transmits a signal that $t$ must receive. The signal can travel along any edge. An edge $e \in E$ fails (gets disconnected) with probability $1 - \pi(e)$, where $\pi(e)$ is the *success probability* of $e$. The probabilities of different edges are assumed to be independent. The probability that a path $P = e_1 e_2 \ldots e_k$ succeeds, i.e., that the signal travels from one end to the other of $P$, is therefore $\pi(P) = \Pi_{i=1}^{k} \pi(e_i)$. The *reliability* of network $G$ is the probability that at least one of the paths between $s$ and $t$ succeeds; equivalently, it is the probability that $G$ remains connected. Given the equivalence between the boolean conditions in a SQL query $Q$, and the structure of its corresponding query graph $G$, we propose scoring the answer to $Q$ as the network reliability of $G$. More precisely,

**Definition 5** *Let $B = (a_1, \ldots, a_n, \sigma_1, \ldots, \sigma_m)$ be a complete binding of $G$. For any edge $e_i$, we define its success probability as $\pi(e_i) = \sigma_i$ (recall that $\sigma_i \in [0, 1]$). We define the score of $B$, denoted $score(B)$, to be the reliability of network G under these edge probabilities.*

For partial bindings

$$PB = (b_1, \ldots, b_n, [\ell_1, L_1], \ldots, [\ell_m, L_m]),$$

we will compute a range of scores $[min(PB), max(PB)]$ as follows: Let the *minimum network of $PB$*, resp. *maximum network of $PB$*, be the network $G$ where the success probability of any edge $e_i$ is defined as $\pi(e_i) = \ell_i$, resp. $\pi(e_i) = L_i$. Then $min(PB)$, resp. $max(PB)$, is the reliability of the minimum, resp. maximum, network of $PB$. The following result will be used in Section 6.4 to explain our strategy for choosing edge probes.

**Proposition 1** *Let $PB_1, PB_2$ be two partial bindings such that $PB_1 \subseteq PB_2$. Then:*

*(i) $[min(PB_1), max(PB_1)] \subseteq [min(PB_2), max(PB_2)]$. In particular, if $PB_1$ is a complete bindings, then $score(PB_1) \in [min(PB_2), max(PB_2)]$.*

*(ii) If there exists at least one path $P$ such that all edges of $P$ are bound to non-zero values in $PB_1$, but at least one such edge is unbound in $PB_2$, then $min(PB_1) > min(PB_2)$. If no such path exists, then $min(PB_1) = min(PB_2)$.*

**Proof 1** *Since $PB_1 \subseteq PB_2$, claim (i) is immediate.*

*To see why (ii) is true, suppose first that there exists a path $P$ satisfying the conditions as stated. Let $e_i \in P$ be an edge which is unbound in $PB_2$. This implies that $PB_2[e_i] = [0, L_i]$, so in the minimum network of $PB_2$, $\pi(e_i) = 0$. Therefore, path $P$ always fails, so it contributes nothing to the network reliability $min(PB_2)$. By contrast, since $PB_1[e_i] = \sigma_i > 0$ for all* *edges $e_i \in P$, it follows that $\pi(P) > 0$ in the minimum network of $PB_1$, so $P$ contributes towards the network reliability $min(PB_1)$. Because $PB_1 \subseteq PB_2$, any other path $P'$ has at least the same probability in the minimum network of $PB_1$ as in the minimum network of $PB_2$. This implies that $min(PB_1) > min(PB_2)$. For the last claim, if no path $P$ satisfies the stated conditions, it follows that: either $P$ contains an unbound edge in both $PB_1$ and $PB_2$; or all edges of $P$ are bound in both $PB_1$ and $PB_2$. In the first situation, $\pi(P) = 0$ in both minimum networks, while in the second situation, $\pi(P)$ is the same in both minimum networks. Since this is true for all paths $P$, $min(PB_1) = min(PB_2)$.*

Computing the reliability of a general network is NP-Hard [68]. The Monte-Carlo algorithm in [36] approximates the reliability of a network with arbitrarily high precision. Multiple iterations are executed, and the precision increases with the number of iterations. Note that one could also compute the network reliability in a deterministic way by the inclusion/exclusion formula over paths. However, the complexity of this approach grows exponentially with the number of paths, and quickly becomes impractical. Therefore, we will employ the Monte-Carlo algorithm for computing the scores of bindings, and assume that enough iterations are executed so that all approximation errors are negligible.

## 6.4 Top-k Algorithm

In this section, we present our algorithm for efficiently computing the top-$k$ complete bindings of a query graph. Our cost model assumes that tuple scores are stored remotely and are expensive to access. To this end, we design an efficient edge probing strategy that computes the top-$k$ bindings based on a subset of tuple scores.

Our strategy generalizes Fagin's Threshold Algorithm (TA) [25]. The TA algorithm assumes that each object in a database has $m$ attributes stored in $m$ lists. The score of an object is computed using some monotonic aggregation function $f$, such as min or average. The algorithm works by doing sorted access in parallel to each of the $m$ sorted lists. For each object $B$ that is seen under sorted access, TA then does a random access to the other lists to find the corresponding scores for object $B$ and computes its overall score $f(B)$. Only the $k$ objects with highest overall score are stored, at any given time. TA defines the *threshold value* $\tau$ to be

$f(\underline{x_1}, \ldots, \underline{x_m})$ (where $\underline{x_i}$ is the last object seen under sorted access on list $i$) and halts when the $k$ highest scores are at least equal to $\tau$.

In our setting, the objects correspond to complete bindings, and the $m$ attributes of an object $B$ correspond to the $m$ edge bindings in $B$. The value of an attribute is the score of the corresponding edge binding. The monotonic function $f$ is $score(B)$. However, a direct application of the TA algorithm is impossible in our model, as we explain below. Suppose we started by doing a sorted access in parallel on all edges, i.e., an SA probe on each edge. For each binding $e_i \to SA(e_i)$ that is retrieved under sorted access, we would need to know the object $B$ to which it belongs. However, in our case, one edge binding may be part of many complete bindings, and we have no way of identifying them at this point. Even if an edge binding occurred in only one complete binding $B$ for which we could somehow obtain an identifier, the TA algorithm would still require random accesses on all other edges (using $B$'s id) to find all the edge bindings in $B$ and their scores. Clearly, this would lead to many expensive edge probes.

Instead, our approach modifies the TA method in several crucial ways: We maintain sets of objects together, and compute lower and upper bounds for the scores of all objects in a set. Each such set has a succint representation as a partial binding. We may store more than $k$ (complete or partial) bindings at any given point. While we still do sorted access in parallel over all edges, we do not follow such a step by compulsory RA probes on all edges. Instead, we design and study several strategies for deciding what RA probes to execute.

Throughout this section, we use the query graph from Figure 6.5(a) to illustrate these ideas. This graph is obtained from the query graph in Figure 3, where edge $e_6$ was removed for simplicity. As mentioned above, we assume that each edge in the graph has a sorted list of tuple scores, in descending order of scores. Ties are broken in an arbitrary but fixed manner. We say that the topmost tuple has level 1, the next tuple has level 2, a.s.o. We will maintain a global level $s$, which is originally set to 0, i.e., the pointer in each sorted list lies above the first tuple. To execute SA probes in parallel on all edges, we increment $s$ and access the tuple at level $s$ on each edge. If an edge has fewer than $s$ levels, then the result of its SA probe is undefined, and no further SA probes are executed.

Our algorithm employs parallel SA probes to generate bindings in which all nodes and

| |
|---|
| $PB_{*,0} = (*, *, *, *, [0,1], [0,1], [0,1], [0,1], [0,1])$ |
| $PB_{*,1} = (*, *, *, *, [0,0.9], [0,1], [0,0.9], [0,1], [0,1])$ |
| $PB_{*,2} = (*, *, *, *, [0,0.7], [0,0.5], [0,0.8], [0,0.9], [0,1])$ |
| $PB_{*,3} = undefined$ |

Table 6.1: AllStar bindings for the graph in Figure 6.5(a).

edges are unbound, but edge ranges are progressively tighter. We call such bindings *AllStar*. More precisely, the *AllStar of level* $s$ is defined as $PB_{*,s} = (*, \ldots, *, [0, \sigma_1^s], \ldots, [0, \sigma_m^s])$, where $\sigma_i^s$ is the score of the tuple on level $s$ in the sorted list of $e_i$. For $s = 0$, $PB_{*,0} = (*, \ldots, *, [0,1], \ldots, [0,1])$.

**EXAMPLE 4:** *The graph in Figure 6.5(a) has AllStar bindings of levels 0, 1, and 2. They are depicted in Table 6.1.*

## 6.5 Top-k Algorithm

Our overall approach is described in Algorithm 3. It takes as input a query graph $G$, which comprises, in addition to its node and edge structure, information about the data sources from which edge tuples can be retrieved (via edge probes).

The algorithm maintains a set of partial bindings $\mathcal{S}$, and a set of complete bindings $\mathcal{T}$. Initially, $\mathcal{S} = \{PB_{*,0}, PB^1, \ldots, PB^k\}$, where $PB^i$ is the partial binding having the source node bound to the $i$th value in PREDEF-SET, and all other nodes and edges unbound; and $\mathcal{T} = \emptyset$. As the algorithm executes the *while* loop, partial bindings from $\mathcal{S}$ are replaced by new bindings with fewer unbound edges. Eventually, some of the partial bindings in $\mathcal{S}$ become complete bindings, and may be added to $\mathcal{T}$. The set $\mathcal{T}$ stores at most $k$ complete bindings at any given time, and they are the bindings with highest scores. The algoritm terminates when $|\mathcal{T}| = k$. It may also terminate sooner if $\mathcal{S}$ becomes empty, which occurs if the query graph has fewer than $k$ complete bindings (Step 25).

During each iteration, we select the binding $PB'$ with maximum upper bound $max(PB')$. If $PB'$ is a complete binding, we add it to $\mathcal{T}$. Otherwise, $PB'$ is replaced with one or more bindings $PB''$ such that $PB'' \subseteq PB'$ (when adding $PB''$ to $\mathcal{S}$, we also compute $[min(PB''), max(PB'')]$). Each such computation requires either a round of parallel SA

**Algorithm 3** Finding top-$k$ Complete Bindings

top-$k$($G$)[H]

```
1:  $\mathcal{S} \leftarrow \{PB_{*,0}, PB^1, \ldots, PB^k\}$
    $\{$where $PB^i = (Val_i, *, \ldots, *, [0,1], \ldots, [0,1])\}$
    $\{Val_i$: $i$th value in PREDEF-SET$\}$
2:  $\mathcal{T} \leftarrow \emptyset$
3:  $s \leftarrow 0$ $\{$level of SA probes$\}$
4:  **while** $|\mathcal{T}| < k$ **do**
5:    pick $PB' \in \mathcal{S}$ s.t. $max(PB') = \max_{PB \in \mathcal{S}} max(PB)$
6:    delete $PB'$ from $\mathcal{S}$
7:    **if** $PB'$ is complete binding **then**
8:      $\mathcal{T} \leftarrow \mathcal{T} \cup \{PB'\}$
9:    **else if** $PB'$ is AllStar **then**
10:     $s \leftarrow s + 1$; do SA probes of level $s$ on all edges
11:     **if** all SA probes are defined **then**
12:       $\mathcal{S} \leftarrow \mathcal{S} \cup \{PB_{*,s}\}$
13:       $\mathcal{S} \leftarrow \mathcal{S} \cup \{(PB_{*,s}, e_i \rightarrow SA(e_i, s))\}$, $\forall e_i$ : edge
14:     **end if**
15:   **else**
16:     choose unbound edge $e$: $PB'[e] = [0, L(e)]$
17:     do RA probe on $e$
18:     **for** each tuple $\tau \in RA(e)$ **do**
19:       **if** $\sigma(\tau) \leq L(e)$ AND $(PB', e \rightarrow \tau) \notin \mathcal{S}$ **then**
20:         $\mathcal{S} \leftarrow \mathcal{S} \cup \{(PB', e \rightarrow \tau)\}$
21:       **end if**
22:     **end for**
23:   **end if**
24:   **if** $\mathcal{S} == \emptyset$ **then**
25:     return $\mathcal{T}$
26:   **end if**
27: **end while**
28: return $\mathcal{T}$
```

probes, or an RA probe, depending on whether or not $PB'$ is AllStar. We explain each case below.

*Replacing an AllStar (Steps 9-14)*: We first increment the level $s$ and execute all SA probes in parallel, as explained above. If at least one probe is undefined, then we do not generate any new bindings. In this case, no subsequent iteration will enter Step 10 (note that $PB'$ is deleted from $\mathcal{S}$ in Step 6). If, however, all probes are valid, we add the new AllStar to $\mathcal{S}$. We also bind each edge $e_i$ in turn to its tuple of level $s$, i.e., to $SA(e_i, s)$. In total, we add exactly $|E| + 1$ new partial bindings in Steps 12 and 13. It is trivial to verify that all these new bindings are included in $PB'$. We make the observation that the set $\mathcal{S}$ contains exactly one AllStar as long as the algorithm passes the test in Step 11, and no AllStar thereafter.

**EXAMPLE 5:** *Table 6.2 shows three of the six bindings added to $\mathcal{S}$ during the first iteration, as a result of selecting $PB' = PB_{*,0}$ in Step 5. Refer also to the graph in Figure 6.5(a).*

**Remark 2** *In Step 13 of Algorithm 3, we could also take an "eager" approach, by attempting to create partial bindings in which several compatible edges are simultaneously bound. In Example 5, such a binding could be $PB_{1,2} = (PB_{*,1}, e_1 \to (a_1, b_1), e_2 \to (b_1, c_1))$, which is valid, since both edge bindings require the value in node $u$ to be $b_1$. Instead, we ignore this possibility, and allow the algorithm to generate $PB_{1,2}$ in Step 20 of a later iteration, either as $(PB_1, e_2 \to (b_1, c_1))$, or as $(PB_2, e_1 \to (a_1, b_1))$. Suppose that $PB_{1,2}$ is generated as $(PB_1, e_2 \to (b_1, c_1))$, during the iteration for which $PB_1$ is chosen in Step 5. This will require executing the RA probe $RA(e_2, u \to b_1)$ in Step 17 of that iteration. Hence, we will access the tuple $(b_1, c_1)$ for a second time (the first time was as the result of the probe $SA(e_2, 1)$.) Therefore, we appear to be inefficient when it comes to minimizing the number of edge probes.*

*There are two reasons for which we choose this "lazy" approach to edge binding in Step 13. First, notice that executing the RA probe $RA(e_2, u \to b_1)$ in a subsequent iteration is not superfluous, as this probe also returns the tuple $(b_1, c_2)$, which is not returned by the probe $SA(e_2, 1)$. In fact, if after the first iteration $\mathcal{S}$ contained only $PB_{1,2}$, but not $PB_1$, then we could not later generate any complete bindings in which $e_1 \to (a_1, b_1)$ and $e_2 \to (b_1, c_2)$. But discarding such complete bindings at this point is incorrect, as we cannot guarantee that they are not among the top-$k$. The correct alternative is to put both $PB_{1,2}$ and $PB_1$ in $\mathcal{S}$,*

| |
|---|
| $PB_{*,1} = (*, *, *, *, [0, 0.9], [0, 1], [0, 0.9], [0, 1], [0, 1])$ |
| $PB_1 = (PB_{*,1}, e_1 \rightarrow (a_1, b_1)) = (a_1, b_1, *, *, 0.9, [0, 1], [0, 0.9], [0, 1], [0, 1])$ |
| $PB_2 = (PB_{*,1}, e_2 \rightarrow (b_1, c_1)) = (*, b_1, c_1, *, [0, 0.9], 1, [0, 0.9], [0, 1], [0, 1])$ |

Table 6.2: Bindings computed during the first iteration for the graph in Figure 6.5(a).

*thus increasing the size of $\mathcal{S}$. This is a non-trivial problem: In the extreme case, all $|E|$ edge bindings $e_i \rightarrow SA(e_i, 1)$ may be mutually compatible (instead of just $e_1$ and $e_2$). In such a case, the eager approach would have to add $2^{|E|}$ partial bindings to $\mathcal{S}$ in order to maintain correctness (each of these bindings would leave a different subset of edges unbound).*

*Second, note that if $PB_1 \in \mathcal{S}$, it may still be selected in Step 5 of a later iteration, which may still trigger the RA probe $RA(e_2, u \rightarrow b_1)$. We conclude that the lazy approach is in fact more efficient than the eager one.*

*Replacing other bindings (Steps 15-23)*: For ease of presentation, we have omitted some details in Step 16 of Algorithm 3. More precisely, the edge $e$ chosen in this step must have at least one of its endpoints bound to a value, since otherwise we cannot execute an RA probe. Suppose that $e = u \rightarrow v$. If both $u$ and $v$ are bound to values $a$, resp. $b$, then the RA probe asks whether the tuple $(a, b)$ exists on edge $e$. If it does, then $e$ is bound to the score $\sigma((a, b))$; otherwise, $e$ is bound to 0; the bindings of $u$ and $v$ remain the same in either case. If only one endpoint of $e$ is bound, it is possible that the RA probe returns multiple tuples. In that case, we bind $e$ in turn to each such tuple. In general, there are multiple unbound edges with one bound endpoint. We choose one randomly from among them.

The resulting new bindings are added to $\mathcal{S}$, provided that they satisfy the conditions in Step 19. We discuss the second condition first. Clearly, this condition ensures that we keep $\mathcal{S}$ as small as possible, and that we do not run unnecessary iterations by selecting duplicate bindings in Step 5. Moreover, it also ensures that we do not double-count complete bindings in the result set $\mathcal{T}$. The test can be executed very efficiently by keeping a hash table on the bindings in $\mathcal{S}$. The next example illustrates how duplicates may arise.

**EXAMPLE 6:** *Consider two different iterations over the graph from Figure 6.5(a): In the first iterations, we choose $PB' = PB_1$ in Step 5, while in the other iteration, we choose*

*$PB' = PB_2$ in Step 5; $PB_1$ and $PB_2$ are the bindings defined in Table 6.2. Suppose that for $PB_1$, we choose the edge $e = e_2$ in Step 16, and for $PB_2$ we choose $e = e_1$ in Step 16. Table 6.3 shows the bindings generated during Steps 15-23 of each iteration. Since $PB_3$ is generated as a duplicate during the second iteration, it is not added to $\mathcal{S}$ again.*

| | |
|---|---|
| Step 5: | $PB' = PB_1 = (a_1, b_1, *, *, 0.9, [0,1], [0,0.9], [0,1], [0,1])$ |
| Steps 15-23: | $PB_3 = (a_1, b_1, c_1, *, 0.9, 1, [0,0.9], [0,1], [0,1])$ |
| | $PB_4 = (a_1, b_1, c_2, *, 0.9, 0.5, [0,0.9], [0,1], [0,1])$ |
| Step 5: | $PB' = PB_2 = (*, b_1, c_1, *, [0,0.9], 1, [0,0.9], [0,1], [0,1])$ |
| Fails Step 19: | $PB_3 = (a_1, b_1, c_1, *, 0.9, 1, [0,0.9], [0,1], [0,1])$ |

Table 6.3: Bindings generated in Steps 15-23 of two different iterations, for the graph in Figure 6.5(a): $PB_3$ is generated twice, but only added once to $\mathcal{S}$.

We now discuss the first condition in Step 19. Recall that we wish to generate new bindings $PB''$ from $PB'$ such that $PB'' \subseteq PB'$. The test $\sigma(\tau) \leq L(e)$ ensures this for all bindings generated in Step 20. The following example illustrates a situation when the test fails, i.e., $\sigma(\tau) > L(e)$.

**EXAMPLE 7:**

*Consider the iteration over the graph from Figure 6.5(a), in which Step 5 chooses $PB' = PB_5$ as depicted in Table 6.4. (Binding $PB_5$ was added to $\mathcal{S}$ in Step 13 of an earlier iteration, since $PB_5 = (PB_{*,2}, e_2 \rightarrow (b_1, c_2))$.) Suppose that for $PB_5$, we choose the edge $e = e_1$ in Step 16. Then $L(e_1) = 0.7$, since the range for $e_1$ is $PB_5[e_1] = [0, 0.7]$. The RA probe $RA(e_1, u \rightarrow b_1)$ returns the tuple $(a_1, b_1)$, with score $0.9 > 0.7$. Therefore, binding $PB_6$ is not added to $\mathcal{S}$. Note that $PB_6 \subseteq PB_4$, where $PB_4 \in \mathcal{S}$ is defined as in Table 6.3. Hence, all complete bindings contained in $PB_6$ are also contained in $PB_4$, and we do not miss any information by ignoring $PB_6$. On the contrary, we eliminate a redundant partial binding.*

| $PB' = PB_5 = (*, b_1, c_2, *, [0,0.7], 0.5, [0,0.8], [0,0.9], [0,1])$ |
|---|
| $PB_6 = (PB_5, e_1 \rightarrow (a_1, b_1))$: $(a_1, b_1, c_2, *, 0.9, 0.5, [0,0.8], [0,0.9], [0,1])$ |

Table 6.4: Enforcing the inclusion property for the graph in Figure 6.5(a): $PB_6 \not\subseteq PB_5$, so $PB_6$ is not added to $\mathcal{S}$.

To prove that Algorithm 3 works correctly we need the following two lemmas.

**Lemma 1** *Let $B$ be a complete binding added to $\mathcal{T}$ in some iteration $i$. Then $score(B) \geq max(PB)$ for any partial binding $PB$ that belongs to $\mathcal{S}$ at the end of any iteration $j$, $j \geq i$.*

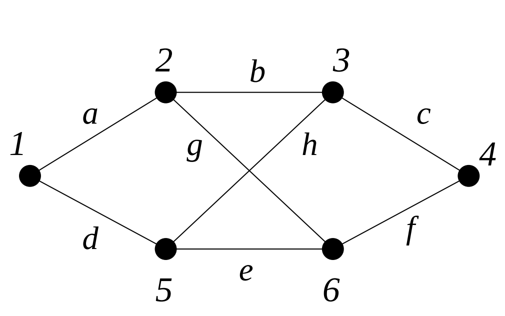
**Lemma 2** *Let $B$ be a complete binding that is never added to $\mathcal{T}$. Then at the end of each iteration in Algorithm 3, there exists at least one binding $PB \in \mathcal{S}$ such that $B \in PB$ (and therefore, $score(B) \in [min(PB), max(PB)]$).*

**Proof 2** Lemma 1*: We use induction on iteration $j$. For $j = i$: $B$ is added to $\mathcal{T}$ if and only if $B$ is selected in Step 5, so $score(B) = max(B) \geq max(PB)$ for any $PB$ that belongs to $\mathcal{S}$ in iteration $i$. Suppose the claim is true for some iteration $j$. In iteration $j+1$, the only new partial bindings $PB''$ in $\mathcal{S}$ are those generated either in Steps 9-14, or in Steps 15-23, from the binding $PB'$ chosen in Step 5. As discussed above, $PB'' \subseteq PB'$, which implies $max(PB'') \leq max(PB')$. Since $PB'$ belongs to $\mathcal{S}$ after iteration $j$, $max(PB') \leq score(B)$, and the claim follows.*

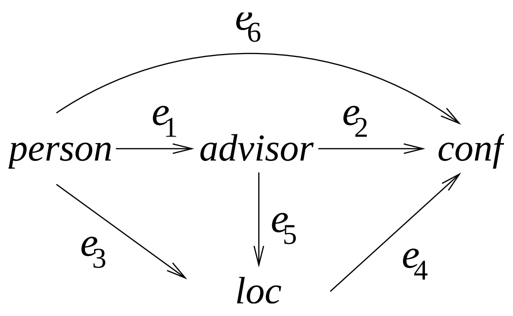
**Proof 3** Lemma 2: *Each edge $e_i$ in $B$ is bound to a tuple $\tau_i \in Tup(e_i)$, with tuples on adjacent edges having compatible node bindings. Let $s_i$ be the level of tuple $\tau_i$ in the sorted list on edge $e_i$. Without loss of generality, assume that $s_1 \leq \ldots \leq s_m$. Then along each edge $e_i$, any tupple on a level $s \leq s_1 - 1$ has score at least as large as $\sigma(\tau_i)$. We deduce that $B \in PB_{*,s}$ for all $s \leq s_1 - 1$. Moreover, the algorithm passes the test in Step 11 during any iteration prior to choosing $PB' = PB_{*,s_1-1}$ in Step 5. Therefore, $\mathcal{S}$ contains one $PB_{*,s}$, with $s \leq s_1 - 1$, during all such iterations, (If the algorithm returns without ever choosing $PB_{*,s_1-1}$ in Step 5, then our claim holds).*

*Once $PB_{*,s_1-1}$ is chosen in Step 5, Steps 9-14 are executed. The test in Step 11 is still true, since there exist tuples $\tau_i$ at levels $s_i \geq s_1$ on all edges $e_i$. Therefore, $PB_1 = (PB_{*,s_1}, e_1 \rightarrow \tau_1)$ is added to $\mathcal{S}$. Note that $PB_1$ binds edge $e_1$ to tuple $\tau_1$, the same as $B$. For all $i \geq 2$, $PB_1[e_i] = [0, L_{s_1}(e_i)]$, where $L_{s_1}(e_i)$ is the score of the tuple on level $s_1$ in $e_i$. Since $\tau_i$ has level $s_i \geq s_1$, it follows that $\sigma(\tau_i) \in [0, L_{s_1}(e_i)]$. We deduce that $B \in PB_1$.*

*The binding $PB_1$ remains in $\mathcal{S}$ until $PB_1$ is chosen in Step 5 of a later iteration. Then, Steps 15-23 are executed. Let $e_k$ denote the edge chosen in Step 16; $e_k$ must be adjacent to $e_1$, so we can do an RA probe. Since $\tau_k$ is compatible with $\tau_1$, tuple $\tau_k$ is among those returned by the RA probe. Moreover, $\sigma(\tau_k) \leq L_{s_1}(e_k)$, as discussed above. Therefore, $PB_2 = (PB_{*,s_1}, e_1 \rightarrow$*



(a) Synthetic join graph



(b) Real-life query graph

Figure 6.6: Graphs used in experiments

*$\tau_1, e_k \to \tau_k$) is added to $\mathcal{S}$, and $B \in PB_2$. We can now repeat this argument with $PB_2$ instead of $PB_1$. By induction, we show that after any iteration there exists $PB_r \in \mathcal{S}$ with $r$ bound edges, $r \leq m$, such that $B \in PB_r$. If $r = m$ and $PB_m = B$ is added to $\mathcal{S}$, then it is never deleted, since $B$ is never selected in $\mathcal{T}$.*

Let $B$ be a complete binding. We claim that if $B \notin \mathcal{T}$ at the end of Algorithm 3, then all complete bindings of $\mathcal{T}$ have scores larger or equal to $score(B)$. Let $B' \in \mathcal{T}$ be an arbitrary complete binding. Since $B \notin \mathcal{T}$, Lemma 2 implies that after the *last* iteration, $\mathcal{S}$ contains a partial binding $PB$ such that $B \in PB$. Therefore, $score(B) \leq max(PB)$. By Lemma 1, $max(PB) \leq score(B')$. Hence, $score(B) \leq score(B')$, and this is true for any $B' \in \mathcal{T}$. We conclude with the following.

**Theorem 1** *For any query graph $G$ that admits at least $k$ complete bindings, the set $\mathcal{T}$ returned by algorithm top-$k$($G$) contains the top-$k$ complete bindings of $G$.*

## 6.6 Experimental Evaluation

In this section we report the results of the extensive experimental study we conducted to evaluate the benefits of our approach for various query graphs and data distributions. We implemented our method using Java with SDK 1.5 and ran experiments on a CentOS machine with 3.0 GHz Intel Xeon CPU and 16 GB RAM.

### 6.6.1 Experiment setup

We implemented Algorithm 3, which throughout this section is referred as the `SMART` method. In all experiments, PREDEF-SET is the entire domain of the source attribute. We also implement a rank-join [34] based approach (`RJ`) as follows: The rank-join algorithm is first applied to each join path to generate the top-$k$ join results with the scoring function being the product of all edge scores. We then apply the rank-join algorithm to the graph treating each path as data sources to produce the overall top-$k$ join results with the scoring function being the network reliability. Note that we extend the original rank-join algorithm to consider random access as well as sorted access. We do not compare with the naive approach which instantiates and sorts all join results because both approaches we study are orders of magnitude better.

We consider various graphs in our experiments. We evaluate our approach using both synthetic and real world datasets (the motivating example). We show experimental results for one synthetic join graph (see Figure 6.6(a)), and for the join graph over real world datasets from Figure 6.6(b). For synthetic datasets, we consider various types of data distribution (uniform v.s. skewed, uncorrelated v.s. correlated). We evaluate the performance by counting the number of SA and RA probes, as defined in Section 6.3.1. We set $\alpha$=0.1 and $\beta$=0.1 and report $Join\ Cost\ = \sum_{RA\ probe} Cost_{RA} + \sum_{SA\ probe} Cost_{SA}$.

### 6.6.2 Graphs and Datasets used in experiments

For testing purposes, we created two different graphs, in order to study the effect of various graph properties on the efficiency of each method. Figure 6.6 shows the two graphs used in our experiments (with numbers annotating nodes and letters annotating edges). In the synthetic graph, we assume the leftmost and the rightmost nodes are the source and destination nodes, respectively. Rather than assigning directions to edges in some arbitrary manner, we choose to use undirected edges. This is because the number of undirected paths between the source and destination is higher than the number of directed paths, making each instance more challenging. We want to point out, however, that our methods are directly applicable to both directed and undirected graphs.

The synthetic graph has 8 distinct paths between the source and destination nodes, such as

$a - b - c$ and $a - g - e - h - c$. It also has 9 minimum cuts; for instance, ($a$, $d$) or ($c$, $h$, $d$). We list the number of nodes, edges, paths and cuts of the two graphs in Table 6.5.

| | Nodes | Edges | Paths | Cuts |
|---|---|---|---|---|
| Synthetic | 6 | 8 | 8 | 9 |
| Real-world | 4 | 6 | 4 | 4 |

Table 6.5: Graph Statistics

We test our algorithm using both synthetic and real world dataset.

**[Synthetic Dataset]:** We generate a variety of datasets for our experiments, which model different types of real-life instances. For each edge in one of the three graphs, we must generate tuples and their corresponding scores. Let ($v_i$, $v_j$, $score$) denote a scored tuple, where $v_i$ and $v_j$ represent the values of the tuple corresponding to the end nodes of its edge, and $score$ is its score. Each tuple may join with multiple tuples on other edges. In our dataset, we set the number of tuples on each edge to 200 and the average fan-out of each tuple to 4. The tuple scores are generated randomly, as explained below. We are interested in studying the effect of the following two parameters on the efficiency of the methods:

- **Uniform vs. Skewed score distribution** We generate two datasets: In the first dataset, scores on an edge are drawn from the uniform distribution on $[0, 1]$. In the second dataset, scores on an edge follow the Zipf's distribution [84]. With a traditional Zipf's distribution ($s$ = 1), the tuple score is the inverse of its rank.

- **Edge-Correlated vs. Uncorrelated scores** Tuples that join, from adjacent edges, may or may not have correlated scores. We test the performance of our approach in both scenarios. For correlated datasets, we pick a join path for which a high-score tuple from one edge implies high scores of the join partners from other edges. We limit the correlations to be among the top few (10%) tuples on the selected path.

**[Real-world Dataset]:** We use the motivating example discussed in Section 6.2 for the real-world experiment (Figure 6.6(b)). In such a query, we are trying to find the top-$k$ bindings (*person, location, advisor, conference*). In particular, edge scores are computed as follows.

- The scores of edges $e_1$ and $e_5$ are computed based on the researcher's papers accepted by the conference. For each paper, the researcher gets a score of 1 divided by the number of authors of the paper. For example, a researcher gets a score of 0.7 if he has two papers with 2 and 5 authors accepted by the conference. Since this score can reach a value greater than 1, we set an upper bound of 0.9.

- The scores of edge $e_2$ and $e_6$ are computed as 100 divided by the distance (in miles) between the researcher and the conference location, with an upper bound of 0.7.

- We assign a score between 0.3 to 0.9 to edge $e_3$ based on the conference reputation.

- The relation score between a researcher and his or her advisor (edge $e_4$) is based on the graduation year: it gets a score of 0.8 when the researcher was still under supervision and decrease by a factor of 2 every year after graduation.

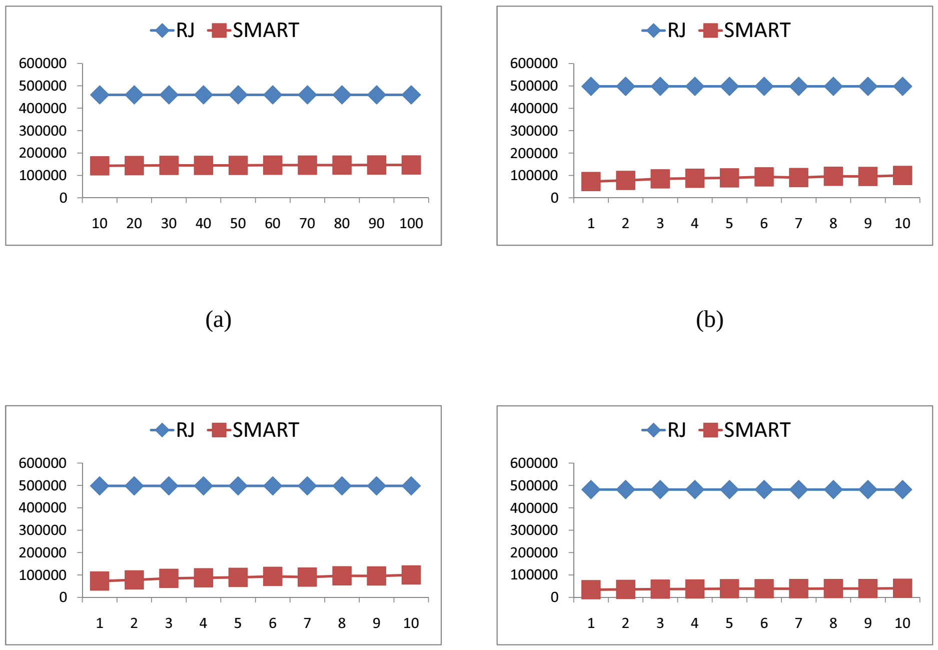
We extracted data from a snapshot of the DBLife dataset , which contains the publication and conference information up to the year of 2006. In order to find genealogy information of researchers, we use the data from the AI Genealogy Project[3] , which provides genealogy information for researchers in AI area. By corroborating the data from AI Genealogy and DBLife, we were able to check out 59 AI researchers, as well as their advisors. We manually retrieved the affiliation of the researchers and conference locations and computed the distance between researchers and conferences for edge $e_2$ and $e_6$. Our real world dataset[4] contains information for 91 researchers and 110 conferences.

### 6.6.3 Uniform Datasets

Figure 6.7(a) shows the Join Cost for the `SMART` and `RJ` methods for the uniform uncorrelated dataset. The x-axis is the number of top-$k$ answers computed. We vary $k$ from 10 to 100. As shown, the `SMART` method clearly outperforms the `RJ` method in all four distributions. In addition, the cost of the `RJ` method is the same over all $k$ values. This can be explained as follows. First of all, since multiple paths may share the same edge and the `RJ` method is applied

[3]http://aigp.eecs.umich.edu/

[4]http://paul.rutgers.edu/∼alexng/dataset.txt



(a) (b)

(c) (d)

Figure 6.7: Cost of top-$k$ join queries for synthetic join graph: (a) Uniform Uncorrelated; (b) Skewed Uncorrelated; (c) Uniform Correlated; (d) Skewed Correlated

to each path of the graph, it incurs cost on the same edge repeatedly (*e.g.*, path $a \rightarrow b \rightarrow c$ and $a \rightarrow g \rightarrow f$ share edge $a$). More importantly, the RJ method computes the top-$k$ result on the path level, making it difficult to decrease the *threshold* value [34]. Assume RJ joins path $p_1$ and $p_2$ and it computes the threshold value as $max(f(E_{top}^{(1)}, E_{current}^{(2)}), f(E_{current}^{(1)}, E_{top}^{(2)}))$, where $E_{top}^{(i)}$ and $E_{current}^{(i)}$ refer to the edge scores of the top-1 and current join result on path $p_i$, and $f$ is the computation of network reliability. Even if the current join result on path $p_i$ has a low score, it could still have high scores on a few edges along the path, making the score of the overall join result high. In fact, we observe in the experiments that even the top-1 join query requires the RJ method to retrieve all join results on each path, which explains why the RJ method has the same cost over all $k$ values. Compared with the RJ method, the SMART method reduces the cost by 68% on average.

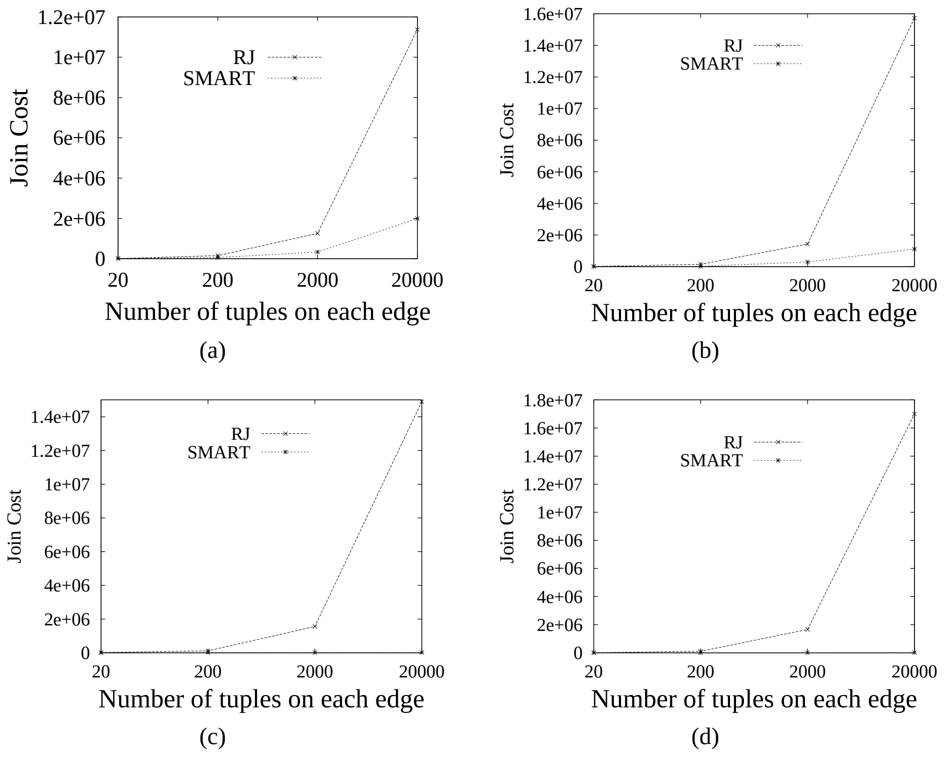


Figure 6.8: Cost of top-$k$ join queries for the large scale dataset: (a) Uniform uncorrelated; (b) Uniform correlated; (c) Skewed uncorrelated; (d) Skewed correlated

### 6.6.4 Skewed and Correlated Datasets

Figure 6.7(b), 6.7(c) and 6.7(d) show the performance comparison for skewed and correlated datasets. As shown, the performance gain of the `SMART` method magnifies as the datasets have skewed and correlated distribution. The `RJ` method performs similarly over skewed, correlated, and uniform datasets, largely due to the fact that it has to instantiate all the join results on each path. By contrast, the `SMART` method performs better over the skewed (32%) and correlated dataset (24%), versus the uniform dataset. We attribute this cost reduction to the fact that in the skewed dataset the tuple scores drop faster, and thus the SA probes could effectively reduce the upper bound of unseen bindings. For the correlated dataset, our `SMART` method benefits by identifying early a few partial bindings instantiated from the correlated path edges that are likely to have very high scores.

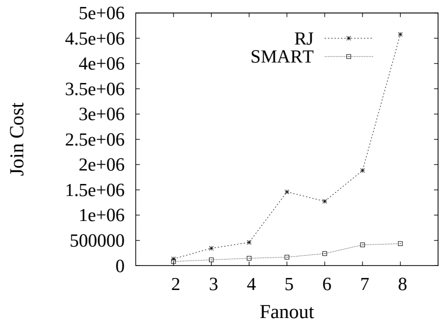


Figure 6.9: Cost of top-$k$ join queries as a function of fanout

### 6.6.5 Large Datasets

In previous section, we show the performance of our approaches in a fairly small dataset, with each edge hosting 200 tuples with a fanout of 4. Despite the superiority of the `SMART` method, we are also interested in how it scales in large dataset and with large fanout value. Figure 6.9 plots the cost for the `RJ` and `SMART` methods over uniform uncorrelated dataset for a fanout value from 2 to 8. As the fanout value grows, the cost for the `SMART` grows steadily in a slow pace. Compared with a fanout of 2, the cost for `SMART` method grows to 5.2 times for a fanout of 8. On the contrary, the cost for the `RJ` grows more than 34 times for the same fanout change. These results demonstrate that our approach is extremely suitable in datasets where heavy joins are expected (*i.e.*, large fanout).

Figure 6.8 plots the cost for the `RJ` and the `SMART` methods in large datasets, with the number of tuples on each edge ranging from 20 to 20000 and a fixed fanout of 4. Under four different types of datasets, the `SMART` method unanimously demonstrates further benefits compared the other two methods. By increasing the size of the dataset by 1000 times, the cost of the `SMART` method only grows 117.53, 137.33, 2.22, 3.05 times for each of the four datasets respectively. This is because the `SMART` method can prune out a large set of unnecessary binding processing by maintaining a tight upper bound. On the contrary, the cost of the `RJ` method grows by a factor of 1084 times among all the four datasets simply because it has to expand all partial bindings on each join path.

| $Bindings$ | $(e_1, e_2, e_3, e_4, e_5, e_6)$ | score |
|---|---|---|
| (*"Tao Li"*, *"Washington"*, *"Mitsunori Ogihara"*, *"CIKM 2004"*) | (0.833, 0.34, 0.75, 0.8, 0.833, 0.34) | 0.9627 |
| (*"Tao Li"*, *"Toronto"*, *"Mitsunori Ogihara"*, *"SIGIR 2003"*) | (0.667, 0.7, 0.8, 0.8, 0.667, 0.7) | 0.9471 |
| (*"Daphne Koller"*, *"Seattle"*, *"Joseph Y. Halpern"*, *"IJCAI 2001"*) | (0.9, 0.142, 0.9, 0.003, 0.9, 0.041) | 0.9137 |

Table 6.6: Top-3 Bindings of real-world experiments

### 6.6.6 Real-World Experiments

We show in Section 6.6.2 how we extract real world datasets. Table 6.6 shows the top-3 bindings as well as the edge scores for the real dataset experiment. As shown, our algorithm returns reasonable results for such a real life query. In particular, all edges are instantiated for each of the 3 bindings, indicating that every path contributes to the final score of the bindings. Although the third binding has the highest score on one of the paths (the single edge path $e_1$), the other two bindings have relatively high scores on all paths, therefore and result in higher overall score.

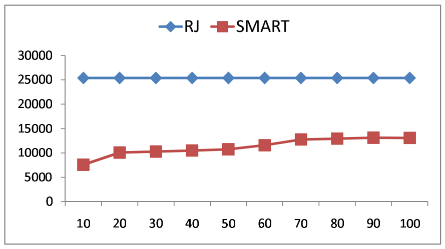


Figure 6.10: Cost of top-$k$ join queries for the real-world dataset

Figure 6.10 shows the cost of the SMART and RJ approaches for the real-world experiments. Similar as the synthetic experiments, the SMART method achieves significant cost savings compared with the RJ method. On average, the SMART method beats the RJ as much as 70%. This demonstrate that our algorithm is practical when used in real life applications.

## 6.7 Conclusions

We proposed a novel branch-and-bound approach for top-$k$ join query processing, under a cost model in which data access is expensive. Each data instance has an associated score. We model the score of the overall answer as a network reliability problem. Our algorithm dynamically retrieves a subset of the data on each join edge, and maintains tight upper and lower bounds for sets of answers. We conduct experiments with different types of datasets and query graphs, and show that our algorithm significantly outperforms the rank-join algorithm. The benefits further improve if data scores are correlated and/or skewed, which is often the case for real-life datasets. The work presented in this chapter was published in [70].

# Chapter 7

# Conclusions and future work

## 7.1 Conclusions

In this thesis, we studied the problem of corroborating answers from multiple sources. Given a query to which multiple sources provide different and possible conflicting answers, corroboration aims to identify the correct answer by distinguishing the trustworthiness of the sources. The key idea of a corroboration approach is that the more trustworthy a source is, the more likely it is to provide the correct answer. However, in order to design an effective and efficient corroboration algorithm, we have to address several challenges. The first and foremost challenge is how to derive the trustworthiness of a source and given the trustworthiness of a source, how to evaluate the probability its answer is the correct answer. Second of all, in a case where all the sources agree on a single answer, how to validate the correctness of the answer. Finally, in a case where each source only provides partial answers and it is required to combine partial answers from multiple sources to construct a final answer, how to evaluate the quality of the answer and how to efficiently derive the correct answer.

To address each of these challenges, we studied three empirical problems for which we propose novel corroboration algorithms. We first studied the problem of question answering in which multiple sources provide conflicting answers (Chapter 4). By extracting answers from the documents obtained from a search engine, our corroboration technique ranks the answers based on the number, relevance and similarity of the web sources reporting them, as well as the prominence of the answers within the sources.

The second problem we investigated was to verify the correctness of claims that are agreed upon between all sources (Chapter 5). Intuitively, a claim supported by all sources must be true, simply because there is no other source rebutting it. However, in a real world scenario this might not necessarily be true since agreeing sources could be out-dated or wrong due to copy/paste. As a solution, we proposed a novel corroboration approach that evaluates the claims on a gradual basis. Different from existing methods that apply corroboration on all the claims at once, we evaluate the claims one set at a time and gradually refine the trustworthiness of the sources.

The third problem we studied was queries in which a single source is insufficient to provide a candidate answer (Chapter 6). To answer such queries, users have to fetch and combine information from multiple sources and construct a final answer. The process of combining information from two sources is similar to the join operation in a relational database and hence the problem can be viewed as a join query processing over multiple databases. The main bottleneck of join query processing is tuple accessing which typically exhibits high and variable latency. For this problem, we propose a branch-and-bound algorithm that tightly bounds the scores of partial results and determines a good order in which it accesses the tables so as to minimize efforts in the computation of the top-k answers.

For each of the three problem, we demonstrate that there exists only trivial approach or the challenges render the performance of the state-of-the-art algorithms less than satisfactory. In comparison, our proposed approach for each problem significantly outperforms existing approaches in answer accuracy and computational efficiency.

## 7.2 Future work

Based on the findings from the studies presented in this thesis, we identify several future work directions that could be of interest and improve the strength of corroboration techniques.

### Corroboration over text-based answers

To the best of our knowledge, existing corroboration techniques including ours assumed the presence of structured answers or the ease of extraction of succinct answers from information sources. This assumption might not hold for certain scenario in which the fact is text-based. For example, news agencies usually report on breaking stories and provide relevant background tid-bits. It could be of interest to certain group of people to verify the correctness of those ‘gossips’. As another example, users reviews posted online greatly affect people’s purchase decision of a product. Due to the pervasiveness of falsified user reviews it is necessary to develop a corroboration technique that can produce genuine and accurate product reviews. In both use cases, the unique challenge is that the answers of interest are not entities or attributes of an entity but are rather descriptions of past events or experiences. It is important to define a proper data model to accommodate such answers. In addition, different from existing techniques that consider that two answers either agree or are in conflict, text-based answers could agree with each other while being phrased differently.

### Use of a finer-grained trust scores for the sources

Most existing corroboration techniques use a single trust score for each source. In Chapter 5 we proposed a multi-value trust score for the sources. In such a setting, each trust value is used to evaluate a subset of the listings. The intuition is that each trust value is a finer-grained measure of a source toward a certain of group of listings. While we derive the multi-value trust scores using a heuristic approach, we could explore if such multi-valued trust scores actually correspond to one of the properties of the entities. For instance, it is not uncommon that a source might be more accurate toward New York based restaurants or another source is more up to date of restaurants serving Asian cuisine.

### Exploring the dependency among facts

Several existing work [17, 18, 7, 16] have investigated the dependencies among the sources in the sense that some sources might be copying information from other sources. However, most existing techniques assumed that facts are independent or only considered basic functional dependencies such as one that only one fact could be true when considering the attribute value of an entity. An exception is [26] in which the authors explored the relationship among multiple attributes of entities (denoted as denial constraints) when computing the data currency. In practice, facts are rarely completely independent and taking into consideration of fact dependencies could help us leverage domain specific knowledge. As a simple example, consider a source that provides personnel contact information including *home addresses* and *phone numbers*. While it is not always true that the *city* information of the *address* record matches the *area code* part of the *phone* record, a mismatching pair does not have the same probability that both values are true compared with a matching one. Instead of using the trust scores of the source to evaluate the quality of the two attributes, it is more intuitive to plugging in the relationship between attributes so as to improve the corroboration results.

The relationship between attributes do not necessarily have to come from the same entity. Consider a data source that provides personnel contact information with only *home addresses* as well as spousal information. By leveraging the *address* information of the spouse personnel, it is possible to design a corroboration algorithm that outperforms one that only considers independent facts. However, the dependencies among the attributes, although intuitive, are rarely readily defined and easy to incorporated into a corroboration system. To use those information, a suite of sub-problems (including schema matching, entity linking, etc) need to tackled before a successful corroboration algorithm could be designed.

# References

[1] Steven P. Abney, Michael Collins, and Amit Singhal. Answer extraction. In *Proc. of the 6th Applied Natural Language Processing Conference (ANLP'00)*, 2000.

[2] David Azari, Eric Horvitz, Susan Dumais, and Eric Brill. Web-based question answering: A decision-making perspective. In *Proc. of the 19th Conference on Uncertainty in Artificial Intelligence (UAI'03)*, 2003.

[3] Anton Bakalov, Ariel Fuxman, Partha Pratim Talukdar, and Soumen Chakrabarti. SCAD: collective discovery of attribute values. In *WWW*.

[4] Raju Balakrishnan, Subbarao Kambhampati, and Manishkumar Jha. Assessing relevance and trust of the deep web sources and results based on inter-source agreement. *TWEB*, 7(2):11, 2013.

[5] Somnath Banerjee, Soumen Chakrabarti, and Ganesh Ramakrishnan. Learning to rank for quantity consensus queries. In *SIGIR*.

[6] Yaniv Bernstein and Justin Zobel. Redundent documents and search effectiveness. In *Proc. of 14th ACM International Conference on Information and Knowledge Management (CIKM'05)*, 2005.

[7] Laure Berti-Equille, Anish Das Sarma, Xin Dong, Amélie Marian, and Divesh Srivastava. Sailing the information ocean with awareness of currents: Discovery and application of source dependence. In *CIDR*, 2009.

[8] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proc. of the 18th IEEE International Conference on Computer Communications (INFOCOM'99)*, 1999.

[9] Sergey Brin, James Davis, and Hector Garcia-Molina. Copy detection mechanisms for digital documents. In *Proc. of the 1995 ACM International Conference on Management of Data (SIGMOD'95)*, 1995.

[10] Jiangping Chen, Anne Diekema, Mary D. Taffet, Nancy J. McCracken, Necati Ercan Ozgencil, Ozgur Yilmazel, and Elizabeth D. Liddy. Question answering: Cnlp at the trec-10 question answering track. In *Proc. of the 10th Text REtrieval Conference (TREC 2001)*, 2001.

[11] Tao Cheng and Kevin Chen-Chuan Chang. Beyond pages: supporting efficient, scalable entity search with dual-inversion index. In *EDBT*.

[12] Tao Cheng, Xifeng Yan, and Kevin Chen-Chuan Chang. Entityrank: Searching entities directly and holistically. In *Proc. of the 33rd International Conference on Very Large Data Bases (VLDB'07)*, 2007.

[13] Yueh-Hsuan Chiang, AnHai Doan, and Jeffrey F. Naughton. Modeling entity evolution for temporal record matching. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 1175–1186, 2014.

[14] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 1991.

[15] Nilesh Dalvi and Dan Suciu. Management of probabilistic data: foundations and challenges. In *Proc. of the 26th ACM symposium on Principles of database systems (PODS'07)*, 2007.

[16] Xin Dong, Laure Berti-Equille, Yifan Hu, and Divesh Srivastava. Global detection of complex copying relationships between sources. *PVLDB*, 3(1):1358–1369, 2010.

[17] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. Integrating conflicting data: The role of source dependence. *PVLDB*, 2(1):550–561, 2009.

[18] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. Truth discovery and copying detection in a dynamic world. *PVLDB*, 2(1):562–573, 2009.

[19] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. From data fusion to knowledge fusion. *PVLDB*, 7(10):881–892, 2014.

[20] Xin Luna Dong, Barna Saha, and Divesh Srivastava. Less is more: Selecting sources wisely for integration. *PVLDB*, 6(2):37–48, 2012.

[21] Xin Luna Dong and Divesh Srivastava. Big data integration. *PVLDB*, 6(11):1188–1189, 2013.

[22] Doug Downey, Oren Etzioni, and Stephen Soderland. A probabilistic model of redundancy in information extraction. In *IJCAI*, 2005.

[23] Susan Dumais, Michele Banko, Eric Brill, Jimmy Lin, and Andrew Ng. Web question answering: Is more always better? In *SIGIR*, 2002.

[24] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[25] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences (JCSS)*, 66(1), 2003.

[26] Wenfei Fan, Floris Geerts, and Jef Wijsen. Determining the currency of data. *ACM Trans. Database Syst.*, 37(4):25, 2012.

[27] Yi Fang, Luo Si, Zhengtao Yu, Yantuan Xian, and Yangbo Xu. Entity retrieval with hierarchical relevance model. In *Proc. of the Eighteenth Text REtrieval Conference (TREC 2009)*, 2009.

[28] Alban Galland, Serge Abiteboul, Amélie Marian, and Pierre Senellart. Corroborating information from disagreeing views. In *WSDM*, pages 131–140, 2010.

[29] Liang Ge, Jing Gao, Xiao Yu, Wei Fan, and Aidong Zhang. Estimating local information trustworthiness via multi-source joint matrix factorization. In *ICDM*, pages 876–881, 2012.

[30] Luis Gravano, Panagiotis G. Ipeirotis, Nick Koudas, and Divesh Srivastava. Text joins for data cleansing and integration in an rdbms. In *ICDE*, pages 729–731, 2003.

[31] Sanda M. Harabagiu, Dan I. Moldovan, Marius Pasca, Rada Mihalcea, Mihai Surdeanu, Razvan C. Bunescu, Roxana Girju, Vasile Rus, and Paul Morarescu. Falcon: Boosting knowledge for answer engines. In *Proc. of the 9th Text REtrieval Conference (TREC-9)*, 2000.

[32] Eduard H. Hovy, Laurie Gerber, Ulf Hermjakob, Michael Junk, and Chin-Yew Lin. Question answering in webclopedia. In *Proc. of the 9th Text REtrieval Conference (TREC-9)*, 2000.

[33] Eduard H. Hovy, Ulf Hermjakob, and Chin-Yew Lin. The use of external knowledge of factoid qa. In *TREC*, 2001.

[34] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. In *VLDB*, pages 754–765, 2003.

[35] Valentin Jijkoun and Maarten de Rijke. Retrieving answers from frequently asked questions pages on the web. In *Proc. of the 14th ACM International Conference on Information and Knowledge Management (CIKM'05)*, pages 76–83, 2005.

[36] Richard M. Karp and Michael Luby. Monte-carlo algorithms for enumeration and reliability problems. In *FOCS*, 1984.

[37] Gjergji Kasneci, Jurgen Van Gael, David H. Stern, and Thore Graepel. Cobayes: bayesian knowledge corroboration with assessors of unknown areas of expertise. In *WSDM*, pages 465–474, 2011.

[38] Gjergji Kasneci, Fabian Suchanek, Maya Ramanath, and Gerhard Weikum. How naga uncoils: Searching with entities and relations. In *Proc. of the 16th International Conference on the World Wide Web (WWW'07)*, 2007.

[39] Benny Kimelfeld and Yehoshua Sagiv. Maximally joining probabilistic data. In *PODS*, pages 303–312, 2007.

[40] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[41] Yannis Kotidis, Amélie Marian, and Divesh Srivastava. Circumventing data quality problems using multiple join paths. In *CleanDB*, 2006.

[42] Nick Koudas, Amit Marathe, and Divesh Srivastava. Flexible string matching against large databases in practice. In *Proc. of the 30th International Conference on Very Large Data Bases (VLDB'04)*, pages 1078–1086, 2004.

[43] Cody C. T. Kwok, Oren Etzioni, and Daniel S. Weld. Scaling question answering to the web. In *WWW*, 2001.

[44] Jian Li, Barna Saha, and Amol Deshpande. A unified approach to ranking in probabilistic databases. In *VLDB*, pages 502–513, 2009.

[45] Pei Li, Xin Luna Dong, Andrea Maurino, and Divesh Srivastava. Linking temporal records. *PVLDB*, 4(11):956–967, 2011.

[46] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. Truth finding on the deep web: Is the problem solved? *PVLDB*, 6(1):550–561, 2013.

[47] Amélie Marian, Nicolas Bruno, and Luis Gravano. Evaluating top-k queries over web-accessible databases. *ACM Transactions on Database Systems (TODS)*, 29(8), 2004.

[48] Amélie Marian, Luis Gravano, and Nicolas Bruno. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2), 2004.

[49] Amélie Marian and Minji Wu. Corroborating information from web sources. *IEEE Data Eng. Bull.*, 34(3):11–17, 2011.

[50] Andew McCallum. Information extraction: distilling structured data from unstructured text. *ACM Queue*, 3(9):48–57, 2005.

[51] Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. Supporting incremental join queries on ranked inputs. In *VLDB*, pages 281–290, 2001.

[52] Aditya Pal, Vibhor Rastogi, Ashwin Machanavajjhala, and Philip Bohannon. Information integration over time in unreliable and uncertain environments. In *WWW*, 2012.

[53] Marius Pasca and Sanda M. Harabagiu. High performance question/answering. In *Proc. of the 24th Annual International ACM SIGIR Conference (SIGIR'01)*, 2001.

[54] Jeff Pasternack and Dan Roth. Knowing what to believe (when you already know something). In *COLING*, pages 877–885, 2010.

[55] Jeff Pasternack and Dan Roth. Making better informed trust decisions with generalized fact-finding. In *IJCAI*, pages 2324–2329, 2011.

[56] John M. Prager, Eric W. Brown, Anni Coden, and Dragomir R. Radev. Question-answering by predictive annotation. In *Proc. of the 23rd Annual International ACM SIGIR Conference (SIGIR'00)*, 2000.

[57] Dragomir R. Radev, Weiguo Fan, Hong Qi, Harris Wu, and Amardeep Grewal. Probabilistic question answering on the web. In *Proc. of the 11th International Conference on World Wide Web (WWW'02)*, pages 408–419, 2002.

[58] Christopher Ré, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.

[59] Theodoros Rekatsinas, Xin Luna Dong, and Divesh Srivastava. Characterizing and selecting fresh data sources. In *SIGMOD*, 2014.

[60] Mohamed A. Soliman, Ihab F. Ilyas, and Kevin Chen-Chuan Chang. Top-k query processing in uncertain databases. In *ICDE*, pages 896–905, 2007.

[61] Matthew Solomon, Cong Yu, and Luis Gravano. Popularity-guided top-k extraction of entity attributes. In *WebDB*, 2010.

[62] Swapna Somasundaran, Theresa Wilson, Janyce Wiebe, and Veselin Stoyanov. Qa with attitude: Exploiting opinion type analysis for improving question answering in on-line discussions and the news. In *Proc. of the International Conference on Weblogs and Social Media*, 2007.

[63] Sandeep Tata and Jignesh M. Patel. Estimating the selectivity of *tf-idf* based cosine similarity predicates. *SIGMOD Record*, 36(2):7–12, 2007.

[64] Martin Theobald, Holger Bast, Debapriyo Majumdar, Ralf Schenkel, and Gerhard Weikum. Topx: efficient and versatile top- query processing for semistructured data. *VLDB J.*, 17(1):81–115, 2008.

[65] Martin Theobald, Jonathan Siddharth, and Andreas Paepcke. Spotsigs: robust and efficient near duplicate detection in large web collections. In *Proc. of the 31st Annual International ACM SIGIR Conference (SIGIR'08)*, pages 563–570, 2008.

[66] Martin Theobald, Gerhard Weikum, and Ralf Schenkel. Top-k query evaluation with probabilistic guarantees. In *Proc. of the 30th International Conference on Very Large Data Bases (VLDB'04)*, 2004.

[67] TREC Question Answering Track.

[68] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8:410–421, 1979.

[69] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[70] Minji Wu, Laure Berti-Equille, Amélie Marian, Cecilia M. Procopiuc, and Divesh Srivastava. Processing top-k join queries. *PVLDB*, 3(1):860–870, 2010.

[71] Minji Wu and Amélie Marian. Corroborating answers from multiple web sources. In *Proc. of 10th International Workshop on Web and Database (WebDB'07)*, 2007.

[72] Minji Wu and Amélie Marian. A framework for corroborating answers from multiple web sources. *Inf. Syst.*, 36(2):431–449, 2011.

[73] Minji Wu and Amélie Marian. Corroborating facts from affirmative statements. In *EDBT*, pages 157–168, 2014.

[74] Jinxi Xu, Ana Licuanan, Jonathan May, Scott Miller, and Ralph Weischedel. Trec2002 qa at bbn: Answer selection and confidence estimation. In *Proc. the 11th Text REtrieval Conference (TREC'02)*, 2002.

[75] Hui Yang and Tat-Seng Chua. Qualifier: Question answering by lexical fabric and external resources. In *Proc. of the Conference of the European Chapter of the Association for Computational Linguistics*, pages 363–370, 2003.

[76] Xiaoyan Yang, Cecilia M. Procopiuc, and Divesh Srivastava. Recommending join queries via query log analysis. In *ICDE*, pages 964–975, 2009.

[77] Ke Yi, Feifei Li, George Kollios, and Divesh Srivastava. Efficient processing of top-k queries in uncertain databases. In *ICDE*, pages 1406–1408, 2008.

[78] Xiaoxin Yin, Jiawei Han, and Philip S. Yu. Truth discovery with multiple conflicting information providers on the web. In *KDD*, 2007.

[79] Xiaoxin Yin, Jiawei Han, and Philip S. Yu. Truth discovery with multiple conflicting information providers on the web. *IEEE Trans. Knowl. Data Eng.*, 20(6):796–808, 2008.

[80] Dell Zhang. Web based question answering with aggregation strategy. In *Proc. of the 6th Asia-Pacific Web Conference*, pages 353–362, 2004.

[81] Bo Zhao, Benjamin I. P. Rubinstein, Jim Gemmell, and Jiawei Han. A bayesian approach to discovering truth from conflicting sources for data integration. *PVLDB*, 5(6):550–561, 2012.

[82] Zhou Zhao, James Cheng, and Wilfred Ng. Truth discovery in data streams: A single-pass probabilistic approach. In *CIKM*, pages 1589–1598, 2014.

[83] G. Zipf. *Selective Studies and the Principle of Relative Frequency in Language*. Harvard University Press, 1932.

[84] George K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley (Reading MA), 1949.