UNIVERSITY OF CALIFORNIA

Los Angeles

Summarizing Massive Information for Querying Web Sources and Data Streams

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Seyed Hamid Mousavi Behbahani

2014

Abstract of the Dissertation

Summarizing Massive Information for Querying Web Sources and Data Streams

by

Seyed Hamid Mousavi Behbahani

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2014

Professor Carlo Zaniolo, Chair

Largely as a result of advances brought by the Web and related technologies, we are now experiencing a tremendous growth in the volume of data streaming between, and stored at, many nodes of the Internet. This "*Big Data*" revolution is underscoring the importance of summarization in general, and in particular in two new application areas that are rich of practical significance and interesting research challenges. Indeed, while summarization techniques, including sampling, histograms, and quantiles, remain critical in analyzing large data sets and optimizing queries in traditional databases, new techniques are needed to address the following two problems. The first is that, in addition to summarization techniques for stored data, we now need online/continuous summaries for the streaming data, e.g., real-time online histograms. When dealing with massive data streams and fast-changing distributions, summaries should be quickly updated with the newly arrived data, in order to reflect the most recent portion (window) of the data stream. The second problem is that the Web is storing large corpora of structured, semi-structured, and unstructured (free-text) documents, and these documents are subject to the ambiguities of natural language and the challenges they pose to machine processing. This situation has so far limited severely the ability of smart applications to use the information contained in Web pages, as needed to realize the "*Se-*

mantic Web" vision. It is however clear that many of these limitations can be overcome and advanced searches and analysis applications can be supported, if the knowledge of each Web page can be summarized into a standard machine-friendly structure. In this dissertation, we attack these two difficult problems by proposing fast summarization techniques for (i) scalar information of data streams and (ii) textual information in Web pages. For scalar data, we present light and fast synopses, namely histograms, combined with various sampling approaches in order to implement more practical summarization techniques over massive data sets and data streams. To our knowledge, this technique provides the most accurate online histograms for data streams with sliding windows. For textual documents, we introduce several techniques and systems for extracting structured summaries from unstructured text and use these structured summaries to complete the existing ones as well as to improve their consistency.

The dissertation of Seyed Hamid Mousavi Behbahani is approved.

John Cho

Stott Parker

Yingnian Wu

Carlo Zaniolo, Committee Chair

University of California, Los Angeles

2014

To my lovely wife and family.

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

First and foremost, I would like to sincerely thank my advisor, Professor Carlo Zaniolo. He was an incredibly insightful mentor for me in my years at UCLA. I truly appreciate him for always believing in me and constantly guiding, helping, and supporting me. I would also like to thank my committee members, professor John Cho, professor Stott Parker, professor Yingnian Wu for their time and for their priceless suggestions and comments. I also had the honor to collaboration with Dr. Markus Iseli and Dr. Deirdre Kerr from the National Center for Research on Evaluation, Standards, and Student Testing (CRESST). I really appreciate their invaluable contribution and support to this research. Without any doubt, working with them was the most pleasant part of my Ph.D. study. I wish to also thank my dear colleague and friend, Shi Gao, who has been a great collaborator on a number of research projects. I am also grateful to members and alumni of the Web Information Systems Laboratory for their encouragement and understanding. I really enjoyed my time working in a fantastic and supporting group with Alex Shkapsky, Kai Zeng, Mohan Yang, Gurneet Chhokar, Dr. Nikolay Laptev, Dr. Barzan Mozafari, Dr. Carlo Curino, Chongsheng Zhang, Mirjana Mazuran, Dr. Massimo Mazzeo, Hsuan (Cheryl) Chiu, Jiaqi Gu, and many others.

Last but not least, I would like to express my special appreciations to my family who are the real source of inspiration and motivation for me; my beloved wife who has made my tough days easier, joyful, and adventurous, my mother who has been my first, best, and most patient teacher, my father who has been my constant supporter, and my sisters who always loved me unconditionally.

VITA

2004	B.S. (Computer Engineering), University of Tehran, Tehran, Iran.
2003–2005	Research Assistant, Router Lab., University of Tehran, Tehran, Iran.
2007	M.S. (Computer Engineering), Sharif University of Technology, Tehran, Iran.
2007–2008	Lecturer at Sharif University of Technology, Tehran, Iran. Teaching “Introduction to Programming (C++)”, and “Operating Systems Laboratory” courses.
2007–2008	Lecturer at Shariaty Technical College, Tehran, Iran. Teaching “Operating Systems”, “Introduction to Algorithms”, and “Formal Languages and Automata Theory” courses.
2010	Teaching Assistant, Database Systems, Computer Science Department, UCLA.
2011	Research Intern, Microsoft Research - Redmond, WA.
2010–2013	Research Assistant, National Center for Research on Evaluation, Standards, and Student Testing (CRESST), UCLA.
2010–2014	Research Assistant, Computer Science Department, UCLA.

PUBLICATIONS

H. Mousavi, S. Gao, and C. Zaniolo. "*Discovering Attribute and Entity Synonyms for Knowledge Integration and Semantic Web Search*" In SSW Workshop, Rive del Garda, Trento, Italy. 2013.

H. Mousavi, S. Gao, and C. Zaniolo. "*IBminer: A Text Mining Tool for Constructing and Populating InfoBox Databases and Knowledge Bases*" In VLDB, Rive del Garda, Trento, Italy. 2013.

H. Mousavi and C. Zaniolo. "*Fast Computation of Approximate Biased Histograms on Sliding Windows over Data Streams*" In SSDBM, Baltimore, Maryland. 2013. (Best Paper Award winner)

C. Zhang, Y. Hao, M. Mazuran, H. Mousavi, C. Zaniolo, F. Masseglia. "*Mining frequent itemsets over tuple-evolving data streams.*" SAC'13: Symposium on Applied Computing. 2013.

H. Mousavi and C. Zaniolo."*Fast and Space-Efficient Computation of Equi-Depth Histograms for Data Streams.*" In EDBT 2011 Joint Conference, Uppsala, Sweden, 2011.

H. Mousavi, D. Kerr and M. Iseli. "*A New Framework for Textual Information Mining over Parse Trees*", In ICSC2011, Palo Alto, USA, 2011.

H. Thakkar, N. Laptev, H. Mousavi, B. Mozafari, V. Russo, C. Zaniolo. "*SMM: A data stream management system for knowledge discovery.*" In ICDE 2011, pp 757-768, Hannover, 2011.

H. Mousavi, D. Kerr, M. Iseli, and C. Zaniolo. "*Mining semantic structures from syntactic structures in free text documents.*" (CSD Technical Report #140005), University of California, Los Angeles, 2014.

D. Kerr, H. Mousavi, and M. Iseli. "*Automatic Short Essay Scoring Using Natural Language Processing to Extract Semantic Information in the Form of Propositions,*" (UCLA/CRESST Technical Report #831), University of California, Los Angeles, 2013.

H. Mousavi, S. Gao, and C. Zaniolo. "*Discovering Attribute and Entity Synonyms for Knowledge Integration and Semantic Web Search*", (CSD Technical Report #130013), University of California, Los Angeles, 2013.

H. Mousavi, D. Kerr, M. Iseli, and C. Zaniolo. "*OntoHarvester: An unsupervised ontology generator from free text*", (CSD Technical Report #130003), University of California, Los Angeles, 2013.

H. Mousavi and C. Zaniolo. "*Fast Computation of Approximate Biased Histograms on Sliding Windows over Data Streams*", (CSD Technical Report #130002), University of California, Los Angeles, 2013.

H. Mousavi, D. Kerr, M. Iseli, and C. Zaniolo. "*Deducing InfoBoxes from Unstructured Text in Wikipedia Pages*", (CSD Technical Report #130001), University of California, Los Angeles, 2013.

N. Laptev, H. Mousavi, A. Shkapsky and C. Zaniolo. "*Optimizing Regular Expression Clustering for Massive Pattern Search.*" CSD, UCLA, 2012. (Technical Report #120005).

H. Mousavi, D. Kerr, and M. Iseli. "*A new framework for textual information mining over parse trees.*" In (CRESST Report #805). University of California, Los Angeles, 2011.

CHAPTER 1

Introduction

The introduction of *Big Data* in recent years is shifting traditional database management systems (DBMSs) toward more scalable approaches. Nowadays, the processes of querying, analyzing, and mining in many *Big Data* environments are very challenging tasks using traditional DBMSs (if not impossible) due to the size and complexity of the data. Sensor networks, log-records, Web-clicks, telecommunication network monitoring systems, traffic monitoring systems, social networks, stock market tickers, online encyclopedias, online reviewing systems, and customer complaint systems are only a few examples of such environments. In such environments, several issues exacerbate the challenges of efficiently querying and analyzing the data. Here, we listed four of the most important such issues:

- **Unmanageable Size:** As already mentioned, the volume of the data in Big Data applications is one of the biggest challenge in querying and analyzing these data sets.
- **Unstructured and Semi-Structured Data:** Unstructured (Text) and semi-structured data are the most prevalent type of data across the Web. Online encyclopedias, reviewing systems, customer reviews, Blogs, social networks, and many other applications contains vast amount of data in text format or semi-structured formats such as tables, lists, etc. Due to the ambiguity of the natural languages, and lack of a standard terminology (or Ontology), dealing with such data sources adds another level of complexity to the challenge of mining Big Data.
- **Online/Continuous Queries:** Query processing in most Big Data applications

must be performed on-the-fly [BW01]. That is, the users need to get their query's answers at most in few seconds after submitting the query. On the same note, once a query is submitted, the system often should run the query continuously and generate new results considering the new-coming data items in an online fashion. These types of data sets are referred to as *data streams* and have been the focus of many research efforts in the last decade [BBD02], [GGR02], [Mut03], [GZK05], and [Agr07].

- **Data Streams with Sliding Windows:** In applications dealing with data streams, the more recent data are usually of more interest, since the data may be streaming for a long time and the older data items may be invalid or changed. To deal with such cases, the concepts of *ageing/decaying* [CL03] and *sliding windows* [BBD02, LMT05] are introduced. With ageing, as the name offers, we give less weight to older data items as new data items arrives, while with sliding windows we only consider a certain number of the most recent data items.[1]

The most common solution to cope with these issues, in which the quantity and the complexity of the information exceed the capacity of existing computing systems, is to summarize the data sets. Sampling, histograms, quantiles, etc. are some of these summarization techniques, that have been used extensively in traditional databases. However, these techniques now require dramatic changes in order to effectively address the challenges in "*Big Data*". In this dissertation, we attack these challenging problems by proposing fast summarization techniques for (i) scalar information in both data sets and data streams and (ii) textual and structured information in Web pages. The main goal in summarizing massive data sets is not only to reduce the volume of the data, but also into convert it to more machine-friendly structures that can be analyzed with less effort. In this dissertation, we refer to the process of creating and maintaining any kind of summary for the purpose of fast query processing and data analyses as a *summarization technique*.

[1]Or those data items which have arrived no later than a certain amount of time before the current time.

As for the scalar data, one of the earliest works on summarizing data sets was introduced by Munro and Paterson [MP80] in which the problems of sorting and selection in databases with a limited memory space is addressed. Throughout the last few decades many researchers have proposed new summarization techniques for different types of data sets [Vit85a, GM98, GKM01, GKM02, QAA02, GKM02, CM05, ZLX06, GM07, THP08]. Perhaps the simplest summarization technique, though very efficient for many queries, is random uniform sampling. Almost any type of function or technique can be combined with a sampling technique, perhaps with some slight changes. However, for some scenarios sampling can not provide an acceptable balance between the memory/time requirement and the accuracy of the results (e.g. queries involving join operations [CMN99, CM99]). To provide more accurate results using less space and time, more complex structures such as *Quantiles* [GK01a], *Histograms* [IP99], *Wavelets* [VW99, CGR00], etc. should be employed for massive data sets and streaming data environments. These structures, which are often used to approximate the results of users' queries are called *synopsis* or *sketch* data structures [GM99].

Informally, synopses are light data structures that store useful information of the entire data set and can be utilized to approximately answer users' queries in a more time-efficient way. These queries can be *point* or *range queries*, answers to *inner-product queries*, *frequency moments*, *Heavy Hitters*, *Top-k Items*, *frequent patterns*, *entropy estimation*, or other aggregate functions [GM07]. Later, these queries combined with other techniques can be utilized in practical applications such as, approximate query answering [BW01, Gib01, WAA01], security systems over computer networks, sensor monitoring systems, splitting parallel databases, frequent itemset mining systems, mining and monitoring Web logs and click-streams, and several other data mining applications in massive data sets and data streams.

Recently several summarization techniques were proposed for unstructured or semi-structured data as well. Perhaps the most prominent such techniques is provided by the *InfoBoxes* in Wikipedia pages [Wik12], which summarize the most important attributes and their values for entities described in the pages. In addition to being very valuable for human readers, this information has many applications in various information systems and in particular question answering systems such as Faceted Search [AT10], SWIPE [AZ12], and IQ [MRS11]. Moreover, this information has been used extensively to generate *Commonsense Knowledge Bases* (CKBs), e.g, DBpedia [BLK09] and YaGo2 [HSB11], and domain specific knowledge bases (*Ontologies*) [BCS07, Dro03, PD10] in systems which support text summarization, document categorization, semantic search, and automatic essay grading.

Unfortunately, since the current process of generating InfoBoxes is manual, and standard ontologies are often not used, InfoBoxes suffer from serious inconsistency and incompleteness issues [WW08]. In recent years, several automatic techniques have been proposed to address these issues. Many of such works only consider structured or semi-structured part of data [WW08, BEP08, SKW08, BLK09, HSB11]. Some approaches use limited forms of NLP-based techniques to extract InfoBox-like information from text techniques [ECD04, YCB07, CBK10a, PGR10, LBN10, WLH11]. However, none of the mentioned efforts are able to take full advantage of the linguistic morphologies of sentences in the text to generate high-quality structured information. As a result, the existing approaches are not able to exploit hidden information in the text, and are often incapable of extracting those pieces of information that are not mentioned frequently in the documents.

With the introduction of the recent efforts on summarization techniques for massive data sets and data streams on the Web as well as their shortcomings, we encounter the following two major issues still needed to be addressed:

A. First, the proposed summarization techniques on scalar data sets and data streams still are not fully practical and scalable for many Big Data applications, thus more practicable techniques are needed in order to provide highly accurate approximate results, especially in fast data streaming environments with sliding windows. Moreover, the most practical approximation approach for many applications are still the sampling techniques. Although performing well, sampling can be used in combination with other synopses to improve the performance. However, the effect of using sampling techniques in combination with other synopses still needs more attention.

B. The second issue is the lack of efficient and effective summarization techniques on structured, semi-structured, and unstructured data sources. Many of the mentioned approaches rely on manual techniques and none of them follow a common terminology, which makes their use very limited. Perhaps Wikidata [Wik] is the only project trying to provide a [multilingual] standard for presenting the summaries. However, it highly relies on human contributors which makes it very hard to scale. Moreover, a large portion of the Web (some estimate it to be more than 80%) consists of textual and semi-structured information that includes a huge wealth of knowledge. However, current approaches for querying and mining this part of the Web mostly rely on Bag-of-Words techniques. Unfortunately, since these techniques do not capture morphological structure in the text, they are incapable of extracting much of the hidden knowledge in the text. Thus, more sophisticated techniques are required to convert/summarize text into a machine-friendly structure; a structure that can be easily queried and mined with in-hand techniques.

In this work, as discussed in the following section, we mainly aim at addressing these two challenging issues by proposing new summarization techniques.

1.1 Overview and Contributions

To address the first issue mentioned above, we propose two new types of light-weight histograms; *equi-depth histograms* and *biased histograms*. Both types of histogram support massive data sets and data streams with sliding windows, and has been proven to significantly outperform their state-of-the-art peers. We also investigate and provide the most efficient ways of combining these two synopses with sampling techniques to further improve their time and space performance. These histograms will then provide quick and accurate estimations of the data distribution, which are in fact very beneficial for a large set of applications such as query optimization, approximate query answering, distribution fitting, parallel database partitioning, and data mining.

As for the second issue, we propose a new integration technique to i) integrate the existing structured knowledge bases in the Web, ii) improve and complete them using text mining approaches, and iii) provide online tools for human volunteers to revise the final structured summaries (*Knowledge Base* or *KB*). This approach is referred to as *IKBstore* throughout this dissertation. In order to generate the structured summaries in part (ii), we present a novel NLP-based technique to extract structured summary information from free text. This text mining system, which is perhaps the most important component of *IKBstore*, generates summaries in the form of InfoBoxes in Wikipedia and thus is called *InfoBox Miner* or *IBminer*. *IBminer* takes advantage of our recently implemented NLP-based text mining framework (called *SemScape*) that converts (or equivalently summarizes) text into weighted graph structures called *TextGraphs*. TextGraphs hide syntactical structures of the text by providing grammatical connections between terms and words in the text as a weighted directed graph, and thus are much more machine-friendly to be analyzed with respect to the Bag-of-Words model.

More specifically, the following contributions are proposed in this dissertation:

- We design and implement a new text mining framework, called *SemScape*, which converts the unstructured text into a weighted directed graph called *TextGraph*. As opposed to Parse Trees and similar structures, TextGraphs are much closer to the semantic of the text and hide many syntactical features of the text, include single- and multi-word terms (candidate terms) in the text and their grammatical connections, and are able to present ambiguity of the text through weighted edges. We also provide a SPARQL-like language with few new features for querying and mining knowledge from the TextGraphs. With these, SemScape provides very powerful and expressive structures and tools that can be utilized in order to summarize and mine textual documents.

- Using SemScape, we present two complex and novel text mining techniques namely *OntoHarvester* for generating (populating) ontologies and *IBminer* for generating structured information in the form of Wikipedia's InfoBoxes. We should note that the *IBminer* and *OntoHarvester* processes can be seen as automatically annotating the terms and concepts as well as their semantic relations in text. This indeed is the long-standing goal of the Semantic Web for which the users are expected to perform the time consuming task of annotating. However, by employing these two systems, the annotation task can be significantly facilitated, makes the Semantic Web one big step closer to its objectives.

- We propose a technique for integrating available structured summary information in systems such as DBPedia [BLK09], YaGo2 [HSB13], FreeBase [BEP08], MusicBrainz [MUS] , GeoNames [GEO] , etc. The integrated knowledge base, called *IKBstore*, is then improved by mining accompanying text for the subjects in Wikipedia or similar encyclopedias through the *IBminer* System. *IKBstore* also provides two strong tools for browsing and editing the knowledge base by users. With the former one users are able to browse and search the knowledge base, and perform query-by-example as recently proposed in SWIPE [AZ12]. The latter tool also provides several facilities for users to edit the knowledge base without knowing its internal structure. This not only eases the manual revising and annotation tasks for the users, but it also implicitly leads them to use the correct internal terminology, and thus automatically improves the consistency of the resulted knowledge base. *IKBstore* is available for public access at [Sem].

- As for the scalar data, we propose two new synopses for summarizing scalar massive data sets and fast data streams: *Bar Splitting Histogram* (BASH) which provides equi-depth histograms and *Bar Splitting Biased Histogram* (BSBH) which is the modified version of BASH for biased histograms. Both BASH and BSBH histograms support large sliding windows on the data streams. Our extensive experiments in both cases indicate significant improvement with respect to the state-of-the-art algorithms. We should add that, to our knowledge, BSBH is the first algorithm that supports biased histograms over sliding windows of data streams.

- To further reduce the space usage of our histograms, we study the impact of different sampling techniques on both BASH and BSBH algorithm. We also propose a biased sampling technique for BSBH, in which we sample less from the points of interest in the data distribution. This approach is proven to be more effective than the uniform random sampling.

1.2 Organization of the Dissertation

As naturally imposed by the main two shortcomings mentioned previously, we divided the dissertation into two main parts: i) summarization techniques for textual data and semi-structured data, which is explained in the next 4 chapters and ii) summarization techniques for scalar data. Since the former is our more recent work, we start the dissertation with part i) and then we discuss our techniques for part ii) in Chapters 6, 7, and 8.

For the first part, we start by introducing our pattern-based text mining framework called SemScape (Chapter 2) [MKI11a, MKI11b]. As already mentioned, SemScape converts text into more machine-friendly structures called TextGraphs. In Chapters 3 and 4, we explain two systems employing these TextGraphs to respectively generate structured information (*IBminer* [MKI13a, MGZ13a]) and Ontologies (*OntoHarvester* [MKI13b]) from free text. After introducing these subsystems, we present our final integrated knowledge based system (*IKBstore*) which summarizes already existing structured knowledge bases and those generated by *IBminer* from text into a more standard knowledge based [MGZ13a, MGZ13b, MGZ13c]. *IKBstore* is first demonstrated at VLDB 2013 [MGZ13c] and is available at [Sem] for public access.

As for the second main part of the dissertation, we continue by preliminaries and background information on different types of data set and data streams as well as existing summarization techniques for scalar data in Chapter 6. In this chapter, we cover several synopses with the main focus on Histograms and Quantiles. Next, we explain our equi-depth histogram, BASH, in Chapter 7, which is first published in EDBT 2011 [MZ11b]. In Chapter 8, we present our biased histogram algorithm, BSBH, which received best paper award at SSDBM 2013 [MZ13].

We finally conclude the dissertation in Chapter 9 and provide several lines of future work in this chapter.

CHAPTER 2

SemScape: Structuring Free Text

A tremendous amount of publicly available data in the World Wide Web is in free text format. Systems such as online encyclopedias, online reviewing systems, news agencies, social networks, online publications, costumer complaint systems, blogs, etc. are constantly generating textual data. With this increase on the volume of the textual data, users are demanding for more advanced mechanisms for retrieving and accessing the data. Nowadays, people are willing to read short summaries of long articles or news rather than the entire text. They prefer to see the average rating for different features of a service or a product instead of going over all textual reviews of other costumers. They often want to know the hottest topics in the social networks or blogs without spending too much time reading them. More importantly, advanced structured search [AZ12], faceted search [HBS10], question answering systems, and automatic personal assistants such as *Siri* and *Google Now* are getting more popular than traditional keyword-based searches.

All mentioned applications, as well as many other domain-specific ones, require a more effective approach for analyzing and mining text. This approach should be able to extract the semantic of text in a more standard structured format, hide syntactical features of the text, mine domain-specific text, handle multi-lingual text, and support ambiguities and exceptions in natural languages.

Text mining approaches can be divided into two main categories: bag-of-words (also called statistical) and deep NLP-based (or simply NLP-based) techniques. Since bag-of-words machine learning techniques do not exploit morphological structures in the text, they are mostly incapable of addressing all mentioned requirements. Supporting ambiguities or exceptions in text, resolving pronouns and coreferences, and in general mining from relatively smaller text data sets are very challenging in these approaches if not impossible. To deal with problems of this sort, these techniques usually need to use larger data sets and ignore less frequently mentioned information just to exclude some exceptions.

On the other hand, deep NLP-based approaches parse the sentences in the text and convert them into tree-based structures, called *parse trees*. Parse trees contain [some of] the morphological structures in the text with a more machine friendly format, and thus provide a better structure for text analyzing. Although deep NLP-based approaches are much more resource-demanding than keyword-based ones, they are proven to be more effective in addressing the current text mining needs. Moreover, the recent advances in distributed computing techniques also has hugely alleviated the time performance issue of deep NLP-based techniques.

Text mining through NLP-based techniques is often performed by employing some patterns on parse trees [Sta14] (or similar structures [MMM06]). Generating these patterns, either manually or by statistical patten learning techniques, is not a trivial task at all. Since a simple piece of information can be expressed in many different ways through natural languages, many patterns need to be created to extract that piece of information. Generating such patterns is both costly and time-consuming, which is mainly emanated from the fact that parse trees are still carrying various syntactical structures in the text. As a result, to mine parse trees one should deal with these different structures using many complex patterns. We should also add that, automatic pattern generation techniques are not any better, since they often need large training data sets (that have to be created manually) and are not usually able to learn patterns for exceptions.

To address this issue and ease the process of text mining, we present a new and more expressive structure, called *TextGraph*, that is much closer to the semantic of the text, and thus requires simpler and fewer patterns to be analyzed. TextGraphs capture grammatical connections between words and terms in the text in a more standard way than other structures such as parse trees [Sta14] and dependency trees [MMM06]. TextGraphs contains multi-word terms and their grammatical roles which is missed in similar structures. Moreover, by providing weights for the edges/links, TextGraph can better support text ambiguity.

TextGraphs are in fact generated by our newly proposed text mining framework, *SemScape*. SemScape uses statistical parsers [Sta14] to generate few most probable pares trees of the sentences, and then uses a small set of tree-based patterns to annotate the parse trees with some useful information called *MainParts*. MainParts carry up the hidden information in the leaves and lower branches of the parse trees to the upper non-terminal nodes. In this way, one need much simpler and more general patterns to mine the annotated trees, referred to as the *MainPart* (*MP*) Trees. Finally, SemScape uses another set of tree-based patterns over the MP Trees to extract grammatical relations between words and terms in the text to generate the TextGraphs. This two step generation of the TextGraphs from parse trees has significantly reduced the number and the complexity of required patterns.

More specifically, we present the following contributions in SemScape:

- We introduce an annotated parse tree called *MainPart tree* or *MP Tree* in which non-terminal nodes are annotated with important information resides in their branches. We also introduce a tree-based query language, called Tree-Domain (TD) Rules. TD rules can be used to query both parse trees and MP trees. Using MP trees, one can extract information from the parse trees with fewer and less complex patterns, which consequently eases the entire process of mining parse trees.
- Utilizing the MainPart trees, we propose a weighted graph representation of the text, called TextGraph, which is able to hide many syntactical features of the text. TextGraphs are more expressive than tree structures such as parse trees and

dependency trees, include multi-word (*candidate*) terms in the text, and are able to present ambiguity of the text through weighted edges.

- To be able to query and mine TextGraphs, SemScape provides a SPARQL-like query language. This query language, called Graph-Domain Rules (GD rules), provides few new features to simplify expressing queries on TextGraphs. We also present an optimization technique for matching graph patterns in TextGraphs to avoid several unwanted join operations in regular SPARQL engines.
- We propose a new Coreference Resolution technique to resolve pronouns and other references in the text. This is performed through a new component in SemScape called *Story Context* (*SC*) which uses a large body of contextual, taxonomical, and categorical information from various sources. SC also takes advantage of many syntactical patterns specifying possible or impossible resolutions. The impossible patterns significantly improve the quality of the final resolutions, which is completely overlooked in the current state-of-the-art.
- SemScape is also able to adapt with different domains by accepting an ontology. Once an ontology is fed to the framework, it generates TextGraphs with higher focus on the known terms and concept and eliminates many unrelated terms. This makes the framework capable of dealing with very noisy text data sets as well.

Since SemScape uses a pattern-based mining technique (with the full support of patterns for capturing syntactical exceptions) to generate TextGraphs, it provides a natural way for incrementally improving the system by adding more rules to capture missing grammatical connection and excluding wrongly generated connections. Currently, all patterns mentioned in this work are created manually, however one may use supervised or semi-supervised techniques to create more patterns. The SemScape framework has been used in our two text mining applications *IBminer* and *OntoHarvester* respectively discussed in Chapters 3 and 4, and is proven to be very effective.

2.1 Preparing Parse Trees

To prepare the text, we first partition it into its paragraphs and sentences and then simplify the sentences so that the parsing takes place more effectively and efficiently. To illustrate this process as well as other steps that SemScape performs to generated TextGraphs, we use the following example text throughout the section:

Motivating Example: "*Barack Obama (born August 4, 1961) is the 44th and current President of the United States. He is the first African American to hold the office. Born in Honolulu, Hawaii, Pres. Obama is a graduate of Columbia University and Harvard Law School, where he was president of the Harvard Law Review.*"

As for the first step, SemScape finds terms and values in known formats such as dates, floating point numbers, url addresses, etc. and uses a uniform way to represent them (by eliminating all occurrences of the *period* character). For instance, in our running example date '*August 4, 1961*' is converted to a standard format ('*date-1961-8-4*'). Then, SemScape tags abbreviation terms (e.g. Pres., Mr., U.S., etc.) used in the text. After these simple steps, SemScape partitions the text into paragraphs considering the *NewLine* character as the delimiter. Paragraphs are important to SemScape since they specify the scope of the pronouns used in them as explained in Section 2.4. Finally, SemScape uses *end of sentence* characters ('.', '?', '!', etc.) to extract the sentences.

Next, SemScape parses each sentence using a probabilistic parser (e.g. Charniak [Cha08] and Stanford [Sta14] parsers). For each sentence, we generate N_{pt} (>1) parse trees (PTs) using the parser. Having more than one PT will i) help us better deal with the inaccuracy and noisiness of the parsers in many cases, ii) increase the amount of extracted information, and iii) provide a better way for representing ambiguity in the text. For many cases, the first parse tree is not completely correct, so using the secondary parse trees may help improving the results. In some cases, more than one parse trees may be correct and using them helps generating more information as well as capturing possible ambiguity in text. One such parse tree for our motivating example is shown in

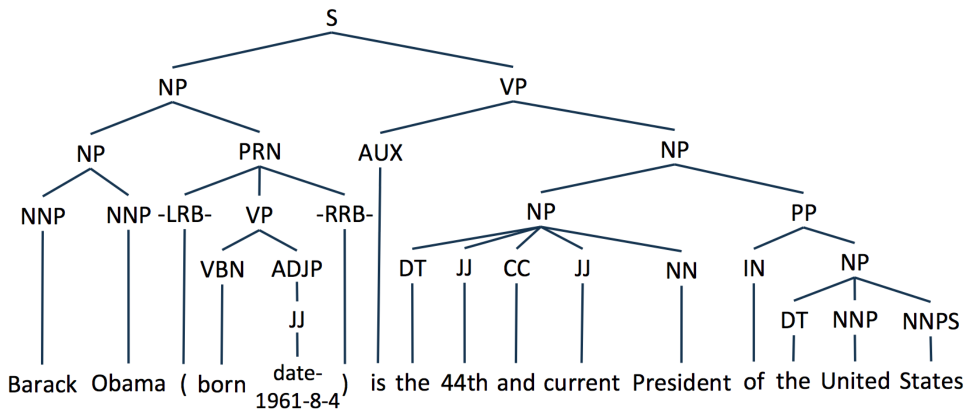


Figure 2.1: Most probable parse tree for our running example.

```
[-1]   (S
[0]       (NP
[0, 0]       (NP (NNP Barack_1) (NNP Obama_2))
[0, 1]       (PRN
[0, 1, 0]       (-LRB- (_3)
[0, 1, 1]       (VP (VBN born_4) (ADJP (JJ date-1961-8-4_5)))
[0, 1, 2]       (-RRB- )_6)))
[1]       (VP
[1, 0]       (AUX is_7)
[1, 1]       (NP
[1, 1, 0]       (NP
[1, 1, 0, 0]       (DT the_8)
[1, 1, 0, 1]       (JJ 44th_9)
[1, 1, 0, 2]       (CC and_10)
[1, 1, 0, 3]       (JJ current_11)
[1, 1, 0, 4]       (NN President_12))
[1, 1, 1]       (PP
[1, 1, 1, 0]       (IN of_13)
[1, 1, 1, 1]       (NP (DT the_14) (NNP United_15) (NNPS States_16))))))
```

Figure 2.2: Most probable parse tree for our running example in parenthesized format with our addressing schema.

Figure 2.1.

Each word in the generated parse trees is assigned an ID to make the system capable of uniquely addressing words in the text. This is mainly required to avoid confusion among repeated words and more importantly, to preserve the order of words and terms in the TextGraphs. SemScape also uses a simple addressing scheme to address nodes in the tree (Figure 2.2). Basically, each node address contains its parent address plus its position in the ordered list of siblings.

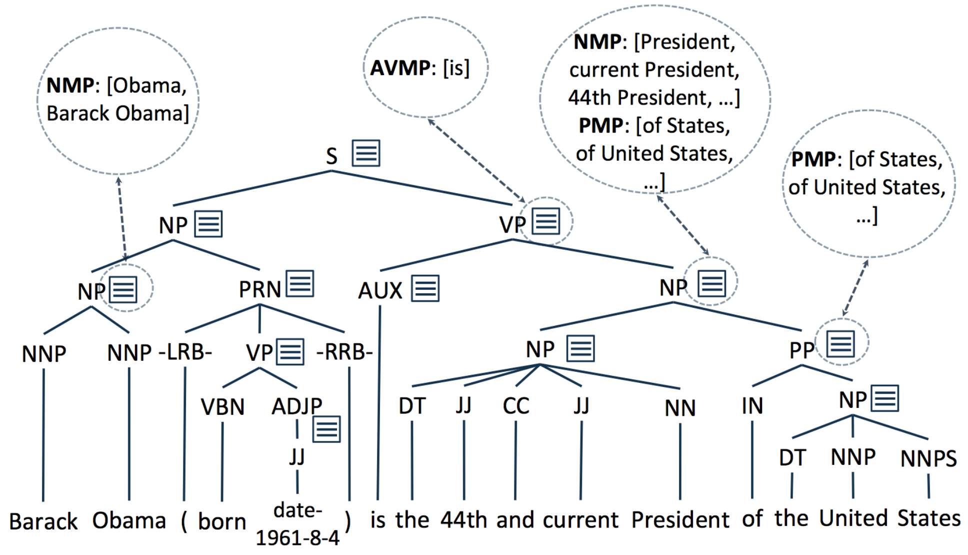

NMP: [Obama, Barack Obama]
AVMP: [is]
NMP: [President, current President, 44th President, ...] PMP: [of States, of United States, ...]
PMP: [of States, of United States, ...]
S
NP
VP
NP
PRN
AUX
NP
NNP
NNP
-LRB-
VP
-RRB-
NP
PP
VBN
ADJP
DT
JJ
CC
JJ
NN
IN
NP
JJ
DT
NNP
NNPS
Barack Obama (born date-1961-8-4) is the 44th and current President of the United States

Figure 2.3: Annotated parse tree (MainPart Tree) for the parse tree in Figure 2.1.

2.2 MainPart Trees

Parse trees are much richer structures than the text. In fact they have been frequently used in different studies to improve the bag-of-words techniques for extracting information from text. However, they still suffer from two important issues that make their use challenging, and thus limited:

- The most important issue is that the structure of the parse trees is hugely dependent to the grammar and morphological structures in text. In other words, parse trees are still far from the semantic of the sentences. Thus, extracting information from such parse trees still requires dealing with such various syntactic structures.
- The second issue is that parse trees (as well as dependency trees) are only connecting words together. Multi-word terms (A.k.a. *Candidate Terms*) and their roles in the sentences are completely missing from these structures.

To address these issues, we propose a richer structure by annotating the non-terminal nodes in the parse trees with useful information about their underlying sub-trees as shown in Figure 2.3. For instance consider the left most NP in Figure 2.3. As shown in the figure, this noun phrase is representing either '*Barack Obama*' or '*Obama*'. These pieces of information will carried up to the upper nodes in the parse trees so other application will not need to search deep in the trees branches. These types of information are referred to as *Main Parts* (*MPs*) as they specify the main part of data in each branch. MPs may contain multi-word (candidate) terms as well, which addresses the second issue mentioned earlier. The annotated parse trees are referred to as *MainPart Tree*s or *MP Tree*s.

To extract MPs in the parse trees and assign them to their corresponding nodes, we use tree-based patterns/rules, which are also called Tree Domain (TD) rules. Take the NP at address $[1, 1, 0]$ in Figure 2.2 as an example. This NP, which specifies the object of the verb '*is*' in the sentence, contains the phrase '*the 44th and current President*'. The most important component (or part) of this phrase is obviously the term '*President*', which is referred to as a Noun MainPart (NMP) of the mentioned NP. A TD rule for extracting this NMP is shown below:

Rule 1.

```
RULE mainPartRule1 ('NMP') {
   PATTERN:
             (NP *
                  (? |JJ |ADJP )
                  (? |CC )
                  (JJ |ADJP )
                  (NP |NN |NNS )
                  !* )
   RESULT: < [−1], [3] >
   RESULT: < [−1], [0] + [3] >
   RESULT: < [−1], [2] + [3] >
}
```

This rule consists of two parts: PATTERN and RESULT. PATTERN specifies a tree-like pattern for which we need to find matches in the PTs of the sentences in the text. The RESULT parts indicate how the MPs should be generated and to which node they should be assigned. We should add that PATTERNs are nested patterns and more expressive than regular expressions (or equivalently finite automata) [AM06]. This differentiates our work from most of the existing NLP-based techniques. Moreover, the tree-based format of our patterns makes them more readable and user friendly.

In Rule 1, PATTERN specifies noun phrases whose last four branches are i) an adjective or an adjective phrase (?|JJ|ADJP), ii) a conjunction (?|CC), iii) another adjective or adjective phrase (JJ|ADJP), and iv) a noun or a noun phrase (NP|NN|NNS). The first two branches are optional (indicated by a ?). From the parse tree shown in Figure 2.1 (and in parenthesized format in Figure 2.2), it is easy to see that '*44th and current President*' in our motivating example matches this pattern. If any match is found for this PATTERN, the first RESULT in Rule 1 adds the NMPs of the forth branch ('*President*' with address [3] in the pattern tree and address [1,1,0,4] in the matching tree) of the matching tree to the NMP list of its root (the node with address [-1] in the pattern tree and address [1,1,0] in the matching tree).

With its last two RESULTs, Rule 1 also suggests two multi-word terms, '*44th President*' and '*current President*'. This sort of terms are usually referred to as *Candidate Terms* in the literature, and can be directly used in *Name Entity Recognition* systems [NS07] as explained later in this section.

The MainPart Tree (MP Tree) for our running example is depicted in Figure 2.4 in parenthesized format[1]. Currently, SemScape uses 135 TD rules (accessible at [Sem]) to generated the following four types of MP information:

- **Noun MainParts (NMPs):** As already explained, NMPs are defined for noun-related nodes (NP, NN, NNS, CD, JJ, $ADJP$, ...), and they indicate the actual term(s) represented by these node.
- **Active Verb MainParts (AVMPs):** A similar concept is used for the verbs-related non-terminal nodes (S, VP, VB, VBZ, VBD, VBN, ...) in the parse trees; however, since verbs have two forms, passive and active, we have to have two types of main-parts for verb-related nodes. Thus, AVMPs capture the active verbs of the verb-related nodes.
- **Passive Verb MainParts (PVMPs):** Similar to the previous case, PVMPs cap-

[1] With this format, it is actually easier to grasp the idea of matching in our TD rules.

```
[-1] S ⇒ NMP: {Barack Obama, Obama}
[0]     NP ⇒ NMP: {Barack Obama, Obama}
[0, 0]      NP ⇒ NMP: {Barack Obama, Obama}
[0, 0, 0]       NNP ⇒ NMP: {Barack}
[0, 0, 1]       NNP ⇒ NMP: {Obama}
[0, 1]      PRN
[0, 1, 0]       -LRB-
[1, 1, 1]       VP ⇒ AVMP: {born}
[1, 1, 1, 0]        VBN ⇒ AVMP: {born}
[1, 1, 1, 1]        ADJP ⇒ NMP: {date-1961-8-4}
[1, 1, 1, 1, 0]         JJ ⇒ NMP: {date-1961-8-4}
[1, 1, 2]       -RRB-
[1]     VP ⇒ AVMP: {is}
[1, 0]      AUX ⇒ AVMP: {is},
                    PMP:{{of, the United States}, {of, States}, {of, United States}}
[1, 1]      NP ⇒ NMP: {President, current President,
                    44th President, President of United States, ...},
                    PMP: {{of, the United States}, {of, States}, {of, United States}}
[1, 1, 0]       NP ⇒ NMP: {President, current President, 44th President}
[1, 1, 0, 0]        DT ⇒ NMP: {the}
[1, 1, 0, 1]        JJ ⇒ NMP: {44th}
[1, 1, 0, 2]        CC ⇒ NMP: {and}
[1, 1, 0, 3]        JJ ⇒ NMP: {current}
[1, 1, 0, 4]        NN ⇒ NMP: {President}
[1, 1, 1]       PP ⇒ PMP:{{of, the United States}, {of, States}, {of, United States}}
[1, 1, 1, 0]        IN ⇒ NMP: {of}
[1, 1, 1, 1, 0]     NP ⇒ NMP: {States, United States}
[1, 1, 1, 1, 0, 0]      DT ⇒ NMP: {the}
[1, 1, 1, 1, 0, 1]      NNP ⇒ NMP: {United}
[1, 1, 1, 1, 0, 2]      NNPS ⇒ NMP: {States}
```

Figure 2.4: MP Tree for the parse tree in Figure 2.2. For brevity, we did not include all the MP information in the graph.

ture the passive verbs in verb-related non-terminal nodes. Passive verbs are of particular importance since they change the regular roles of the subjects and the objects in the sentences.

- **Preposition MainParts (PMPs):** The fourth MainPart set is for prepositions and preposition phrases. Both noun-related and verb-related nodes may contain PMPs. PMP of a node specifies a possible preposition for that node.

We should add that, generating PMPs is the most challenging among the four types of MainParts, since it often requires a good understanding of the contextual knowledge, commonsense knowledge, and some sort of reasoning. For many cases, ambiguity in the text makes this task even more challenging. In the current implementation, we have only considered a very small set of patterns for generating PMPs and improving them are left for the future work.

Applying the mentioned 135 TD rules over the parse trees does not significantly increase the delay of generating MP tree comparing with the parse tree delay. However as shown next, MP trees require simpler and fewer patterns for being analyzed. Thus, they can serve as a good replacement of regular parse trees with a small effort in many existing text mining applications.

2.3 TextGraphs

Although the MPTs generated in the previous section are richer structures than parse trees, they are still not completely suitable for representing semantics in the text. This is mainly because of the tree-based structure of parse trees, which limits the number of direct connections of terms to other terms to only one. This problem is alleviated to some extend by dependency trees [CdJ10] in which words can be used as non-terminal nodes as well. However, dependency trees are still limiting, since:

- Dependency trees do not still capture multi-word terms and their role in the sentences.
- They still inherit the limitation of tree-based representation. Representing semantics in a more standard way requires a more expressive structure, e.g. graph-base structure.
- They do not provide a systematic way to represent the text ambiguity (e.g. confidences for the links).

To address these shortcomings, we introduce an even richer structured representation of text called *TextGraph*. *TextGraphs* are machine-friendly weighted graph structures, that represent grammatical connections between words and terms in the sentences, where terms are single- or multi-word phrases representing a thing or a concept. Each link in the TextGraphs is assigned a *confidence value* (weight) indicating SemScape's confidence on the correctness of the link and an *evidence count* indicating the frequency.

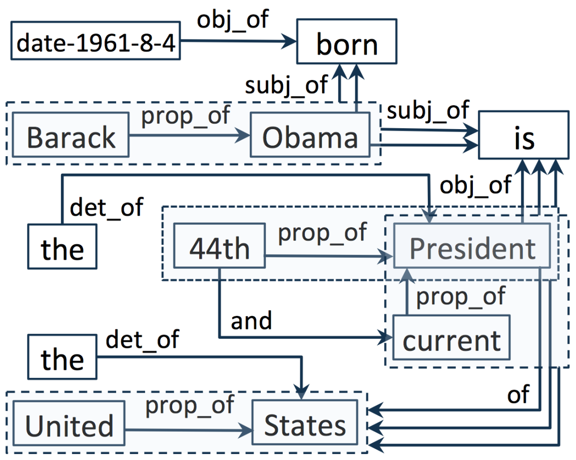


Figure 2.5: Part of the TextGraph for the first sentence in our running example.

A simplified TextGraph for the first sentence in our running example is shown in Figure 2.5. This graph connects words and terms to each others through grammatical links such as '*subj_of*', '*obj_of*', '*prop_of*', '*det_of*', etc. The complete list of link types in TextGraphs with their purposes is published in [MKI13b]. The graph also identifies multi-word (candidate) terms (as shown in dashed boxes) and their roles and links to other component of the sentences. For instance, the TextGraph contains two possible subjects for the verb '*is*' in sentence which are '*Barack Obama*' and '*Obama*'. These two at the same time play the subject role for the verb '*born*'.

With TextGraphs, more effective and efficient algorithms can be designed to extract knowledge from text by combining graph-based and statistical methods. This is mainly because:

- Representing the text in a graph-based structure makes it possible to take advantage of many existing graph-based mining algorithms.
- TextGraphs already contain *candidate terms* in text that facilitates the process of many applications requiring these terms.
- They are closer to the semantic of the text, by providing meaningful terms and their grammatical connections in a more standard graph-based structure.
- They are weighted which is very beneficial for dealing with ambiguity in natural languages and noisiness of the parsers.

These, in fact, hugely differentiate TextGraphs from their counterpart representation techniques such as parse trees and dependency trees. Later in next section, we also show how SemScape resolves pronouns and coreferences in the text to improve the TextGraphs. To generate the TextGraphs, we again take a pattern based approach to find grammatical relations between words and terms presented in the MP Trees. We refer to these relations as either *links* or *triples* throughout this section. To extract links, we created more than 270 TD rules which are all available in [Sem]. The generated triples by these rules are later combined into the final TextGraph structure. An examples for such TD rules is shown bellow, This rule aims at capturing the '*subject of*' ('*subj_of*') links:

Rule 2.

```
RULE subjectToVerb
{
   PATTERN:
               (S
                  (NP )
                  (VP ))
   RESULT (FO1='NMP', FO3='AVMP', conf=.9):
               <[0], 'subj_of', [1]>
   RESULT (FO1='NMP', FO3='PVMP', conf=.9):
               <[0], 'pobj_of', [1]>
}
```

The PATTERN in Rule 2 specifies a pattern in which a Noun Phrase (*NP*) is followed by a Verb Phrase (*VP*). This is the most general form of subject-to-verb link structure in parse trees. Similar to MP rules, Rule 2 indicates that for the matching trees, the NMPs of the noun phrase (NP) should be connected to the active verb main-part (AVMP) of the verb phrase (VP) to generate a *subj_of* link with confidence 0.9. Moreover, the NMPs of the noun phrase (NP) should be connected to the passive verb main-part (PVMP) of the verb phrase (VP) as a '*passive_object_of*' (*pobj_of*) link. For our running example, this rule captures links such as <*Obama*, *subj_of*, *is*> and <*Barack Obama*, *subj_of*, *is*>.

Note that, with the assist of MP information, this single rule can catch most of the *subj_of* and *pobj_of* links in different sentences without needing to know their lower level structure of the parse trees at nodes NP and VP. This is actually one of the most important gains in the SemScape framework, and dramatically decreases the number of required patterns/rules as well as simplifying the patterns required for any text mining application.

Each generated triple has a confidence (indicated by keyword '*conf*' in Rule 2) showing SemScape's confidence on the correctness of the link. If the same link is generated from different rules or from different MP trees, SemScape increases its correctness confidence as discussed in Section 2.3.2. After applying all rules to the MPTs of a sentence and generating the triples, we combine them into the final TextGraph (e.g. Figure 2.5). That is each sentence is converted into a separate TextGraph. In section 2.4, we show how SemScape improves the textGraph of each sentence by resolving its pronouns and coreferences with terms in the same or previous sentences. Next we discuss SemScape's approach for capturing syntactic exceptions in natural languages to improve the TextGraphs.

2.3.1 Support for Exceptions

Syntactic exceptions are the inseparable part of any natural languages. Although capturing exceptions can significantly enhance the quality of the text mining approaches, most of existing approaches do not provide an easy-to-use technique to handle exceptions. On the other hand, finding patterns with no exceptions in natural languages is a very tedious task, and a general pattern should be split down into many smaller patterns too avoid some exception cases. To simplify this process without needing to split our general patterns (e.g. Rule 2), SemScape uses patterns with negative confidence to specify exceptions and remove many of the incorrectly generated triples from the TextGraphs. Exception rules are considered superior to regular rules. That is if the same triple is extracted multiple times by different rules over the MP trees of the same sentence, and one of the extracted triples has a negative confidence, the triples with positive confidence will be eliminated.

To better illustrate this technique, consider the sentence "*In the woods are trees*". Since the sentence is in inverted form, which is not as common as the normal form, the parsers may not be able to recognize the structure correctly. For instance for the mentioned sentence, only one parse trees from the first three suggested parse trees by Stanford parser is correctly capturing the inverted form. Thus, our patterns may generate incorrect information from the incorrect parse trees. (<*trees*, *obj_of*, *are*> in our case). To eliminate this incorrect information, SemScape uses the following pattern:

```
———————————— Rule 3. ————————————
RULE subjectToVerb(Inverted)
{
   PATTERN:
               (SINV
                   (PP )
                   (VP )
                   (NP ))
   RESULT (FO1='NMP', FO3='AVMP', conf=-.9):
               <[2], 'obj_of', [1]>
   RESULT (FO1='NMP', FO3='AVMP', conf=.9):
               <[2], 'subj_of', [1]>
}
```

This pattern matches the correct parse tree and generates a triple with negative confidence (<*trees*, *obj_of*, *are*>) as well as the correct triple (<*trees*, *subj_of*, *are*>). Using the negative-weighted triple, SemScape can eliminate incorrect triples generated from incorrect parse trees and improve the final TextGraph for the sentence.

2.3.2 Combining Confidence Value

As already mentioned, every triple in TextGraph is assigned a *confidence* value. Since the same triple may be generated more than once (from different rules or different parse trees), we need to combine their confidence value c. Similar to [LWW11], the only assumption for the combination process is that evidences of the same piece of information are independent from each other. Thus, if a piece of information has been generated twice by different rules or from different MainPart Trees, once with confidence $c_1 \geq 0$, and once with $c_2 \geq 0$, we combine the confidence to $c = 1-(1-c_1)(1-c_2) = c_1+(1-c_1)c_2$. This new confidence is higher than both c_1 and c_2 which indicates the link's correctness probability is now higher. For each triple, we also count the number of time it has been generated and refer to it as the evidence frequency or count (e). We should note that, if one of the confidence values are negative for a particular triple (specifying an exception), we eliminate all the same triples with positive confidence as explained in previous section.

2.3.3 Enriching TextGraphs with Ontologies

Another important feature of SemScape is the ability to adapt an ontology and provide more related *candidate terms* with respect to the specified ontology. The Ontology here can be both domain-specific or domain independent in OWL. For simplicity, one can also specify a list of concepts instead of Ontology (e.g. the list of all subjects in Wikipedia). There are two main advantages of this feature which are discussed next.

The first advantage is to better control the volume of generated Noun MainParts. To understand how, we should note that for complex noun phrases (NPs) there might be several possible candidate terms. Not all of these candidate terms are useful or meaningful. Therefore, suggesting all of them as MPs may lead to a very large set of candidate terms which lowers the efficiency of the system. To prevent this problem, SemScape is made capable of utilizing an ontology, say O. Using O, SemScape only generates candidates terms that either i) contain less than three words, ii) are part of an existing concepts in O, or iii) contain a concept from O. In most simple cases this generates all possible candidate terms; however for many long noun phrases, this helps us reduce the size of the TextGraphs.

The second advantage of incorporating an ontology in the framework is to allow domain-specific applications to better utilize the framework. In a similar way, SemScape is also able to recognize *Named Enmities* [NS07] in the provided text which is not a trivial task in other approaches. It can also spot more related parts of the text with respect to the ontology which consequently improves system's robustness on dealing with noisy corpora. This feature is discussed in greater detail in [MKI13b].

2.3.4 Graph Domain Patterns

SemScape provides a graph-based query language, called *Graph Domain* or *GD* rules, to let users and applications mine the TextGraph. Although the format of GD rules is very similar to that of SPARQL [SPA12], its implementation is somehow different as explained in Section 2.5. In GD rules, we also introduce few extended features for simplifying the information mining process from TextGraphs. These features are introduced later in this section. Besides the external applications that can benefit from GD rules, SemScape uses GD rules for two purposes. The first one is completing and improving the TextGraphs using few GD rules. These patterns are often much easier to be expressed by GD rules than TD rules. The second purpose is to perform Coreference resolution which is the topic of the next section in which we provide examples of GD rules. Readers are also referred to [Sem], [MKI13b] and [MKI13a] for more examples of such rules.

We should add that GD rules are usually considered as in batch of patterns aiming at mining certain types of information. That is SemScape may be fed with sets of GD rules for different tasks (and applications). For instance as shown in [MKI13b], we fed the system with rules for generating ontological links in an automatic ontology generation system. Similar idea is used in [MGZ13c] to generate structured information from free text. Thus, once the GD rules are specified by an application, SemScape will apply them on all TextGraphs of the provided text, combine the resulted tuples, and report them back to the application.

In addition to the features supported by SPARQL, GD rules introduce the following features to ease the process of mining TextGraphs[2]:

[2]Readers are referred to [MKI13a] for examples on each new feature.

- Each rule may have multiple SELECT clauses to allow generating multiple pieces of information from same patterns.
- One may use keyword 'NEG' before the SELECT keyword to specifying exceptions similar to TD rules in section 2.3.1.
- One may use keyword 'NOT' before any triple in the WHERE clauses to indicate the absence of some links in the pattern, which requires a more complex expression in SPARQL.

2.4 Coreference Resolution

In textual documents, a pronoun or a reference may be used to refer to a term or a concept. These terms or concepts may be or may be not mentioned in the text itself. For instance in our running example word '*he*' in the second sentence is a pronoun referring to '*Barack Obama*' and '*Pres. Obama*' is a reference for '*Barack Obama*'. Resolving these types of references, called *Anaphora or Coreference Resolution*, is a very crucial step for improving the performance of any text-based knowledge acquisition technique. Unfortunately, this task is very challenging since it requires a huge amount of contextual knowledge, commonsense knowledge, and in many cases complex and ad hoc inferencing techniques [MML09]. The ambiguity of the natural languages also aggravates this issue.

In SemScape, we propose a new technique to resolve pronouns and references through the *Store Context* (*SC*) component. At its highest level, SC recognizes characters (also called *mentions*) in the text, learns contextual information about them from different resources, and uses this contextual information and a novel pattern-based technique to match characters to each other and resolve pronouns and references in the text. Each sentence in the text has its own SC in SemScape. The SC of the sentences of a paragraph are combined into the SC of the paragraph, and the SC of the paragraphs are combined to construct the SC of the entire document. In this way, we specify different resolution scopes. For instance the resolution for relative pronouns are considered the same sentence, while this scope for pronouns is the its entire paragraph. The scope for other references is considered to be the entire document. These steps are explained next in this section.

2.4.1 Recognizing Characters

The first step to construct the SC's structure is to recognize possible characters or mentions. This is a relatively easy task for us since TextGraphs already provide all the candidate terms (Noun MainParts) in text. We use these candidate terms as the characters of [the story of] the text. However, some characters in the text are more important (due to their role) than the others, and as a result they are more probable to be referred with pronouns or other references. To determine the importance of a character, each character is assigned an evidence count and a confidence weight. Whenever the same character is encountered in different roles (relations), we increase its evidence count by one and its confidence as explained in Section 2.3.2. We should mention that at this stage each occurrence of the same candidate term is considered as a separate character.

2.4.2 Mining Characters Context

After generating the list of characters, we gather information about some of their important and distinguishing attributes or properties. These properties are sometimes called *agreement* properties. Currently, we consider seven properties: *isMale* (if it is a male or a female), *isPerson* (if it is a person or not), *isOrganization* (if it is an organization or not), *isLocation* (if it is an geographical location or not), *isAnimal* (if it is an animal or not), *isObject* (if it is a thing or not), and *isPlural* (if it is plural word or not). Other properties could be added to this set to improve the final resolutions results. However, we found these the most useful and differentiating properties. We should add that most of the exiting works in this area only consider person, gender, and number as their agreement properties [Hob78, LL94, CE09, HK09].

For each property, SC uses a different source of information to estimate their value, which ranges between -1 and 1. Value 1 means 100% confidence that the property holds and value -1 means 100% confidence it does not (value 0 indicates no information about the property). As opposed to most similar approaches which use only true/false values for characters properties [Hob78, LL94, CE09, HK09], SC uses probability of being true or being false to better deal with uncertainty.

To evaluate the mentioned properties, we use the rich structured knowledge provided by Wikipedia's categorical information. For instance if one of the ancestor of a character is the '*Category:People*' category in Wikipedia, we set its *isPerson* property to 1. The same approach is used for *isOrganization*, *isLocation*, *isAnimal*, and *isObject* properties. This technique can be used for many potential properties that one may want to add to our initial list of properties. However, the main drawback of this technique is that for many characters, there is either no equivalent title in Wikipedia or no categorical information. Thus, we use the following heuristics to mine more contextual information on each character:

- We use VerbNet [KKR08] which for each verb v specifies its possible subject or object as either an organization, a person, an animal, or an object. For instance, in the sentence '*The agent was killed in a terrorist attack.*', due to the meaning of the verb '*killed*', '*agent*' is probably a person or an animal.
- For *isPlural*, we use the POS tags generated from the parser as well as some TD rules (e.g., terms containing the word '*and*' or '*or*' are considered plural).
- For *isMale*, we use lists of masculine (e.g., waiter, king, etc.) and feminine (e.g., waitress, queen, etc.) terms and lists of male or female proper first names.
- For *isPerson*, we use some POS tag information (e.g., NNP), our male and female first names lists, as well as some TD rules (e.g., any term *renaming* the words 'who' or 'whom' could be a person).
- We also have a list of animal names to add more evidence to *isAnimal*.

- For *isObject*, if there is evidence that a term is not a person and is not an animal, we increase the confidence that it is an object. Any term *renaming* the word 'which' could also be an object.

Combining Potentially Referencing Characters: Another new technique to improve our understanding about the values of the seven properties for different characters is to use taxonomical relations namely *type_of* relations between the characters. These relations are generated using our OntoHarvester system [MKI13b] and Hyponym information in WordNet [SR98]. The key idea is that if character γ_1 is *type_of* character γ_2 with confidence c, then γ_2 may be used as a coreference for γ_1 (in other words, γ_1 may be referred to as γ_2) with confidence c. For instance, since '*algebraic equation*' is *type_of* '*equation*', after the first time '*algebraic equation*' is mentioned in text, it can be referenced with the '*equation*' for the rest of text. This essentially means that γ_1 can inherit the properties of γ_2 and vice versa. In order to do so, for each property f we update its confidence value in γ_1 ($\gamma_1.f$) as follows:

$$\gamma_1.f = \left\{ \begin{array}{ll} \gamma_1.f + (1-\gamma_1.f)c\gamma_2.f & \text{if } \gamma_1.f \times \gamma_2.f \geq 0 \\ (\gamma_1.f + c\gamma_2.f)/2 & \text{if } \gamma_1.f \times \gamma_2.f < 0 \end{array} \right.$$

The idea of combining the confidence values is essentially the same as in Section 2.3.2 if properties' values have the same signs. If they have different signs, we simply take a weighted average on their values. By propagating the values of the properties for potentially coreference characters, we ease our later resolution technique which is based on the similarity (or *agreement*) of the character's properties.

2.4.3 Finding Patterns for Coreferences

Although in general resolving pronouns and other references only based on morphological structures in text is not feasible, there are few cases for which these structures may indicate a resolution. For instance consider the sentence "*Bob relieved himself telling him the truth.*". Clearly from the structure of the sentence, one can tell the reflexive pronoun‘*himself*’ refers to ‘*Bob*’ in this sentence. Here the pattern is that if the object of a verb is a reflexive pronoun, it always refers to the subject of the same verb. To capture such a pattern, we use the following GD rule:

```
———————————————— Rule 4. ————————————————
SELECT ( ?2 ‘CoRef’ ?1 )
WHERE {
    ?1 ‘subj_of’ ?3.
    ?2 ‘obj_of’ ?3.
    FILTER (regex(?2, ‘ˆitselfˆ|ˆherselfˆ|ˆhimselfˆ...’, ‘i’))
}
—————————————————————————————————————————-
```

In addition to reflexive pronouns, we use *predictive nominative* constructs, first exploited by [PD08], *appositive* and *role appositive* constructs [HK09], and *relative pronouns* construct [RLR10]. As can be seen, there are only very few such constructs that explicitly specify a resolution. However, there are many cases that a construct explicitly indicates two characters can NOT refer to each other (be each others resolution). For instance, in our earlier example, ‘*him*’ can refer to neither ‘*Bob*’ nor ‘*himself*’. We refer to such cases as *impossible resolutions*. In general the object and subject of most verbs can not be each other’s resolutions unless the object is a reflexive pronoun. This pattern is captured with the following GD rule in SemScape:

```
———————————————— Rule 5. ————————————————
SELECT ( ?2 ‘NoCoRef’ ?1 )
SELECT ( ?1 ‘NoCoRef’ ?2 )
WHERE {
    ?1 ‘subj_of’ ?3.
    ?2 ‘obj_of’ ?3.
    FILTER NOT (regex(?3, ‘ˆbeˆ|ˆbecomeˆ|ˆremainˆ...’, ‘i’))
    FILTER NOT (regex(?2, ‘ˆitselfˆ|ˆherselfˆ|ˆhimselfˆ...’, ‘i’))
}
—————————————————————————————————————————-
```

Some other obvious constructs indicating negative or impossible resolutions are terms connected through a preposition, terms for which one is part of the other, and terms connected with conjunctions such as ‘*and*’, ‘*or*’, ‘*except*’, etc. Currently, SemScape uses 64 GD rules (available at [Sem]) to extract possible and impossible resolutions between characters from the morphological information in text. For possible resolutions, we combine the characters as explained in the previous section. Inheriting the property's value is even more useful when one of the characters is a pronoun, since most of the mentioned seven properties for pronouns are easily inferrable. The impossible resolutions, on the other hand, are used in the next section where we explain how SemScape performs the final character resolution.

2.4.4 Resolving Characters

Say that a character γ_1 from a sentence s needs to be resolved. We calculate the similarity (see next paragraph) between γ_1 in s and all other non-pronoun characters in s that are not an impossible resolution for γ_1. Our studies shows that filtering by impossible resolution reduces the search space by more than half for each sentence. Similarity values larger than a predefined threshold are reported as resolutions. If no value exceeds the threshold, resolution search for γ_1 is continued among characters in the previous same-paragraph sentences, iteratively. In this way we take the *recency* into account. If γ_1 is a pronoun and no resolution is found for it in its paragraph, the search will be stopped; this should be a rare case. If γ_1 is not a pronoun and it is not resolved yet, sentences of the previous paragraph are also checked until resolutions are found or until the beginning of the document is reached. In other words, the scope for a pronoun's resolution is only its paragraph in SemScape, while the scope for other coreferences is the entire document.

Characters Similarity Measurements: To compute the similarity of two characters γ_1 and γ_2, we define the distance for property f as δ_f=$(\gamma_1.f-\gamma_2.f)/2$ $(-1\leq$ δ_f $\leq$1) and average for property f as μ_f=$(\gamma_1.f+\gamma_2.f)/2$ $(-1\leq$ μ_f $\leq$1) respectively. Thus, we compute the similarity of characters γ_1 and γ_2 by:

$$1-(\sum_{f\in F} d_f)/|F|$$

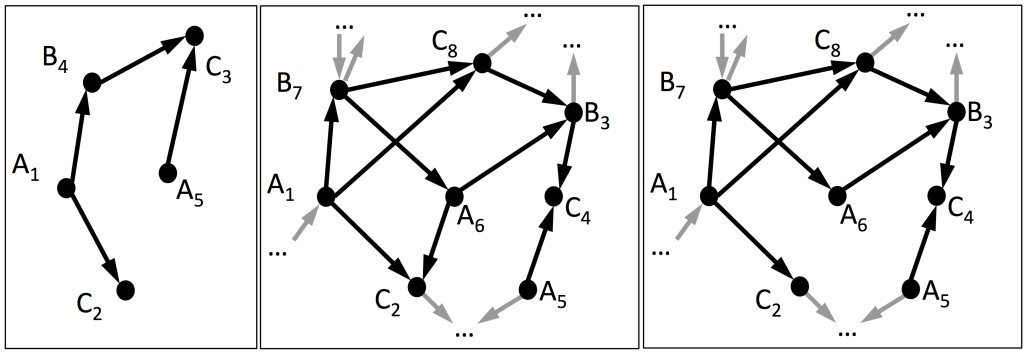


Figure 2.6: From left to right a) The GD query (q), b) a TextGraph that contains q (tg), and c) a TextGraph that does not contain q (tg').

where F is the set of all properties (i.e. $|F| = 7$ in our case) and d_f is the dissimilarity of γ_1 and γ_2 for property f which is defined as:

$$d_f = \frac{|\delta_f|}{|\mu_f| + 1}$$

It is easy to see that $0 \leq d_f \leq 1$. The above formula indicates that more difference ($|\delta_f|$) essentially means more dissimilarity, especially when the average is smaller (e.g., the dissimilarity of properties with values -.2 and .2 is more than the dissimilarity of properties with values .5 and .9 even though their distances are the same).

2.5 Graph Matching Optimization

As you may have already noticed, TextGraphs can be presented in RDF-like triple format, and thus queried by SPARQL. However, there is a subtle difference between TextGraphs and RDF graphs. As the TextGraphs are the graph representation of text, they may include several different nodes with the same name or label while nodes in RDF must have unique names. For instance in the TextGraph shown in Figure 2.5, there are two nodes with label '*the*'. This makes querying the TextGraphs using SPARQL inefficient as explained next.

To understand the above issue, consider the GD query (q) in Part a) of Figure 2.6. Five nodes with three different labels (A, B, and C) are connected through four links in this query. As mentioned earlier, to differentiate among nodes with the same label, SemScape assigns an ID to each node (e.g. 'A_1', 'A_5', and 'A_6' in Part b) of Figure 2.6 or '*the_8*' and '*the_14*' in Figure 2.5). To ease our discussions, we assume all edges have the same label, say '*link*'. Now consider two TextGraphs tg and tg' in Parts b) and c) of Figure 2.5. As can be seen, although both tg and tg' contain all the individual edges of the query graph q, only tg contains a subgraph matching q. To find such matches, q can be expressed with the following SPARQL query:

——————————- Rule 6. ——————————

```
SELECT *
WHERE {
    ?1 'link' ?2 .
    ?1 'link' ?4 .
    ?4 'link' ?3 .
    ?5 'link' ?3 .
    FILTER (strStarts(?1, 'A_'))
    FILTER (strStarts(?2, 'C_'))
    FILTER (strStarts(?3, 'C_'))
    FILTER (strStarts(?4, 'B_'))
    FILTER (strStarts(?5, 'A_'))
}
```

———————————————————————-

Although this approach perfectly works for GD query q using any SPARQL engine, it is very inefficient and slow, since it can not take advantage of any smart indexing techniques. Notice that the engine needs to traverse all triples for each where clause in the form of <?1 'link' ?3 .> and then filter them based on the specified FILTER commands. The same phenomenon can happen in other RDF resources, however it is much more frequent in TextGraphs due to too many same-label nodes.

To take advantage of the same-label nodes in our query engine, we first ignore all the IDs assigned by the SemScape to nodes and collapse both query graphs and TextGraphs by combining same-label nodes into a single node. The collapsed version of the examples in Figure 2.6 is depicted in Figure 2.7. As shown in Figure 2.7, for every link in the collapsed version, we store all the associated links in the actual graph for later retrieval. Now the collapsed version of query q, called q_c (Part a) Figure 2.7), can be answered through the following SPARQL query:

——————————— Rule 7. ———————————

```
SELECT *
WHERE {
    'A' 'link' 'C' .
    'A' 'link' 'B' .
    'B' 'link' 'C' .
}
```

——————————————————————————-

Answering collapsed queries (e.g. Rule 7.) over collapsed TextGraphs is much faster than regular queries (e.g. Rule 6.) over original TextGraphs. This is due to two main reasons: i) the collapsed queries are smaller in the number of edges which consequently reduce the number of join operations to answer them by a SPARQL engine and ii) there are more literal nodes and less variable nodes in the collapsed query graphs, which makes a better use of indexing optimizations. However, finding a match for a collapsed query on a collapsed TextGraph does not necessarily mean that the actual query matches the actual TextGraphs (e.g. q_c matches tg'_c but q does not match tg'.) A match, here, only indicates that all edges in the query graph have at least one corresponding edge in the TextGraph. This check, in fact, reduces the search space by eliminating many TextGraphs in which one or more of the specified edges in q is missing. For other cases, to verify that the matching edges construct the same graph structure specified by q, we use the following verification algorithm.

Verification: After finding the matches for the collapsed queries on the collapsed TextGraphs, we expand the matching edges to those in the actual edges. For instance, consider TextGraph tg and its collapsed version tg_c (Part b), Figure 2.7). Running the collapsed query q_c over tg_c will return the subgraph shown in Part a) Figure 2.8. Once we expand this subgraph to its corresponding subgraph in tg, we create a graph in which each edge matches at least one of the edges in the original query graph q. This subgrapg which is shown in Part b) Figure 2.8 is referred to as tg_m.

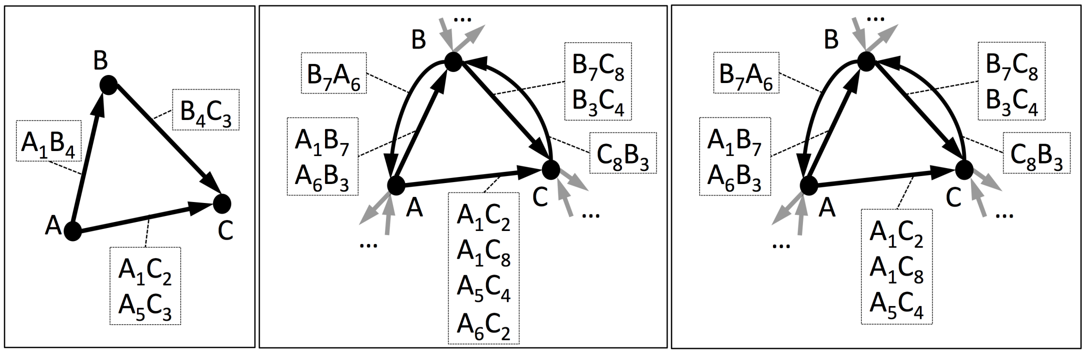


Figure 2.7: From left to right: Collapsed version of a) q (q_c), b) tg (tg_c), and c) tg' (tg'_c).

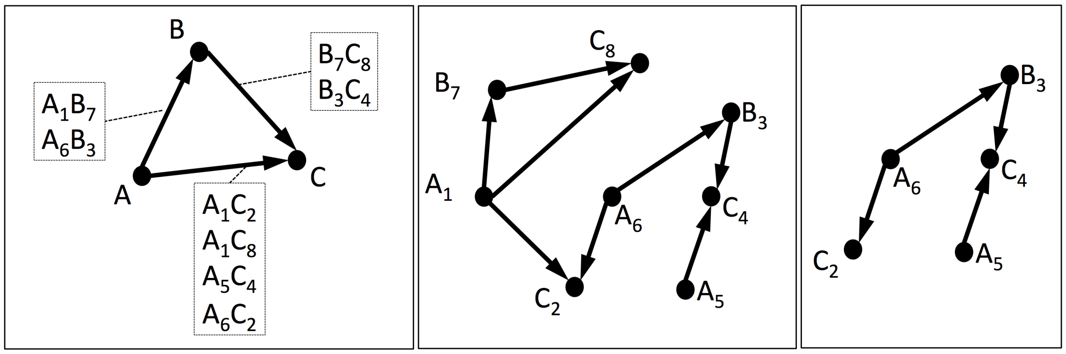


Figure 2.8: From left to right: a) the subgraph of tg_c that matches q_c, b) the expanded graph of part a, and c) the subgraph of tg that matches q.

Now, the goal is to verify that the query graph q matches with tg_m or part of it. To this end, any sub-graph matching technique can be utilized. Fortunately, since both nodes and links in TextGraphs are labeled and the queries are mostly small graphs, the problem is much simpler than the general subgraph matching (subgraph isomorphism) problem which is NP-Complete [Coo71]. In our current implementation, we use a simple recursive backtracking procedure.

CHAPTER 3

IBminer: Generating Structured Summaries from Text

Structured summaries of textual documents are playing an increasingly important role in Web information systems. In addition to being very valuable for human readers, this information has many applications in various information systems and in particular question answering systems such as Faceted Search [AT10], SWIPE [AZ12], and IQ [MRS11]. This information has also been used extensively to generate *Commonsense Knowledge Bases* (CKBs) [BLK09, HSB11, WW08] and domain specific knowledge bases (*Ontologies*) [BCS07, Dro03, PD10] in systems which support text summarization, document categorization, semantic search, and automatic essay grading.

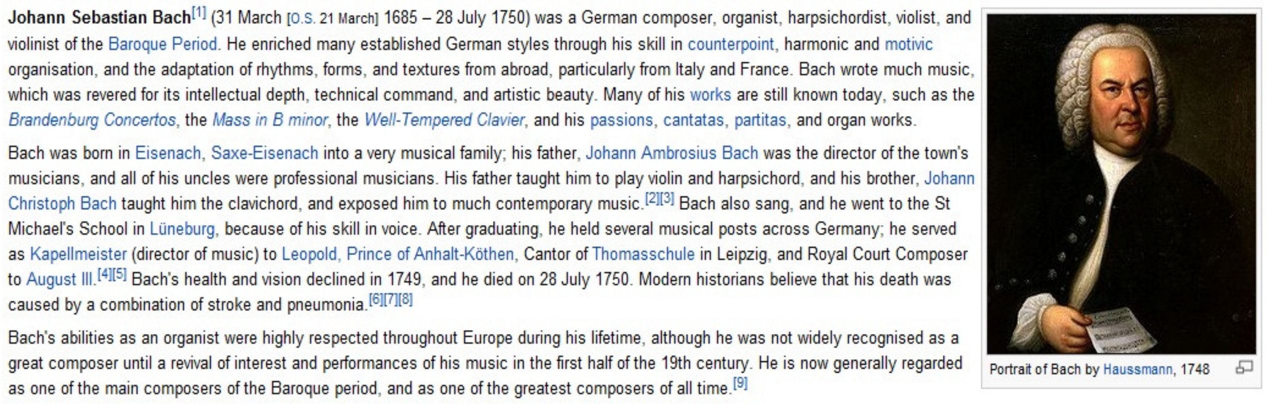
A prominent example of such structured summaries is provided by the *InfoBoxes* in Wikipedia pages [Wik12], which summarize the most important attributes and their values for the entities described in the pages. Unfortunately, since the current process of generating InfoBoxes is manual, and standard ontologies are often not used, InfoBoxes suffer from several issues [WW08]. The first problem is limited coverage, due to the fact that many pages in Wikipedia have incomplete InfoBoxes or no InfoBox at all—according to DBpedia, more than 40% of the pages in Wikipedia are missing their entire InfoBox. The second problem is that attributes may have several synonyms or aliases (e.g., 'birthdate', 'date of birth', and 'born'), which are used interchangeably, making the knowledge base seemingly inconsistent. Finally, the third problem is that, due to the manual construction, lots of erroneous or misspelled InfoBox triples (around 4% according to our studies) are created.

In recent years, several automatic techniques have been proposed to alleviate these issues. Many of such works only consider structured or semi-structured part of data [WW08, BEP08, SKW08, BLK09, HSB11]. Some approaches use limited form of NLP-based techniques to extract InfoBox-like information from text techniques [ECD04, YCB07, CBK10a, PGR10, LBN10, WLH11]. However, none of the mentioned efforts are able to take full advantage of the linguistic morphologies of the sentences in the text to generate high-quality structured information. As a result, the existing approaches are not able to exploit hidden information in the text, and are often incapable of extracting those pieces of information that are not mentioned frequently in the documents.

In this section, we describe our system, called *IBminer*, which addresses the mentioned issues by generating structured information in two main phases: First it extracts meaningful connections between mentioned terms in the text by using morphological structures in the text. These links are referred to as *semantic links* and represented as $<subject, link, value>$ triples. Then using a large body of categorical/taxonomical information (for $subject$s and $value$s) and using the original InfoBox triples in Wikipedia, *IBminer* automatically learns patterns for mapping the $link$ parts in semantic links to appropriate attribute names and generates the final InfoBox triples.

Being able to capture morphological structures in text through the SemScape framework and combining them with contextual information for terms and phrases in text, *IBminer* is effectively capable of deriving accurate information from rather small corpora. To improve the quality of the generated structured information, *IBminer* also employes an effective type-checking technique as well as an approach for dealing with exceptions in natural languages. Our studies in fact show that by only considering the abstract of the pages in Wikipedia, *IBminer* is able to generate structured information with more than 97% accuracy while improving the coverage of DBpedia by at least 27%. Moreover, since *IBminer* uses free text, it can be simply fed with textual information from any resources. Thus, it can improve the performance of many applications using structured information.

More specifically, we provide the following contributions in this section:


Johann Sebastian Bach[1] (31 March [O.S. 21 March] 1685 – 28 July 1750) was a German composer, organist, harpsichordist, violist, and violinist of the Baroque Period. He enriched many established German styles through his skill in counterpoint, harmonic and motivic organisation, and the adaptation of rhythms, forms, and textures from abroad, particularly from Italy and France. Bach wrote much music, which was revered for its intellectual depth, technical command, and artistic beauty. Many of his works are still known today, such as the *Brandenburg Concertos*, the *Mass in B minor*, the *Well-Tempered Clavier*, and his passions, cantatas, partitas, and organ works.

Bach was born in Eisenach, Saxe-Eisenach into a very musical family; his father, Johann Ambrosius Bach was the director of the town's musicians, and all of his uncles were professional musicians. His father taught him to play violin and harpsichord, and his brother, Johann Christoph Bach taught him the clavichord, and exposed him to much contemporary music.[2][3] Bach also sang, and he went to the St Michael's School in Lüneburg, because of his skill in voice. After graduating, he held several musical posts across Germany; he served as Kapellmeister (director of music) to Leopold, Prince of Anhalt-Köthen, Cantor of Thomasschule in Leipzig, and Royal Court Composer to August III.[4][5] Bach's health and vision declined in 1749, and he died on 28 July 1750. Modern historians believe that his death was caused by a combination of stroke and pneumonia.[6][7][8]

Bach's abilities as an organist were highly respected throughout Europe during his lifetime, although he was not widely recognised as a great composer until a revival of interest and performances of his music in the first half of the 19th century. He is now generally regarded as one of the main composers of the Baroque period, and as one of the greatest composers of all time.[9]

Portrait of Bach by Haussmann, 1748

Figure 3.1: Excerpts from the Wikipedia page for "*Johann Sebastian Bach*" (Retrieved on 5/7/12)

- We propose the *IBminer* system which extracts structured summary information (in the form of $<subject, attribute, value>$) from completely unstructured text. IBminer also introduces a very strong type-checking mechanism that automatically infers the acceptable value domains for each attribute. These value domains greatly improve the accuracy of *IBminer*'s final results.
- IBminer also suggests *attribute synonyms*, which may differ based on the taxonomical information for the subject and value of the triples. This mainly improves the consistency of the results. Moreover, our synonym structure can reconcile the different terminologies used for attribute names in the existing knowledge bases.
- We perform extensive experiments on a multi-core implementation of IBminer and evaluate its performance using the long abstracts in Wikipedia. We have also conducted experiments by evaluating the results of semantic search over *IBminer*'s results and those of DBpedia. To this end, we first created a set of popular queries and then provided the answers for the queries using these two knowledge bases, which showed at least 53% improvement on the number of found answers for the queries.

3.1 Overview and Example

After preprocessing the text and partitioning it into its paragraphs and sentences, *IB-miner* performs the following operations to achieve its high-level functionality of deriving InfoBoxes from free text:

- **Generating TextGraphs:** IBminer converts the sentences in the *TextGraphs* using SemScape (As explained Chapter 2).
- **Phase I:** Using SPARQL-like graph patterns on the TextGraphs, IBminer generates a set of initial triples called *semantic links* in the form of $<subject$, $link$, $value>$ where $subject$ is an entity, $value$ is either an entity or a value, and $link$ is a semantic relation between $subject$ and $value$. Each semantic link represents a small piece of information that is potentially useful to generate an InfoBox triple.
- **Phase II:** Using a large body of categorical/taxonomical information and learning from the original InfoBox triples in Wikipedia, IBminer generates an intermediate structure called *Potential Matches* (*PM*). *PM* is a very large set of automatically generated patterns which will be used to translate link names in the *semantic links* into appropriate attribute names and generate new InfoBox triples.

Both mentioned phases take advantage of pattern-based mining techniques. However, there is a huge difference between the two sets of patterns, used in these phases. In the former case, we use a limited set ($\approx$ 60) of domain-independent patterns which are designed manually. Since these patterns are only capturing morphological structure they can be used in any domain with no changes. Our experiments in this study also prove this claim. On the other hand, patterns in phase two are created automatically based on the contextual (taxonomical or categorical in our case) information available for each domain. They can easily adapt with new domains without any extra effort.

To ease the rest of our discussions, we use the following running example throughout this chapter:

Example: Consider the Wikipedia page for '*Johann Sebastian Bach*' (Figure 3.1). For '*Bach*' and many other classical musicians, the entire InfoBox is missing from the Wikipedia page. However, several pieces of information about *Bach* can be extracted from the accompanying text (e.g., *birthdate*, *death date*, *birth place*, *nationality*, and *occupations*).

After generating TextGraphs using SemScape (such as the one in Figure 3.2) for the sentences in the *Bach* page, IBminer employs a set of SPARQL-like queries called *Graph Domain* (GD) rules to extract information from TextGraphs in the form of triples, which are referred to as semantic links (e.g. <*Bach*, *was*, *composer*>, <*Bach*, *was*, *German*>, <*Bach*, *was of*, *Baroque Period*>, etc.).

For each generated semantic link $<s, l, v>$, IBminer maps the link name l to one or more known attribute names from the existing InfoBoxes based on the values of s, l, and v as well as categorical information of s and v. In the example above, the link 'was' in the triple $<Bach, was, composer>$ will be mapped to the attribute name '*occupation'* based on the knowledge that i) '$Bach$' belongs to the category '*people*', ii) '$composer$' belongs to the category '*occupation in music*', and iii) according to the existing examples, the link 'was' between these two categories is usually referred to as '*occupation*'.

The rest of this chapter is organized as follows: We first present the key parts of *IBminer*, Phase I in Section 3.2 and Phase II in Section 3.3. The actual implementation of *IBminer* in a multi-core environment is discussed in Section 3.4. In Section 3.5, we describe the results obtained from extensive experiments. Finally, we proceed with a discussion of related work in Section 3.6 followed by the Conclusion section.

3.2 Generating Semantic Links

The next step to generate InfoBoxes is to find the semantic connections between terms used in the TextGraphs. These connections are referred to as *semantic links* in our work,

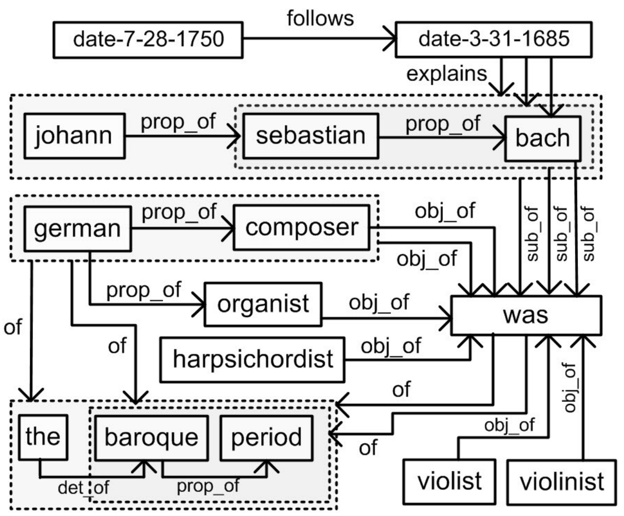


Figure 3.2: Part of the TextGraph for first sentence in Figure 3.1.

and will be used later to generate the final InfoBox triples. Before continuing, let us first define semantic links:

Semantic Link. Semantic link is a triple, denoted as $<subject, link, value>$ where $subject$ and $value$ are two candidate terms in a TextGraph connected through the phrase $link$. For instance, <*Bach*, *was*, *composer*>, <*Bach*, *was*, *German*>, and <*Bach*, *was*, *organist*> are three possible semantic links in our running example.

It is important to understand that with the above definition some of the links in the TextGraph are also semantic links (e.g. <*Date-3-31-1685*, *was*, *Bach*>). However, there are many more semantic links that can be extracted from the TextGraphs. This section explains how *IBminer* extracts such links with a pattern-based approach.

3.2.1 Extracting Semantic Links

To generate semantic links such as the ones introduced above, *IBminer* uses a limited set of carefully created graph-based patterns. These patterns, which are also called Graph Domain rules, capture common structures in the TextGraphs that indicate possible semantic links between candidate terms. Then from the matching sub-graphs for these patterns, *IBminer* constructs the semantic links. Semantic links that are created more than once from different patterns or different sentences are also merged as explained later in this section.

Graph Domain Rules. Given a TextGraph T, Graph Domain (GD) rule is a graph query executed on T which returns semantic links matching certain conditions. The format of GD rules is similar to SPARQL, with some extended key words for TextGraphs. To understand our GD rules and the process of using them to generate semantic links, we provide several examples in this section. We should stress that the GD rules in *IBminer* are completely generic and domain independent. Our experiments in Section 3.5 also verify this claim, where we applied these rules on three different domains and observed steady results for all the domains. In the current implementation of *IBminer*, we have created 59 such GD rules which are available for online access [Sem].

As shown in Figure 3.2, terms are represented by nodes in the graph and connected in various ways. For instance, the term '*Johann Sebastian Bach*' is connected to the term '*composer*' though a connecting node (verb) '*was*'. This is actually a very common pattern that can generate several small facts such as <*Johann Sebastian Bach*, *was*, *composer*>. Generating such a fact in bag of keyword approaches or even in shallow NLP-based techniques is indeed very challenging. *IBminer* uses the following GD rule to extract similar facts to the mentioned one:

———————————— Rule 1. ————————————
```
SELECT ( ?1 ?3 ?2 )
WHERE {
    ?1 "subj_of" ?3.
    ?2 "obj_of" ?3.
    NOT("not" "prop_of" ?3).
    NOT("no" "det_of" ?1).
    NOT("no" "det_of" ?2).
}
```
————————————————————————————————-

As depicted in the part a) of Figure 3.3, the pattern graph (WHERE clause) of Rule 1, specifies two nodes with names $?1$ and $?2$ which are connected to a third node ($?3$)

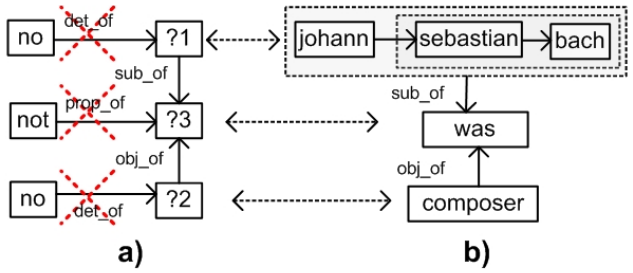


Figure 3.3: Part a) shows the graph pattern for Rule1, and b) depicts one of the possible matches for this pattern.

respectively with ‘subject_of’ (sub_of) and ‘object_of’ (obj_of) links. One possible match for this pattern in the TextGraph of our running example is shown in part b) of Figure 3.3. Due to the structure of the TextGraphs, matching multi-word terms (hyper nodes in the TextGraphs) to the nodes in the patterns is an easy task for *IBminer* whereas this is actually a challenging issue in works such as [WW10] which are based on dependency parse trees. Using the SELECT clause in Rule 1, the rule returns several triples for our running example such as:

- <*Johann Sebastian Bach*, *was*, *composer*>,
- <*Sebastian Bach*, *was*, *composer*>,
- <*Bach*, *was*, *composer*>,
- <*Johann Sebastian Bach*, *was*, *organist*>,
- <*Sebastian Bach*, *was*, *organist*>, etc.

Section 3.2.2 discusses how *IBminer* assigns confidence value to each of the above triples. As can be seen, we have made the syntax of the GD rules similar to that of SPARQL [SPA12]. The SELECT clauses in GD pattern may indicate more than one triple. However, we added the notation of ‘NOT’ to indicate the absence of some links in the pattern, which requires a more complex expression in SPARQL.

As a more complex example, consider Rule 2 which captures *Bach*’s nationality in our running example. This rule looks for a ‘*to be*’ verb (?3) with an object (?2), where the object has an adjective (?4). If such a pattern is found, the subject (?1) of the verb is connected to that adjective (?4) via the verb (?3). The following results are generated by this rule for our running example:

- <*Johann Sebastian Bach*, *was*, *German*>,
- <*Sebastian Bach*, *was*, *German*>,
- <*Bach*, *was*, *German*>, etc.

```
———————————————— Rule 2. ————————————————
SELECT ( ?1 ?3 ?4 )
WHERE {
    ?1 “subj_of” ?3.
    ?2 “obj_of” ?3.
    ?4 “prop_of” ?2.
    ?4 “POS_Tag” ?5.
    NOT(”not” “prop_of” ?3).
    NOT(”no” “det_of” ?1).
    FILTER (regex(?3, “^am^|^is^|^are^|^was^|^were^|^be|^...”, “i”))
    FILTER (regex(?4, “^JJ^|^ADJP^”, “i”))
}
————————————————————————————————————————-
```

Note that by using POS-Tag information which carried up from parse trees to the TextGraph and the FILTER keyword which accepts *Regular Expressions*, it is easy to verify that the node matching to the variable $?4$ is an adjective. Without this filter, Rule 2 will generate wrong triples such as <*Bach*, *was*, *music*> for sentence “*Bach was a music composer.*”.

Many of the semantic links in text can be found in prepositional phrases. In order to capture this type of information, we have considered rules such as the following one[1].

```
———————————————— Rule 3. ————————————————
SELECT ( ?1 ?3+?4 ?2 )
WHERE {
    ?1 “subj_of” ?3.
    ?2 ?4 ?3.
    NOT(”no” “det_of” ?1).
```

[1] Similar rules are generated for cases such as passive sentences, sentences including verbs with modifier, prepositions that are connected to nouns, etc.

```
    NOT(”not” “prop_of” ?3).
    FILTER (regex(?4, “^to^|^on^|^in^|^with^|^from^|^at^|...”, “i”))
}
```

This rule captures semantic links in prepositional phrases that are connected to a verb. The link in this case is specified by ?3+?4 which indicates the concatenation of the verb and the preposition. For instance, it generates the following triples from the second paragraph in Figure 3.1.

- <*he*, *died on*, *date-7-28-1750*>,
- <*he*, *served as*, *KapellMeister*>,
- <*he*, *served to*, *Leopold*>, etc.
- <*he*, *went to*, *St Michael's School*>, etc.

For the rest of the chapter, we refer to this set of semantic links generated by IBminer as T_n. To unify similar link names, *IBminer* uses their stems as new link names for triples in T_n. To this end, *IBminer* replicates each triple by replacing the link name with its stem. The stem and synonym information is provided by WordNet [Wor12]. As an example for the semantic link <*Johann Sebastian Bach*, *was*, *composer*>, *IBminer* also generates the triple <*Johann Sebastian Bach*, *be*, *composer*>. This simple expansion technique improves the process of interpreting attributes names (Section 3.3) by populating verb tenses and word variations.

3.2.2 Computing Links Confidence

As discussed in Section 2.3.2, every edge in TextGraph is assigned with *confidence* values. To compute the confidence value of the generated semantic links, we simply use the minimum confidence among the matching edges of TextGraph as the final confidence of each triple. The intuition behind using the minimum is that if one of the links in the matching subgraph is of low confidence, it makes the entire match low confident.

Since the same semantic link may be generated more than once from different rules or TextGraphs, we need to combine its evidence count e and confidence value c. Similar to [LWW11], the only assumption for the combination process is that evidences of the same piece of information are independent from each other. Thus, if a piece of information has been generated twice, once with evidence count and confidence of e_1 and c_1, and once with e_2 and c_2, we combine the evidence count to $e_1 + e_2$ and the confidence to $1-(1-c_1)(1-c_2)= c_1+(1-c_1)c_2$. This new confidence is higher than both c_1 and c_2 which indicates the link's correctness probability is now higher.

3.2.3 Support for Exceptions

Exceptions are an inseparable part of any natural languages. It is almost impossible to create a pattern that is always capturing the correct or intended morphological structure in natural languages. There are several cases of exceptions in which the patterns fail to capture the correct structure. Thus, supporting for these exceptions is a vital component to improve the accuracy in NLP-based text mining techniques. This is unfortunately overlooked in many previous works. However in *IBminer*, we are able to define cases of exceptions and exclude them for the other patterns. In this way many wrong results, generated because of them are eliminated.

Consider the first sentence in the second paragraph of Figure 3.1, which says "*Bach was born in Eisenach, ...*". Due to the structure of the TextGraph for this sentence, Rule 3 generates triple <*Bach*, *born in*, *Eisenach*> which is not exactly correct. The more accurate triple <*Bach*, *was born in*, *Eisenach*>) can be generated using Rule 4. The pattern in Rule 4 is more specific than that of Rule 3, and cancels the inaccurate triple generated by Rule 3, using the negative select clause (shown by keyword NEG). If negative triple t is generated from a sentence by any rule, it will cancel all the generated [positive] t by other rules for that sentence. However, it does not cancel the positive t's that are generated from other sentences. Thus in our running example, triple <*Bach*, *born in*, *Eisenach*> will be replaced by <*Bach*, *was born in*, *Eisenach*>.

```
———————————— Rule 4. ————————————
SELECT
    ( ?1 ?5+?3+?4 ?2 )
    NEG( ?1 ?3+?4 ?2 )
WHERE {
    ?1 “subj_of” ?3.
    ?2 ?4 ?3.
    ?5 “prop_of” ?3.
    NOT(”no” “det_of” ?1).
    NOT(”not” “prop_of” ?3).
    FILTER (regex(?4, “^to^|^on^|^in^|^with^|^from^|^at^|...”, “i”))
}
————————————————————————————————————-
```

3.2.4 Co-reference Resolution

In many of the generated triples, the subject (or value) is either a co-reference or a pronoun referring to the actual subject. For example, the triple <*Bach*, *was born in*, *Eisenach*> generated from the second paragraph in Figure 3.1 basically indicates the birth place of ‘*Bach*’, but which *Bach* it is referring to? Now by examining previous sentences, it can be inferred that ‘*Bach*’ is a co-reference for ‘*Johann Sebastian Bach*’. Similarly for semantic link <*he*, *died on*, *date-7-28-1750*> from the same rule, ‘*he*’ is a pronoun referring to ‘*Johann Sebastian Bach*’, which needs to be resolved. Fortunately, *SemScape* provides a way to resolve some of these co-references and pronouns used in the sentences [MKI13a]. Thus, for any generated triple such as $<s, l, v>$ with confidence c, if we find a resolution for s, say s', with similarity c', we will add the new triple $<s', l, v>$ with confidence $c \times c'$ to our semantic links set (T_n). We use the same approach to resolve v as needed. At the end of this phase, semantic links containing a pronoun, and those where an interrogative pronoun remains in their subject or value parts are removed from T_n.

3.3 Mapping Links to Attributes

In phase II, *IBminer* uses contextual information about subjects and values in the semantic links to map the link names in T_n to attribute names used in the original InfoBoxes. In this process, many of the inaccurate or irrelevant triples in T_n will be dropped since no good maps for their links can be found. For the rest of this chapter, we refer to the set of triples taken from the original InfoBoxes in WikiPedia as T_i.

The key idea in mapping the link name of a semantic link (T_n) to the attribute name in current InfoBoxes (in T_i) is to learn the attribute mapping by example. To understand the concept of attribute mapping and the intuition behind *IBminer* approach to create this mapping, we continue with some examples. Consider the following two semantic links which have been generated from the TextGraphs in the previous section:

- <*Johann Sebastian Bach*, *was*, *composer*>
- <*Johann Sebastian Bach*, *was*, *German*>

Although the attribute names in both triples denote the same term, '*was*', they connect the subjects and values with completely different relations. As the attribute names in the actual InfoBox triples (T_i) suggest, the first '*was*' is mostly interpreted as '*occupation*' while the second one is usually called '*nationality*'. This kind of interpretation for the link names in semantic links based on the best match in existing InfoBoxes is referred to as *attribute mapping*. The difference in the context of the subjects may also imply different mappings. Consider the following triples:

- <*Johann Sebastian Bach*, *be*, *composer*>
- <*songwriter*, *be*, *composer*>

This time '*occupation*' is not the best interpretation for the second '*be*', and a better interpretation would be '*typeOf*'. Thus, the meaning or interpretation of a link in semantic link <s, l, v> depends not only on the link name (l) but also on the taxonomical/categorical information of the subject (s) and the value (v). Under this intuition, to correctly map link name l to attribute names used in T_i, *IBminer* considers the categorical information of the subject and value provided by Wikipedia under *categories* section. For instance, in the triple <*Johann Sebastian Bach*, *was*, *composer*>, knowing that 1) '*Johann Sebastian Bach*' is in category '*people*', 2) '*composer*' is in category '*occupations in music*', and 3) according to the matching examples, link '*was*' between these two categories is mostly called '*occupation*' in T_i, we can infer the new InfoBox triple <*Johann Sebastian Bach*, *occupation*, *composer*>. Next, we formally explain this technique for attribute mapping.

3.3.1 Generating Potential Matches

To map links in T_n to attributes in T_i, *IBminer* constructs a structure called *Potential Matches* (*PM*) by learning from the existing examples. To understand this structure, let us start with the definition of triple match.

Matching triples: Given two triples $t_n =< s_1, l, v_1 >$ and $t_i =< s_2, \alpha, v_2 >$ where $t_n \in T_n$ and $t_i \in T_i$, we say t_n and t_i match each other iff $s_1 = s_2$ and $v_1 = v_2$.

For instance, <*Johann Sebastian Bach*, *was*, *German*> from semantic links (T_n) matches <*Johann Sebastian Bach*, *nationality*, *German*> from InfoBoxes triples (T_i), since both subjects and both values in the two triples are the same. The set of all triples in T_n that match with at least one triple in T_i is referred to as T_m. Using such matching triples between T_i and T_n, *IBminer* automatically generates a set of patterns in a structure called *Potential Matches* or *PM* which is defined as follows:

Potential Matches: Potential Matches are records in the form of <c_s, l, c_v> : α, each associated with a confidence value c and an evidence frequency e. Each record in *PM* indicates that if a subject from category c_s is connected to a value from category c_v with link l in T_n, α may be a map for l with confidence c and evidence e.

As an example, consider *PM* record $<$*people*, *was*, *occupations in music*$>$: *occupation*. This record specifies a simple pattern indicating the link name '*was*' between a subject from '*people*' category to a value from '*occupations in music*' category may be interpreted as the attribute '*occupation*'. Next we explain how *IBminer* used triples in T_m to generate these patterns and build the *PM* structure.

Assume $t_m = <s, l, v>$ ($t_m \in T_m$) with confidence c_m matches $t_i = <s, \alpha, v>$ ($t_i \in T_i$). In the following, $C_s = \{c_{s1}, c_{s2}, ...\}$ and $C_v = \{c_{v1}, c_{v2}, ...\}$ denote the categorical information respectively for s and v. This categorical information is retrieved from Wikipedia. If no category is found for a subject (or a value), we will consider them to be in the most general categories (e.g., '*Category:Things*'). Moreover, in addition to the direct categories, C_s and C_v may contain indirect (more general) categories that will be described in next section. We say attribute α is a *potential match* for link l from any category in C_s to any category in C_v with confidence c_m and evidence count 1. To construct the final list of potential matches (*PM*), for each $c_s \in C_s$ and $c_v \in C_v$, we add the following record to the *PM* structure:

$$<c_s, l, c_v> : \alpha$$

For each newly generated *PM* record, confidence value c is initiated by c_m and evidence frequency e is initiated by 1. It is worthy to mention that the number of potential matches for t_m is $|C_s| \times |C_v|$ where $|C_x|$ is the number of categories for the entity x. If we encounter more evidence for the same record in *PM*, *IBminer* increases its confidence and evidence by combining the records as explained in Section 3.2.2. At the end of this step, *PM* will contain a big list of potential matches with their confidence values and evidence frequencies. In Section 3.3.3, we explain how *PM* is used to generate the final InfoBox triples by attribute mapping.

3.3.2 Selecting Best Categories

A very important issue in generating potential matches is the quality and quantity of the categories for the subjects and values. The direct categories provided for most of the subjects are too specific and only a few subjects are listed in each of these categories. As shown in Subsection 3.5, generating the potential matches over direct categories does not generalize the matches for newly observed subjects. On the other hand, exhaustively adding all the indirect (or ancestor) categories will generate too many inaccurate potential matches and significantly increase the processing time. For instance considering only four levels of categories in Wikipedia's taxonomy, the subject '*Johann Sebastian Bach*' belongs to 422 categories. In this list, there are some useful indirect categories such as '*German Musicians*' and '*German Entertainers*', as well as many categories which are either too general or inaccurate (e.g. '*People by Historical Ethnicity*' and '*Centuries in Germany*'). Considering the same issue for the value part, hundreds of thousands of potential matches may be generated for a single triple in T_m. This issue not only wastes our resources, but also impacts the accuracy of the final results.

To address this issue, we use a flow-driven technique to rank all the categories to which entity s belongs, and then select the best N_C categories. The main intuition is to propagate flows or weights through different paths from s to each of its categories. The categories receiving more weights are considered to be more related to s. Now, L being the number of allowed ancestor levels, we create the categorical structure for s up to L levels. Starting with node s as the root of this structure and assigning weight 1.0 to it, we iteratively select the closest node to s, which has not been processed yet, propagate its weight to its parent categories, and mark it as processed. To propagate weights of node c_i with k ancestors, we increment the current weights of each k ancestors with w_i/k, where w_i is the current weight of node c_i. Although w_i may change even after c_i is processed, we will not re-process c_i after any further updates on its weight to facilitate the algorithm. After propagating the weight to all the nodes, we select the N_C categories with the highest weight.

3.3.3 Generating InfoBox Triples

Once *PM* is generated, *IBminer* uses it to map the link names in semantic links (T_n) into the attribute names in InfoBoxes (T_i). Let t_n=$<s, l, v>$ ($t_n \in T_n$) be the triple whose link (l) needs to be mapped, where s and v have category sets $C_s = \{c_{s1}, c_{s2}, ...\}$ and $C_v = \{c_{v1}, c_{v2}, ...\}$ respectively. The key idea is to find all possible attribute mappings for t_n, and pick the attribute with the highest aggregate frequency and confidence as explained bellow.

To map l, for each $c_s \in C_s$ and $c_v \in C_v$, *IBminer* finds all potential matches such as $<c_s$, l, $c_v>$: α_i. The resulting set of potential matches are then grouped by the InfoBox attribute names, α_i's. For each group we compute the aggregate confidence and evidence frequency of the matches using the technique described in Section 3.2.2. At this point, *IBminer* uses two thresholds to remove potential matches that are very low-confidence (named τ_c) or infrequent (named τ_e), as discussed in Section 3.5. Next, *IBminer* filters the remaining results by a very effective type-checking technique explained in Subsection **??**. The few matched left in the list are called *ranked matches*. Then, we consider the one with the largest evidence count, say pm, as the only attribute map and report the new InfoBox triple $<t_n.s$, $pm.a_i$, $t_n.v>$ with confidence $t_n.c \times pm.c$ and evidence $t_n.e$. In Subsection 3.3.5, we show how the remaining ranked matches are used to find secondary matches.

Normalized Weight: Some attributes (such as '*occupation*') have been frequently used in T_i, while many attributes (such as '*headquarter*') are not used as frequently as others. This imbalance is directly transferred into the PM structure. Thus, the generated InfoBox triples for more frequent attributes may have much higher evidence frequency than those for less frequent attributes. That is if we use very high τ_e, many of correct results for less frequently observed attributes in T_i will be eliminated. On the other hand, using low τ_e will result in accepting many wrong triples for more frequent attributes. To avoid this, once all the Best Matches are generated, we normalize the evidence frequency of matches for attribute α by dividing it by α_{avg_e}, where α_{avg_e} is the average frequency of the correctly generated triples with attribute α. The correctly generated triples are those in the initial KB (T_i) that are also generated by *IBminer*. Similarly, we normalize the confidence of all matches, and compute a single normalized weight by multiplying the normalized evidence and confidence. In this way, we only need to consider a single threshold to tune the precision and recall of the approach. We refer to this threshold as *normalized threshold* (τ).

3.3.4 Type-checking

Attributes may take values from specific domains. For instance, attribute '*origin*' may accept only values from '*geopolitical locations*' domain. This sort of information on domain of values for attributes can be used as a simple, yet effective type-checking technique which in turn improves accuracy of the final results. For example, the value '*German*' fails the type-checking for attribute '*origin*' since in the original InfoBoxes they are not used together at all. On the other hand, '*German*' passes the check for attribute '*nationality*' since in several instances these two are used in the same triple in the original InfoBoxes. To automatically identify the correct domain for values of a given attribute, currently *IBminer* learns from T_i triples. That is, for each attribute α, *IBminer* counts the number of times that any value, say v, is used in a triple in T_i with attribute α. This number is called the *value rank* of v in α's domain. Value ranks are then used to verify the correctness of the generated results.

One of the drawbacks of type-checking is that if we eliminate all the triples with wrong value types, we will never find new values for some of the attributes. For instance, attribute *short description* accepts a variety of values which might not be listed in its domain. To avoid this issue, *IBminer* performs type checking on ranked matches for t_n, only if $t_n.v$ belongs to the domain of one (or more) attribute in the ranked matches. For instance assume for link '*was*' in semantic link <*Johann Sebastian Bach, was, organist*>, two attributes '*occupation*' and '*instrument*' are suggested using potential matches. Since value '*organist*' is frequently used with attribute '*occupation*' but not with attribute '*instrument*', our type checking will cancel the latter and only accepts '*occupation*' as the final attribute map. On the other hand, assume for link '*was*' in semantic link <*Johann Sebastian Bach*, *was*, *organist of Baroque Period*>, two attributes '*short description*' and '*occupation*' are suggested. This time value '*organist of Baroque Period*' has not co-occurred with any of the two attributes, and thus the type checking will not eliminate any of the maps.

Finally for generic data types such as integers, floating points, dates, and URL addresses, *IBminer* records the number of times any value from each data type is used with an attribute to do a more effective type-checking. For instance, if for an attribute, say height, mostly integer or floating point values are used, the type checking will cancel any other value types (e.g. strings, dates, etc.) for this attribute.

3.3.5 Suggesting Secondary Matches

As mentioned earlier, *IBminer* may find more than one ranked match for a triple in T_n. Intuitively, such cases are implying a possible synonym for the mapped attributes. Synonyms play a crucial role in unifying different terminologies used in Wikipedia or other date sources. As an example, consider the triple <*Johann Sebastian Bach*, *was born in*, *Eisenach*> in T_n. The technique mentioned in Section 3.3.1 finds three ranked matches for link '*was born in*'. These matches are '*born*', '*birthPlace*', and '*origin*'. Although all of these are correct matches, *IBminer* only picks the most evident one ('*born*' in this case) since for many cases the less evident ranked matches are wrong.

To verify which of the secondary ranked matches are actually correct, *IBminer* uses a very similar idea to that for *PM*. The idea is that if two attributes are synonyms, they are usually expressed in similar ways in the text (e.g. attributes '*birthdate*' and '*dateOfBirth*' are synonyms and they are both presented in the text with links such as '*was born on*', '*born on*', or '*birthdate is*'). Thus, *IBminer* constructs a structure, called

Potential Attribute Synonyms (*PAS*), to count the number of times each pair of attributes in the InfoBoxes are presented in the text in the same form (i.e. with the same link). These numbers are then used to compute the probability of that any given two attributes are synonyms.

In order to avoid duplication, we avoid further discussion on the PAS structure and leave it for Chapter 5 and more specifically Section 5.3, where we discuss our extended context-aware synonym suggestion system for improving the consistency of the generated knowledge base [MGZ13a, MGZ13b].

3.4 Implementation

Figure 3.4 sketches the architecture of a multi-core implementation of the *IBminer* system. This system is mainly comprised of four modules: *TextGraph Generator*, *Potential Match Generator*, *InfoBox Triples Generator*, and the *Knowledge Base*. Next we explain each of the these modules and their functions:

TextGraph Generator: *TextGraph Generator* performs two main tasks: i) converting text into TextGraphs and ii) capturing the *Semantic Links* from TextGraphs using GD rules. Both tasks above are very time consuming due to employing deep NLP and exhaustive pattern matching techniques. However, the tasks are easily parallelizable, since they can be performed separately for every document. Thus, *IBminer* instantiates several instances of SemScape and frequently assign them a small set of articles to handle. the results are integrated by combining duplicate triples generated from different documents at the end.

Potential Match Generator: As its name conveys, *Potential Match Generator* is responsible for generating the PM structure (and PAS structure). This task is not as time consuming as the others in *IBminer*, however implementing it in parallel is beneficial. To this end, small sets of semantic links are sent to each core and then the generated PM

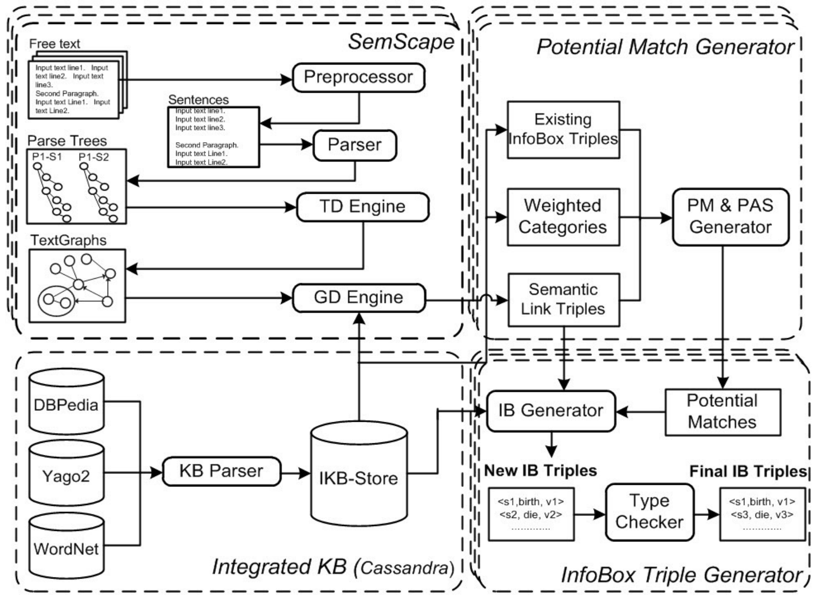


Figure 3.4: The architecture of *IBminer*.

from all cores are gathered and eventually integrated by the *PM Generator* Component. The resulting PM (and PAS) are also stored in our knowledge base for later use.

InfoBox Triples Generator: Using the PM structure and the semantic links, *InfoBox Triples Generator* generates the final InfoBox triples as explained in Section 3.3.3 and store them in our knowledge base. Similar to the previous module, *InfoBox Triples Generator* is parallelized to take advantage of multiple cores.

Knowledge Base: The *knowledge base* module is mainly responsible for storing the initial knowledge base, the intermediate results, the PM and PAS structures, and the final results of IBminer. Our knowledge base, also known as *IKBstore*, is stored in an Apache Cassandra database [Cas] which provides efficient key/value store as well as the ability to handle large amount of data. We should add that, since Cassandra resides in a separate machine, *IBminer* sets up its own cache module which locally stores very frequently accessed knowledge pieces. This is crucial to improve the performance of accessing *IKBstore*.

Although we use only DBpedia in this study, in the current implementation of

IKBstore, we integrate DBpedia [BLK09], YaGo2 [HSB11], MusicBranez [MUS], GeoNames [GEO], and WordNet [SR98] into a general integrated knowledge base [MGZ13a, MGZ13c]. This knowledge base is used in our structured query answering system which is based on SWIPE [AZ12].

Although for the current study, multi-core implementation of IBminer was enough, we have also implemented our IBminer system using MapReduce architecture [DG08] over cluster of machines in Amazon Web Service, EC2. The parallelization idea for this implantation is very similar to those mentioned above.

3.5 Experimental Results

To evaluate IBminer, we used both manual and automatic approaches and obtained the results presented in this section, where, after introducing the data sets used in this study, we present an in-depth evaluation of the precision and recall of IBminer for the Musician dataset (Section 3.5.2). Then in Sections 3.5.3 and 3.5.4, we respectively study the impact of category selection and domain change on IBminer's results. Finally in Section 3.5.5, we compare IBminer with alternative approaches using application-oriented metrics. Thus, we compare IBminer with the baseline DBpedia and with iPopulator [LBN10], a system that is an excellent representative of the current state of the art in extracting structured data from text.

All our experiments were performed on a single machine with 16 cores of 2.27GHz and 16GB of main memory, running under Ubuntu12 OS. On the average, our system spent 3.07 seconds for parsing each sentence on a single CPU. Thus, using 16 cores, we were able to parse and generate *semantic links* for 5.2 sentences per second. All the data sets and the results discussed in this section are available for access in [Sem].

Dataset Name	**Subjects #**	**InfoBox Triples #**	**Subjects with Abstract**	**Subjects with InfoBox**	**Sentences per Abstract**
Musicians	65835	687184	65476	52339	8.4
Actors	52710	670296	52594	50212	6.2
Institutes	86163	952283	84690	54861	5.9

Table 3.1: Description of data sets used in experiments.

3.5.1 Data Sets

To perform our evaluation studies, we created three data sets for the domains of Musicians, Actors, and Institutes from the subjects and their accompanying abstracts in Wikipedia. These data sets do not share any subject, and in total they cover around 7.9% of Wikipedia subjects. To build each data set, we started from a general WikiPedia category describing the domain of the data set (e.g. *Category:Musicians* for Musicians data set) and collected all the Wikipedia pages in this category or any of its descendant categories up to four levels. Table 3.1 provides more details on each data set.

As for our initial knowledge base, we used *DBpedia* which organizes Wikipedia's InfoBoxes by using structured data available in Wikipedia documents, and thus provides a better starting point for *IBminer*. Then we used IBminer to mine the text of the entire long abstract of each article to create our data set. We should stress that no structured data was associated with the text in our experiments. In other words, for each subject one can also collect text from different sources, and employ *IBminer* to generate InfoBox triples from that text.

3.5.2 Precision/Recall Performance

In our experiment, we built the Potential Match from 80% of the Musicians data set, using 50 categories ($N_C = 50$) and 4 levels ($L = 4$). The total number of initial triples for this chunk of data set is $|T_n| = 3.7M$, while $52.9K$ of them match with original InfoBoxes ($|T_m| = 52.9K$). Using these *PM*s and the initial triples generated from the remaining 20% of the abstracts, we generated our InfoBox triples without any frequency and confidence constrains (i.e. $\tau_e = 0$ and $\tau_c = 0.0$). Later in this section, we analyze the effect of N_C and L selection on the results.

To estimate the accuracy of the final triples, we randomly selected 5% of the generated triples ($\approx 42K$) and carefully graded them by matching against their abstracts. Many similar systems such as [CBK10a, HSB11, WLW12] have also used manual grading due to the lack of good benchmarks. Recall estimation is also very hard since we again do not know what portion of the InfoBoxes in Wikipedia are covered or mentioned in its accompanying text (long abstract for our case). Thus, we only used those InfoBox triples which match at least one of our initial triples, and compute how many of them are also generated by *IBminer*. In this way, we make sure that the resulting InfoBox triples were most likely to be mentioned in the text.

To have a better understanding of the issue, we studied the relation between the abstracts and existing InfoBox triples in DBpedia for 50 randomly selected subjects (which have both abstract and InfoBox) from the Musicians domain. Interestingly, out of 1155 unique[2] InfoBox triples, only 305 are covered by their abstracts (i.e. only 24.4%). We should also add that many of the covered cases need extra or common-sense knowledge to be converted to the exact InfoBox format (e.g. different formats for names or dates). More importantly, we have found 47 *wrong triples* (3.8%) and 146 *unimportant triples* (11.7%) (e.g. file names, image names, formats, size, or alignment). Next we will present our results on the Musicians data set.

Best Matches: Part a) in Figure 3.5 depicts the precision/recall diagram for best matches. As we decrease our thresholds on potential match evidence count (τ_e) and confidence (τ_c), we generate more triples with lower recall and precision. To ease our analysis, we study the effect of both threshold in the same experiment by multiplying them. As can be seen, for the first 20% coverage, *IBminer* is generating only correct information. For these cases, τ_e is very high. To reach 97% precision which is more than what DBpedia offers, one should set $\tau_c.\tau_e$ to 6,300 ($\approx .12|T_m|$). For this case as shown in Part c) of the figure, *IBminer* generates around $96.6K$ triples with 37.3%

[2]We had encountered 92 (7.4%) duplicate triples.

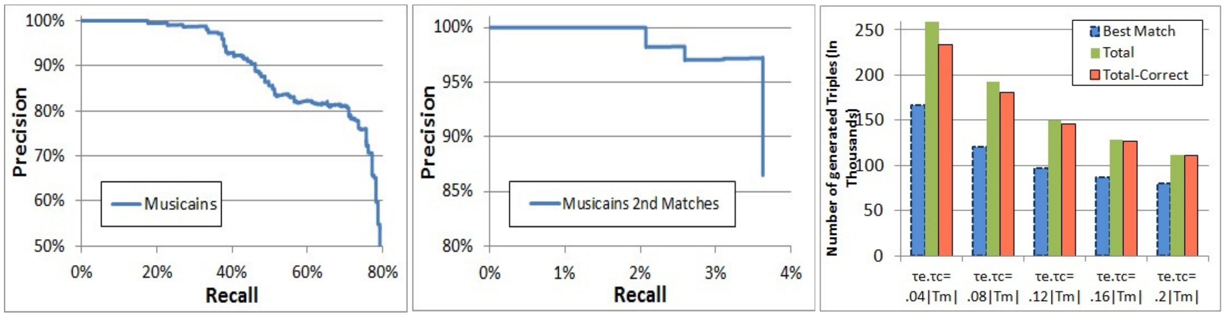


Figure 3.5: Results for Musicians data set: a) Precision/Recall diagram for best matches, b) Precision/Recall diagram for attribute synonyms, and c) the size of generated results for the test data set.

recall.

Secondary matches: We also generate the secondary matches or what we refer to as *attribute synonyms* for all the InfoBox triples generated in the previous step. Precision and recall are computed similarly and depicted in part b) of Figure 3.5 (while the potential attribute synonym evidence count (τ_{se}) decreases from right to left). For instance, to have 97% accuracy one need to set $\tau_{sc}.\tau_{se}$ to 24,000($\approx .9|T_m|$). Although synonym results indicate only 3.6% improvement on the overall recall, the number of correct new triples that they generate are relatively large ($53.6K$ triples).

Part c) of Figure 3.5 summarizes the number of best matching triples, the total number of generated items, and the total number of correct items for various τ_e and τ_{se}. It shows that for τ_e=$.12|T_m|$, we reach up to 97% accuracy while the total number of new InfoBox triples is $146.3K$. If we do not consider duplicate, unimportant, and wrong triples in the original InfoBoxes, then the coverage of the current InfoBoxes is improved by at least 27.6%. This is actually very impressive considering that ***IBminer*** only uses the long abstract of the page.

3.5.3 The Impact of Category Selection

To understand the impact of category selection technique, we repeat the previously mentioned experiments for different levels (L) and category numbers (N_C). Here, we only compare the results based on the recall estimation. First, we fix the N_C at 40 and

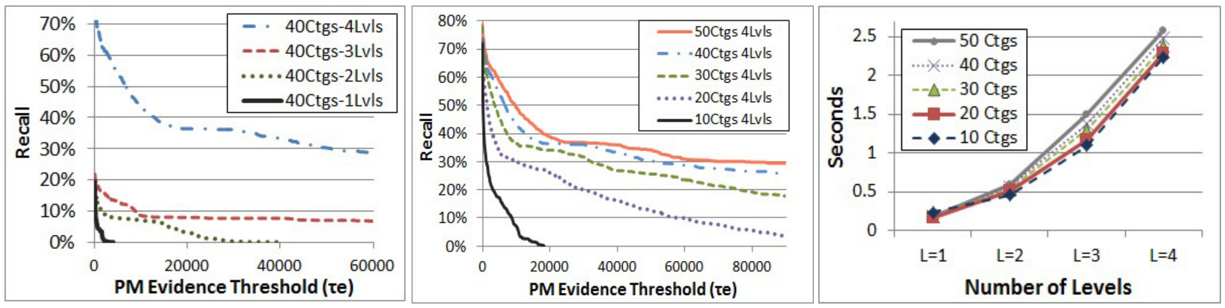


Figure 3.6: a) The impact of increasing level number on Recall, b) The impact of increasing Categories number on Recall, and c) InfoBox generation delay per abstract.

change L from 1 to 4. Part a) of Figure 3.6 depicts the estimated recall (only for the best matches) while *PM*'s frequency threshold (τ_e) increases. Not surprisingly, as L increases, the recall improves significantly, since more general categories are considered in PM construction. On the other hand, using more categories also improves the recall as depicted in Part b) of the figure. In this part, we fix L at 4 and change N_C from 10 to 50. The results indicate that even with 10 categories, we may obtain good recalls for lower frequency thresholds (τ_e). This manly proves the efficiency of our category selection technique, since even with few categories the system is able to reach high coverage.

In Part c) of Figure 3.6, we compare the time performance of all the above cases. This diagram shows the average time spent on generating final InfoBoxes for each abstract. As we increase the levels, the processing time increases exponentially since IBminer performs more database accesses to generate the categories structure. However, we observe very small changes with the increase in N_C, since it does not require any additional database accesses.

3.5.4 Results on New Domains

To study *IBminer*'s performance on different domains, we ran it on two other data sets: i) Actors data set which has similar attributes with the Musicians, and ii) Institutes data set which has completely different set of attributes from the other two. For these ex-

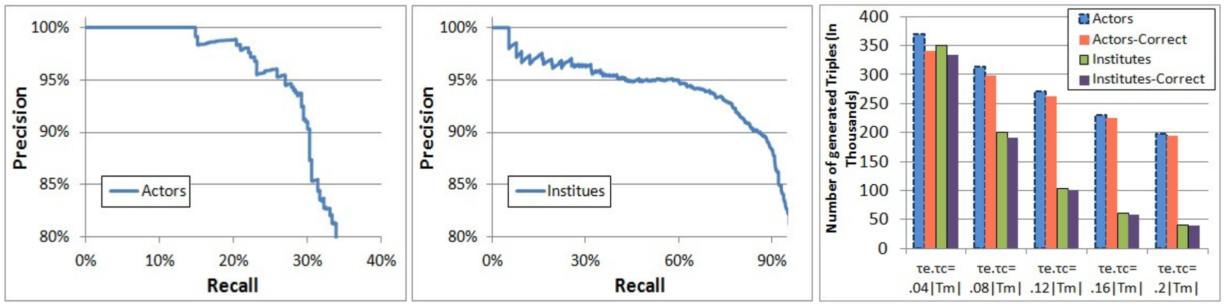


Figure 3.7: a) Precision/Recall diagram for best matches (Actors), b) Precision/Recall diagram for best matches (Institutes), and c) the size of generated results for Actors and Institutes.

periments, we used N_C=50 and L=4. The PM for each case is constructed considering the entire data set. As shown in Table 3.1, the average number of sentences in these two data sets is less than that for Musicians, however the total number of resulted triples are still comparable to that for Musicians data set.

As in the case of Musicians data set, we manually graded triples from the resulted triples for Actors and Institutes for minimum possible thresholds (5000 triples from each). Part a) and b) in Figure 3.7 depict the precision/recall diagrams for Actors and Institutes data sets respectively when $\tau_c.\tau_e$ decreases from left to right. Although we did not create any specific GD rule for these two data sets, their accuracy can still reach high values. This simply shows that our technique is domain-independent.

Part a) of Figure 3.7 depicts the precision in Actors results drops early. The main reason for this outcome is that many triples with same attribute and same value (more than 300 in some cases) in the original InfoBoxes are simply wrong (e.g., values *TV*, *Film*, and *Television* are frequently listed as the *occupation* of a person). A very similar problem exists for the Musicians data set (e.g. *singer* is listed as the *instrument* for a person in more than 300 evidences). Since the the same errors are repeated in several examples, *IBminer* generates some high-confidence wrong triples. Evenso, *IBminer* is still able to generate $270,8K$ results with 97.0% accuracy and 25.6% recall.

On the other hand, the InfoBox triples in the Institutes data set are organized very loosely, with different names used for similar attributes. Thus many more attribute names are used in this data set than the other two, and as a result, for many attributes, *IBminer* is given very few example to learn from,. This directly affects the number of final results (Part b in Figure 3.7). Even so, *IBminer* manages to generate $102.9K$ results for this data set, with 97.0% accuracy and 25.4% recall. Interestingly, if one can tolerate 95% accuracy, the recall for this case can reach up to 56.1% which is quite impressive.

In Part c) of Figure 3.7, we provide the total number of generated results for various evidence and confidence thresholds. Although better tuned thresholds could be used, we kept the same thresholds from Section 3.5.2 (scaled by the size of T_m) and report the total number of generated results and the number of accurately generated ones for each data set in this figure. Although better coverage is reported for the Institutes data set, for accuracy more than 96% much fewer results were generated. This is due to the large number of attribute names and few examples for each of the attributes in the original InfoBoxes in Wikipedia. *IBminer* also generates more results for Actors data set than Musicians data sets, which is mainly due to higher popularity of pages in the Actors data set which contains many contemporary people.

3.5.5 Application-based Evaluation

In previous sections, we showed that the recall of the generated information by IBminer can reach high values. However, to understand how this improvement in recall actually helps the applications of knowledge bases, we carried out an application based experiment. In this experiment, we create a set of popular queries (explained next) and compare the search results obtained after DBpedia is extended by IBminer with the original ones. We also compare these with the results produced by the iPopulator [LBN10] system. Although other similar systems have been proposed recently, we selected iPopulator for two main reasons: i) it is designed for generated information from free text, and more importantly ii) the iPopulator's results are publicly available.

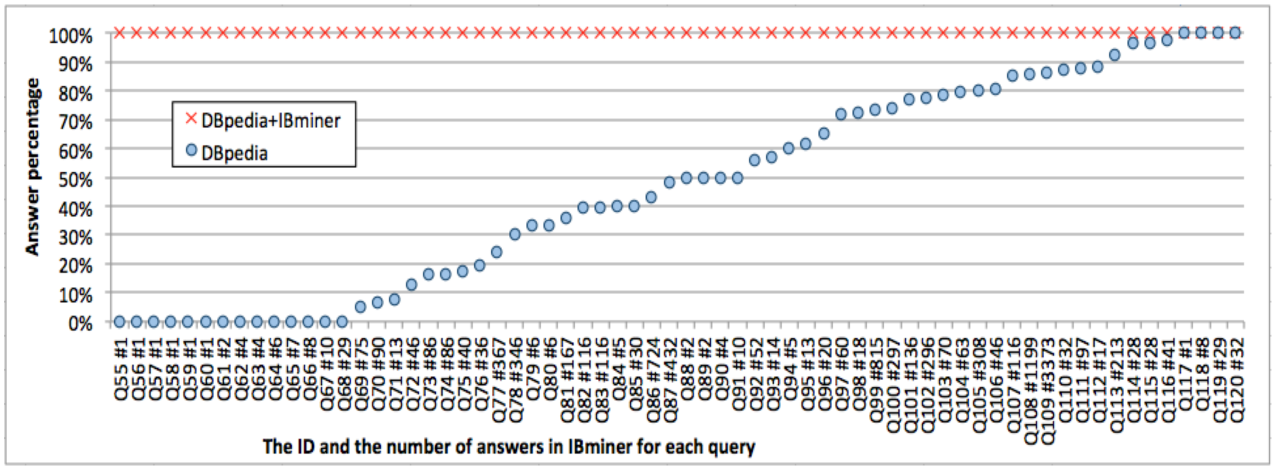


Figure 3.8: Number of results generated for different queries using DBpedia and IBminer knowledge bases.

Creating the query set: In order to create a set of popular queries for our evaluation, we used Google Search Auto-Complete system, and found around 150 keyword queries suggested by this system to complete two phrases: "*musicians who*" and "*actors who*". We were able to translate 120 of these keyword-based queries to SPARQL. The remaining keyword queries, (e.g., "*Actors who are tall*", "*Musicians who married normal people*", etc.) are too vague for a precise translation and quantification and were thus ignored.

Knowledge Bases: Three different knowledge bases (KBs) are used in this evaluation. As for the baseline KB, we use DBpedia's InfoBox triples. Note that both IBminer and iPopulator use DBpedia as their initial knowledge bases. Since the goal is to measure how much IBminer's result improves DBpedia, we combine the triples in DBpedia and IBminer into our second KB called IBminer+DBpedia. We create the third KB similarly for iPopulator by adding its results to those of DBpedia.

After preparing the queries and the knowledge bases, we employed Apache Jena [Jen], and ran the queries using the three knowledge bases. For more than 44% of the queries, no answer is found from any of the knowledge bases. This very clearly proves the incompleteness of the current knowledge bases. Nevertheless, for the remaining queries, the IBminer case almost always (except 4 queries) finds more answers than the baseline case. Figure 3.8 shows what portion of the answers (for each query) that are found using IBminer+DBpedia is also inferable using only DBpedia's knowledge base. To better present the results, we sort the queries based on the percentage introduced above. We also exclude the queries with no answer in any of the knowledge bases from the figure.

As Figure 3.8 indicates, there are several cases (11.6%) in which DBpedia is not able to provide any answer for the queries as opposed to IBminer. Using IBminer, we are actually able to find between 1 to 29 answers for these queries. The number of found results using IBminer+DBpedia is included in the horizontal axis in addition to the queries ID in Figure 3.8. In total for all queries, IBminer improves original DBpedia by 53.3% on answering structured queries. This value for iPopulator is less than 1%. In fact, iPopulator was able to slightly improve DBpedia in query answering only for 6.6% of the queries. This is mainly because of three main reasons: i) the recall of iPopulator is much lower than IBminer (at least 10 times). ii) iPopulator does not recognize synonymous attributes, and iii) some of the generated values in iPopulator are not accurate and in addition to the correct values, they usually contain other unrelated textual phrases as well.

All queries and their answers from the introduced knowledge bases are available at SWIMS website [Sem]. In this website, users can also use our recently proposed demo at PVLDB13 [MGZ13c] and find the list of InfoBox triples for the answers provided for each query. We provide the provenance of each triples, which indicates if the triple is taken from DBpedia, IBminer, etc. Thus verifying the quality of the answers for these queries is quite easy.

3.5.6 Large Scale Experiment:

At the time of writing this dissertation, we are in the process of running *IBminer* on the entire Wikipedia's text. This experiment is distributed over the Hoffman2 cluster at UCLA [Hof], on which we have access up to 400 cores each with 8 GB of main memory. The first three steps of the *IBminer*'s process is now completed which is resulted in more than 2.7 billion semantic links from text, out of which 21.6 million match the existing InfoBox triples. Using these matches, *IBminer* has learnt more than 29.6 million patterns in step c). We have recently started the last step to extract the final structured summaries using the learnt patterns, and the early results indicate that this should at least double the size of the initial KB (i.e. the original DBpedia).

3.6 Related Work

Generating knowledge bases from structured or semi-structured data has been the focus of several works such as FreeBase [BEP08], DBPedia [BLK09], YaGo [SKW08], and YaGo2 [HSB11]. FreeBase is Google knowledge base which is harvested from several source including Wikipedia. DBPedia extracts triples from the structured part of Wikipedia and other data sets (e.g. WordNet [SR98] and GeoNames [GEO]). YaGo2 uses similar techniques and predefined extractors [EG13] to construct knowledge bases mostly from Wikipedia with the main focus on extracting temporal and spatial information. The knowledge base provided by these techniques are usually incomplete and inconsistent since they only rely on only structured data and/or human volunteers.

To address these issues, a large body of research currently focuses on pattern-based knowledge generation from free text. Etzioni et al. in KnowItAll created a general purpose ontology by finding unary and binary predicate instances from the text [ECD04]. Later, Yates et al. proposed a similar system called TextRunner which improves the accuracy of KnowItAll by considering all meaningful relations in the text [YCB07]. Despite the impressive accuracy in their evaluations, they both suffer from limited coverage issues.

In [NMI07], Nguyen et al. used both semantic and syntactic information to extract relations from Wikipedia. However, their approach focuses on the relations between existing entities in Wikipedia, which is not easily applicable on InfoBox generation and population. Other similar systems such as PROSPERA [NTW11] also suffer from the very same problem. Wu and Weld used an automatic technique to create a training set [WW10]. Their main idea, which was originally proposed in [HZW10], is to assign each attribute/value pair in the InfoBoxes to a sentence in the page using some straightforward heuristics. Then, these assignments are used to learn extractors which in turn generate more attribute/values. Although it is shown impressive improvement with respect to TextRunner, their accuracy is still limited to less than 90% for most cases.

NELL [CBK10a] performs a pattern-based fact generation on a large-scale text set and iteratively improves their knowledge base. In 66 days experiment, NELL generated about 230,000 unary relations (Subject/Category triples) and 12,000 binary relations (InfoBox-like triples) with 74% accuracy. Comparing with the size of the input data set which contains 500 million Web pages, the coverage seems too low. Another approach which attempts to generate a large scale commonsense knowledge base is CYC [EG06]. Although, CYC has been recently equipped with several NLP-based and automatic techniques, the main core still relies on supervised techniques on semi-structured data sets.

Another large scale pattern-based knowledge construction system is Probase [WLH11, WLW12], which iteratively induces taxonomical information from large text. At each iteration, Probase uses existing taxonomies to identify new '*isA*' pairs from text and integrates the new pairs into the taxonomies. It also proposes an algorithm to merge and connect concepts. Probase produces a general taxonomy with more than 2.7 million concepts from the textual Web documents. However, Probase is only able to generate taxonomical information which is not the focus of our current system.

Recently, two similar approaches to IBminer are proposed by Lange et al. called iPopulator [LBN10] and Liu et al. called DeepDive [NZR12]. iPopulator presents a new technique which uses existing InfoBox triples in Wikipedia to discover structures in the text, that represent possible attribute/value for the subjects. Unfortunately, iPopulator's coverage is low as our comparison in Section 7.5 showed. DeepDive, on the other hand, first employs NLP tools to extract relations from raw text. Then, the relations are used to train statistical models which convert the initial relations into a knowledge base. Although this approach in nature is very similar to IBminer, it does not provide any evaluation or experiment on the quality of the generated triples.

CHAPTER 4

OntoHarvester: Automatic Ontology Generation from Free Text

By introducing *concepts* and their *relations*, *ontologies* provide a critical information structure that facilitates the processes of sharing, reusing, and analyzing domain knowledge in knowledge-based systems [Gru93]. Ontologies are one of the most important components of the Semantic Web, and many knowledge-based applications, including semantic search, intelligent information integration, and natural language processing, depend on them. This has given rise to ambitious systems, such as FreeBase [BEP08], DBPedia [BLK09], YaGo2 [HSB11], and ProBase [WLW12], that support general ontologies alongside their large-scale knowledge bases. However, in spite of their size and breadth, the ontologies they provide are not comprehensive enough for many domain-specific applications. Therefore, more complete ontologies must be built for a roster of important applications, and cost-effective approaches are needed for this challenging task. Unfortunately, many of the existing ontology generators use manual or highly supervised techniques [Bou92, Vou95, PL01, Dro03, Sno06, WW08]. These approaches make it easier to incorporate knowledge from domain-experts, but their high demands on time and resources impair their practical scalability. To address this problem, automatic or semi-automatic approaches have been proposed, using statistical techniques, occasionally combined with shallow NLP-based methods [MS00, QHF04, BCS07, PD10, DZP10].

These automatic approaches have achieved a fair amount of success, but have also encountered several limitations. In fact, as stated in Poon et al. [PD10], none of the existing techniques have achieved a higher accuracy than 91%. Moreover to reach these relatively high levels they require large training sets highly focused on the domains of interest. A simple analysis of these limitations suggests that many follow from their inability to take full advantage of the linguistics morphologies in the text, whereby they cannot handle well sentence ambiguities and linguistic exceptions. This observation has inspired the design of our *OntoHarvester* system, that uses deep NLP analysis through the SemScape framework to automatically generate domain-specific ontologies from unstructured text. In this section, we describe this approach, and show that it outperforms previous approaches in terms of accuracy and coverage.

OntoHarvester starts with an initial ontology (*Seed*) and iteratively extends it with new terms carefully mined from the input text. At each iteration, OntoHarvester only accepts new terms that are semantically connected to the current concepts. In this way, OntoHarvester remains focused on the specified domains and achieves high resistance to noise. On the other hand, since the semantic connections are mined through a deep NLP-based technique, the system is able to generate very comprehensive ontologies from small corpora. This represents a major improvement over other automatic techniques, which require large corpora to generate domain-specific ontologies, as shown in detailed comparisons presented later in the section.

The main tasks performed by OntoHarverster can be summarized as follows:

1. **Building TextGraphs:** This task is performed using our SemScape text-mining system described earlier in Chapter 2 *SemScape* [MKI11a, MKI13b]. Using SemScape, OntoHarvester captures *candidate terms* and their grammatical links in the text, which are represented as weighted hyper-graphs (*TextGraphs*).

2. **Extracting Semantic Relations:** OntoHarvester uses graph patterns (*Graph Domain* or *GD* Rules) similar to those of Hearst's [Hea92] to extract semantic relations (initially *part_of* and *type_of* relations).

3. **Extracting Concepts:** These semantic relations are used to detect candidate

terms that are strongly connected to the current concepts. These terms are then added to the ontology.

4. **Finding New Relation Types:** In this step, OntoHarvester finds new semantic relation types between currently accepted concepts. These new relation types will help OntoHarvester find more related concepts in the next iterations.

The presentation of this chapter is thus organized as follows:

- The next section introduces a running example used in the rest of the chapter, and then describes the derivation of TextGraphs from input documents.
- In Section 4.2, we introduce OntoHarvester and show how new terms are recognized and incorporated into the ontology on the basis of (i) their frequency and correctness confidence and (ii) the strength of their connections with existing concepts (An important difference from most of the existing works that only use (i)).
- Section 4.2.1 focuses on how OntoHarvester uses Graph-based patterns (*Graph Domain* or *GD* rules), to extract more information with fewer patterns. This represents a significant improvement over existing pattern-based techniques that rely on tree-based patterns or regular expressions.
- Section 4.2.4 describes the technique used by OntoHarvester to automatically suggest new possible domain-specific semantic relations to improve the final Ontology. This is another feature differentiating OntoHarvester from existing works which mostly use predefined relation types.
- Section 4.3 reports on the results of our experiments. We used an application-focused evaluation technique to evaluate our system. To this end, we implemented a “topic identifier” system that mainly uses an ontology to suggest topics for a given piece of text. The results indicate that the ontology created by OntoHarvester can significantly improve the performance of applications using the

ontology. Moreover, extensive experiments to measure precision and recall on several application domains also show significant uniform improvements over previous approaches.

4.1 From Text to TextGraphs

To ease our discussions in this section and to better understand the OntoHarvester's tasks, we continue with the following motivating example:

Running Example: "*An algebraic equation, such as a linear or nonlinear equation, is an expression that contains variables and a finite number of algebraic operations.*"

Using *SemScape*, as described in previous section, OntoHarvester converts all sentences in the text into *TextGraph*s. One such TextGraph for the above example is depicted in Figure 4.1. This TextGraph contains all possible single- or multi-word terms (sometimes called *name entities* in the literature) which np may be representing. E.g. the noun MPs for "*linear and nonlinear equation*" are "*equation*", "*linear equation*", and "*nonlinear equation*"[1]. These MPs are also referred to as *candidate terms* in this section. The TextGraph also include the grammatical links such as <*equation*, *subject_of*, *is*>, <*algebraic equation*, *subject_of*, *is*>, <*linear*, *property_of*, *equation*>, <*expression*, *object_of*, *is*>, etc.

A very important feature of this framework is the ability to adopt an ontology and provide more related *candidate terms* with respect to the specified ontology. To understand why this is important, we should note that for complex noun phrases (NPs) there might be several possible candidate terms. Not all of these candidate terms are useful or meaningful. Therefore, suggesting all of them as MPs will lead to a very large set of candidate terms which lowers the efficiency of the applications. To prevent this prob-

[1]The fourth term in the example is "*linear and nonlinear equation*", which is referred to as Whole Part (WP).

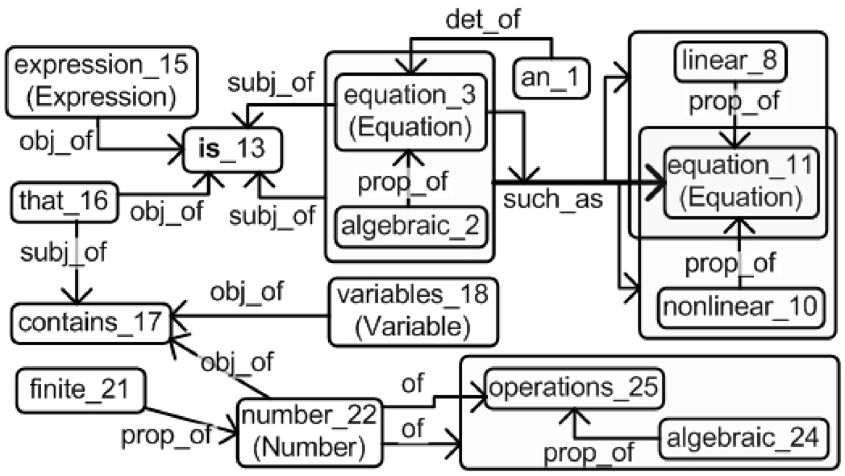


Figure 4.1: Part of the TextGraph for our motivating example.

lem, if the system is fed with an ontology, say O, we only generate candidates terms that either i) contain less than three words, ii) are part of an existing concepts in O, or iii) contain a concept from O. In many cases this generates all possible candidate terms; however for many long noun phrases, this helps reduce the size of the TextGraphs.

As shown in Figure 4.1, all the candidate terms which are also a concept in O are tagged with their concepts in the TextGraphs. For instance, nodes *"expression"*, *"equation"*, *"number"*, and *"variables"* are in O and tagged with their corresponding concepts. Notice that the corresponding concept of a term may be a synonym or alias (e.g. *"variables"* is an alias for *"variable"*).

Algorithm 1 summarizes OntoHarvester's main steps, where the tasks described in this section correspond to steps 2 and 3. These steps are followed by the ontology generation steps described next.

4.2 Ontology Generation

Once the TextGraphs (TGs) are created for the given corpus (τ) as explained in the previous section, OntoHarvester starts its main discovery loop by iterating over the following three main steps (shown as steps 7, 8 and 9 in Algorithm 1).

Algorithm 1 OntoHarvester()

```
1: OntoHarvester ($\tau$, $O_0$) {
2:     $SemScape$ = **new** SEMSCAPE();
3:     $TGs$ = $SemScape$.generateTGs($\tau$, $O$);
4:     $O$ = $O_0$;
5:     $R_T$ = [$type\_of$, $part\_of$, $rename$];
6:     **while** (**true**) {
7:         $rels_o$ = extractOntologicalRelations($TGs$, $O$);
8:         $concepts$ = extractOntoConcepts($rels_o$);
9:         $aliases$ = extractOntoAliases($rels_o$);
10:        $O$ = $O$.add($rels_o$, $concepts$, $aliases$);
11:        **if** (size($concepts$)+size($aliases$) $\leq min_c$)
12:            **return** $O$;
13:        $newR_T$ = extractNewRelationTypes();
14:        $R_T$ = $R_T$.add($newR_T$);
15:        $TGs$ = $SemScape$.reGenerateTGs($\tau$, $O$);
16:    }
17: }
```

- *Extracting Relations*: OntoHarvester extracts ontological relations between the existing terms in TGs at the beginning of each iteration. For instance, from the TextGraph of our running example, OntoHarvester is able to generate relations such as <*algebraic equation*, *type_of*, *expression*>, <*linear equation*, *type_of*, *equation*>, <*variables*, *part_of*, *expression*>, etc. To generate such relations, OntoHarvester uses a set of graph patterns on TGs as explained in Section 4.2.1.

- *Extracting Concepts*: Next, OntoHarvester detects new concepts and aliases using those extracted relations that connect a concept in the current ontology (O) to a non-concept term in TGs. In our running example, considering the above mentioned relations and the fact that "*equations*" is already a concept, OntoHarvester accepts "*linear equation*" (and similarly "*nonlinear equation*" and "*algebraic equation*") as new concepts. Several additional heuristics are employed in this step, which are described in Sections 4.2.2 and 4.2.3.

- *Suggesting New Relation Types*: Before starting the next level, OntoHarvester extract new relation types using *semantic relations* provided in TextGraphs. OntoHarvester initially starts with three types of ontological relations: $type\_of$, $part\_of$, and $rename$. Later at the end of each iteration, it finds new ontological relation types by considering frequently observed semantic relations between current concepts in O. Section 4.2.4 discusses this part in greater detail.

The key idea in above steps is to extend the current ontology (O) only with terms highly related (connected) to O. In this way, as opposed to similar systems [WLB12], OntoHarvester is able to stay focused and generate highly related concepts (to the specified domain) from a noisy corpus. The domain is essentially specified by some of its concept in the initial ontology, O_0. We should note that our experiments in Section 4.3 indicate that O_0 does not need to be large at all. Usually a handful of initial concepts and a small corpus suffice for OntoHarvester to generate a well-focused ontology that describes terms, concepts and relationships learned from the corpus.

In theory, OntoHarvester stops its iterations once no new concept is found. However, iterations at the end of the tail normally produce very few new concepts. Thus, in practice, we can stop the iterations either i) when a certain number of iterations are performed or ii) when the number of newly generated concepts are less than a predefined threshold (min_c).

4.2.1 Extracting Ontological Relations

To generate ontological relations between terms in the TextGraph of each sentence in τ, we use graph domain (GD) patterns/rules described next. Using a syntax similar to SPARQL [SPA08], GD rules are used to specify and detect patterns in TextGraphs which indicate particular fragments of knowledge. In our case, we are interested in using GD rules in order to find taxonomic relations, namely *type_of* (*IS_A*) and *part_of* (*HAS_A*). For instance, consider the relation <*algebraic equation*, *type_of*, *expression*> in our running example. To extract such a link from our motivating example, the following GD rule is used:

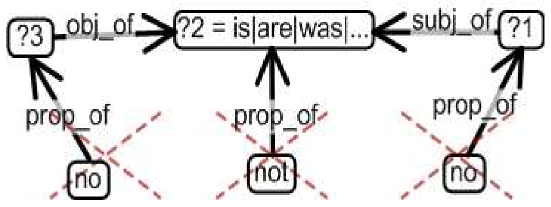


Figure 4.2: Pattern Graph for Rule 1.

——————————— Rule 1. ———————————

SELECT (?1 “type_of” ?3)
WHERE {
    ?1 “subj_of” ?2.
    ?3 “obj_of” ?2.
    NOT(“not” “prop_of” ?2).
    NOT(“no” “prop_of” ?1).
    NOT(“no” “prop_of” ?3).
    FILTER (regex(?2, “^is^|^are^|^was^|^were^|^be ...”, “i”))
}

As shown in Figure 4.2, Rule 1 specifies a pattern in which two nodes (labeled *?1* and *?3*) are connected through a third node (*?2*) that represents any tense of the verb “to be”. The NOT parts filter out the negative sentences; This is usually a challenging task in most current systems. Rule 1 catches the following four results from the TextGraph in Figure 4.1:

- *<equation, type_of, expression>*,
- *<algebraic equation, type_of, expression>*,
- *<equation, type_of, that>*, and
- *<algebraic equation, type_of, that>*.

For the last two relations, OntoHarvester uses SemScape’s *Anaphora Resolution* component to resolve the pronoun “*that*” and replace it with its resolved term (for this case “*expression*”). If no resolution is found, the last two relations will be rejected, since the term “*that*” is in our black list of concepts, alongside all other pronouns. For the above case, pronoun resolution results in re-generation of some of the exiting relations.

However, Rule 2 shows how pronoun resolution can also generate new relations which most existing works are not able to capture.

```
———————————— Rule 2. ————————————
SELECT ( ?1 “part_of” ?3 )
WHERE {
    ?3 “subj_of” ?2.
    ?1 “obj_of” ?2.
    NOT(“not” “prop_of” ?2).
    NOT(“no” “prop_of” ?1).
    NOT(“no” “prop_of” ?3).
    FILTER (regex(?2, “ˆinclude*|ˆcontain*|ˆconsist*|...”, “i”))
}
——————————————————————————————
```

This rule has a very similar structure to Rule 1, but here it defines a pattern for subjects and objects connected with any verbs indicating inclusion (e.g. *include*, *contain*, and *consist*). For our motivating example this rule generates relations such as:

- *<variables, part_of, that>*
- *<finite number of algebraic operations, part_of, that>*

Now, by resolving “*that*” to “*expression*”, OntoHarvester generates new relations such as:

- *<variables, part_of, expression>*
- *<finite number of algebraic operations, part_of, expression>*

In OntoHarvester, we have generated 48 GD rules for taxonomic relation extraction [MKI13b], and our experiments in Section 4.3 show that these powerful rules can be used (with no changes) in different domains. Out of these 48 rules, 30 are for *type_of* relations and the rest are for *part_of*.

Combining the Relations: Each of the extracted relations from the GD rules introduced earlier are assigned with a correctness confidence. The confidence value of a relation generated from pattern p is set to the minimum confidence among the edges in the matching graph for patten p. After applying the GD rules over all TextGraphs, we combine the generated ontological relations from different TextGraphs. The combination process of relations confidence is very similar to that for TD rules; Since we need to keep the confidence values below 1, OntoHarvester uses the formula C=C_1+(1-$C_1)\times C_2$ to combine the confidence of two relations having C_1 and C_2 as their respective confidence values. It is easy to see that $1 \geq C \geq C_1$ and $1 \geq C \geq C_2$.

That is, each time we encounter new evidence for an existing relation, we increase its confidence in proportion to the new relation's confidence. Moreover, for each unique relation, we store its frequency, which measures the number of times this relation is extracted from different sentences.

Relations between concepts in O whose confidence and frequency exceed confidence threshold ($Conf$) and frequency threshold ($Freq$) are added into ontology O. Therefore, we will not include relations with high confidence but low frequency, inasmuch as insufficient evidence exists for them. Symmetrically, relations with high frequency and low confidence will also not be accepted.

4.2.2 Extracting Concepts

As mentioned previously, nodes in TextGraphs can be concepts, non-concept (or candidate) terms, and other words in the text such as verbs, articles, etc. Thus we need to ascertain which of the current candidate terms are actually new concepts. Most previous works rely on statistical or frequency-based techniques to accept candidate terms as concepts, and require large corpora to perform satisfactory. OntoHarvester instead exploits the ontological relations generated in the previous section to detect new concepts. The main intuition in OntoHarvester is that if a candidate term is a concept in the domain of τ, it should be connected to some other concepts of the domain through one or more relations. Thus, OntoHarvester accepts a term as a new concept if it has a "*strong connection*" to other existing concepts in O. Here, by a *strong connection*, we mean that the relations from the new term to existing concepts have frequency and confidence values that are respectively greater than $Freq$ and $Conf$.

With this intuition, all ontological relations connecting any concept in O to a candidate term, say CT_i, are retained and their confidence and frequency are combined exactly as described in the previous section. Now, CT_i's with high frequency and confidence are accepted as new concepts, providing that they do not contain: 1) a word from the black list (e.g. same, an, etc.), 2) an attributive adjective (e.g. short, similar, etc.), 3) a comparatives or superlative adjective, 4) a verb, an adverb, or a pronoun, and 5) a numeric or a symbol. After these adjustments, we accept those candidate terms whose confidence and frequency are larger than our pre-specified thresholds, and add them to O.

4.2.3 Extracting New Aliases

A single concept may have several "names". Research shows that people use different names for the same concept more than 80% of the time [FLG87]. Abbreviations, variations of the term (e.g. plural form), acronyms, aliases, etc. may be used to address the concepts. To find these different names for same concepts, OntoHarvester uses the following heuristics:

- We directly use either TD or GD rules to find aliases explicitly mentioned in the text. (e.g. *"a node may be called a vertex."*). This technique considers aliases as a new type of ontological relations as in Section 4.2.1. Currently we have generated 29 TD rules and seven GD rules for finding such relations respectively in annotated parse trees and TextGraphs.
- If two concepts are "*strongly*" connected only through *type_of* in both directions, they will be considered aliases of each other. (e.g. *<edge, type_of, side>* and *<side, type_of, edge>* means that "*side*" and "*edge*" are aliases.)
- We use synonym information from WordNet and Wikipedia Redirect pages to

detect aliases between concepts in the current ontology.

- The plural form of a concept is considered an alias for its singular form (e.g. *sides* and *side*).

Finally, adjustments similar to those discussed in the previous section will be applied to pick the final aliases. We group all the aliases of the same concept and (randomly) consider one of them as the head of the group. In the next iteration of OntoHarvester, at the concept annotation time, we use the head of each group for tagging any of the aliases in that group.

4.2.4 Extracting New Relation Types

As discussed in Section 3.2, our system is also able to generate general semantic links between the terms in the TGs. The most prominent example of such semantic links is the <*subject*, *verb*, *object*> link, in which the *verb* in a sentence is connecting its *subject*s to its *object*s. Using these links, OntoHarvester attempts to find new ontological relation types between exiting concepts. These new relation types will be used in the next iteration of the OntoHarvester to find more related concepts.

Let R_T be the set of current ontological relation types. R_T initially contains $type\_of$, $part\_of$, and *rename* relation types. The average number of relations in O for these relation types is called avg_{RT}. Also, let f_{sl} be the number of time the semantic link sl is generated between concepts in the current ontology O. Thus, for each sl, if f_{sl} is greater than $avg_c \times avg_{RT}$, OntoHarvester adds sl to the the current relation types (R_T). Where, avg_c is a constant factor for tuning the number of newly found relation types. Our experiments show that if $avg_c > 1$, OntoHarvester will find at most three new relation types (which are "*of*", "*be*", and "*and*"). It is also important to mention that the newly found relation types may not always resemble any specific ontological relation. Nevertheless, they are very useful for connecting concepts of the same domain. That is, they will help significantly in discovering new concepts for the specified domain.

4.3 Experimental Results

We evaluated the effectiveness of OntoHarverster by using both manual and automatic techniques on four data sets from different applications domains. All data sets are available for online access[2].

All the experiments were performed on a single machine with 16 cores of 2.27GHz and 16GB of main memory running an Ubuntu12. On average, our system spends 3.07 seconds for parsing each sentence on a singe CPU. Therefore, using 16 cores, we were able to process and generate *initial triples* for 5.2 sentences per second.

4.3.1 Data Sets and Initial Ontologies

The data sets we use in our evaluation experiments are as follows:

- **SFF**: The first data set is actually Chapter 5 of the book entitled *"Naval Shiphandler's Guide"*. This data set, referred to as SFF (for Ship Fire Fighting), contains around 2,200 paragraphs and 5,000 sentences.
- **PGT**: The second data set is the PGT data set that was also used in [JT10]. It contains the reports on "Patterns of Global Terrorism" for years 1991-2002[3] for a total of about 3,000 paragraphs and 8,000 sentences.
- **MATH**: The third data set is much larger than the other two and contains 24,184 long abstracts ($\approx$172K sentences) of mathematics related pages in Wikipedia. To generate this data set, referred to as MATH, we collected each page in Wikipedia where one, or more, of its ancestor categories belongs to the *"mathematics"* category. To control the size of the the data sets, we searched only up to 3 levels of ancestors.
- **ANIM**: In a similar way, we have created the forth data set, called ANIM. This

[2]http://semscape.cs.ucla.edu

[3]Downloaded from the Federation of American Scientists: http://www.fas.org/irp/threat/terror.htm

data set contains 11966 long abstracts ($\approx$71K sentences) for pages having ancestor category "*animal*".

We should add that the GD patterns for relations extraction are created only once and independently from the above data sets.

The last two data sets present challenging test cases for several reasons, including the fact that they are very noisy. Take for instance MATH, a domain that is more rigorous and well-defined than most. A careful analyses of this data set shows that less than 45% of those pages describe concepts directly related to mathematics. In fact, several pages in this data set actually belong to related topics, such as physics, computer science, astronomy, book titles, universities, people names, and so on.

For each of these data sets, we generated one or two initial ontologies (seeds). The two large seeds for MATH and SFF were manually created as part of other projects, in which the goal was to help designing of instructional materials, evaluation of assessment performance, and the modeling of knowledge, skills, and attitudes[4]. The seeds used in the rest of this section are as follows:

- $O_{\texttt{Anim}}$: The seed contains: *Animal(s)*, *Mammal*, *Fish*, *Bird*, and *Insect*.
- $O_{\texttt{sff\_short}}$: The seed contains: *AFFF (an acronym for Aqueous Film Forming Foam)*, *Extinguisher*, *Fire(s)*, *Firefighting*, *Firemain*, *Hose*, *Nozzle*, *Party*, *Smoke*, *Ventilation*, and *Water*.
- $O_{\texttt{sff\_long}}$: The seed contains 150 concepts and 20 aliases. The index of the *"Naval Shiphandler's Guide"* book is used to create this ontology.
- $O_{\texttt{pgt}}$: The seed contains: *Attack(s)*, *Bombing(s)*, *Hostage(s)*, *Incident(s)*, *Target(s)*, *Terrorism*, *Terrorist(s)*, *Terrorist group*, and *Threat(s)*.
- $O_{\texttt{Math}}$: The seed contains 877 concepts and 186 aliases which are extracted from Common Core State Standards for Mathematics.

[4]For more information visit: http://www.cse.ucla.edu/.

Table 4.1: OntoHarvester (OH) vs. CRCTOL on PGT data set. ($Prob$=.98)

	Freq	**Total**	**Precision**	**Recall**	**F-Score**
Concepts-OH	4	1307	92.9%	41.0%	56.9%
Concepts-OH	5	1069	93.0%	35.5%	51.4%
Concepts-OH	6	925	93.1%	23.0%	36.9%
Concepts-CRCTOL	N/A	N/A	92.8%	4.1%	7.8%
Relations-OH	4	2587	83.8%	25.0%	38.5%

4.3.2 OntoHarvester vs. CRCTOL

In this section, we compare OntoHarvester's performance with CRCTOL [JT10]. To the best of our knowledge, CRCTOL is one of the few works which uses a deep NLP-based approach to generate high-quality domain-specific ontologies. CRCTOL provides a natural test bed for comparison experiments, since it obtains the most accurate results among previous approaches and has both precision and recall computed on a publicly available data set (PGT). In order to compare OntoHarvester's performance with CRCTOL, we ran OntoHarvester for 10 iterations starting from the $O_{\texttt{pgt}}$ seed. The results for three different values for the $Freq$ threshold are depicted in Table 4.1.

To measure the precision of our results, we manually graded 10% of the generated concepts and relations. As for recall, we manually found 200 concepts and 100 relations related to the global terrorism domain from the PGT documents for the year 1991. Then, we checked these concepts and relations against the ones OntoHarvester has generated to compute the coverage[5]. Table 4.1 depicts the comparative results for the extracted concepts of these two approaches. The recall and F-Score of OntoHarvester is significantly higher than those for CRCTOL, while the precision is slightly improved. The main reason for this significant improvement is that, as opposed to CRCTOL and many similar techniques, OntoHarvester uses ontological relations to find more related concepts to the application domain, and gradually extend the ontology. In this way, many wrong or unrelated concepts, which are frequently mentioned in the text, are eliminated since they are not semantically connected to the current Ontology. On the

[5]Similar annotation is done for CRCTOL, but our efforts to contact the authors failed.

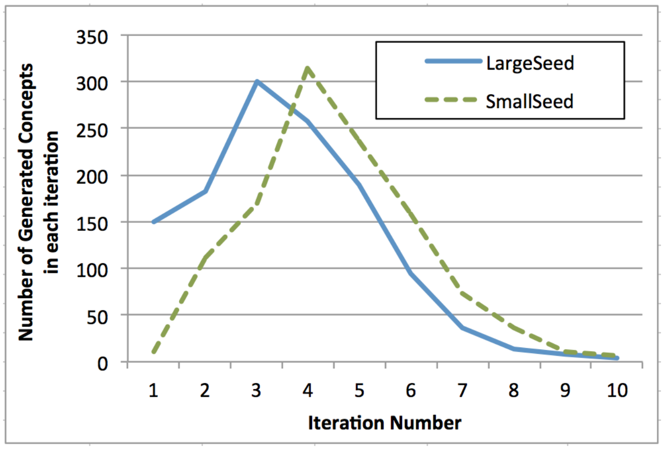


Figure 4.3: The number of generated concepts from SFF at each iteration for small and large seeds. ($Freq$=4, $Conf$=.98)

Table 4.2: The impact of the seed size for SFF.

	Seed	**Total No.**	**Precision**
Concepts	$O_{\mathtt{sff\_small}}$	1131	90.0%
Concepts	$O_{\mathtt{sff\_large}}$	1238	90.9%
Aliases	$O_{\mathtt{sff\_small}}$	187	91.5%
Aliases	$O_{\mathtt{sff\_large}}$	216	94.9%
Relations	$O_{\mathtt{sff\_small}}$	2515	76.1%
Relations	$O_{\mathtt{sff\_large}}$	2770	76.4%

other hand, CRCTOL learns the concepts independently from the relations in one single phase. Thus to avoid wrong concept and improve its accuracy, CRCTOL needs to increase the frequency threshold which consequently reduces the number of generated results.

Since no result is reported for taxonomic relations generated in CRCTOL for this data set, we cannot quantitatively compare the systems in terms of relations coverage. However, similar improvements can be reasonably expected to hold for the quality of the extracted relations, since a big portion of the concepts and thus all relations having those concepts as endpoint are missing in CRCTOL.

4.3.3 The Impact of Seed Size

To study the impact of seed's size on the extracted results, we ran OntoHarvester for SFF with the large and small seeds introduced earlier ($O_{\texttt{sff\_large}}$ and $O_{\texttt{sff\_small}}$ respectively). The results of this experiments are summarize in Table 4.2. We used the same thresholds as in the previous section. Although the smaller seed contains only 10 concepts, rather than 150, starting from this much smaller seed, we were able to generate 91.3% of concepts one can generate by starting from the larger seed. Similar results held for the number of extracted links and aliases. Moreover as shown in Table 4.2, the accuracy of the results was not significantly affected. This verifies that users do not need to spend a lot of time creating the seed: A few domain-specific concepts and a small corpus are sufficient for OntoHarvester to provide high-quality ontologies.

To understand what exactly happens for the case with the smaller seed, we report the number of concepts extracted in each iteration for the run with the larger seed ("LargeSeed") and the run with the smaller seed ("SmallSeed") in Figure 4.3. As expected for the first few iterations, the SmallSeed case generates fewer concepts than LargeSeed case. However after few iterations, the initial ontology is already expanded, many general concepts are found, and OntoHarvester starts to find more concepts than the LargeSeed case. In simpler words, the SmallSeed case catches up with the LargeSeed case after few (less than 10 for SFF) iterations.

4.3.4 OntoHarvester on Various Domains

We ran OntoHarvester for our four data sets and the precision and recall results are summarized in Table 4.3. For all the experiments, we ran 10 iterations and the confidence probability threshold ($Conf$) is set to .98. As for the frequency threshold, we used $Freq$=4 for SFF and PGT, $Freq$=5 for ANIM, and $Freq$=6 for MATH to be proportional to the size of data sets.

Table 4.3 includes the precision of the results obtained for the various test sets.

Table 4.3: The precision/recall results of running OntoHarvester on data sets from different domains.

Data Set	Total	Precision	Recall	F-Score
Concepts-SFF-SmallSeed	1,131	90.0%	-	-
Concepts-SFF-LargeSeed	1,238	90.9%	-	-
Concepts-PGT	1,307	92.9%	41.0%	56.9%
Concepts-MATH	19,476	83.6%	51.7%	63.8%
Concepts-ANIM	6,595	82.3%	-	-
Relations-SFF-SmallSeed	2,515	76.1%	-	-
Relations-SFF-LargeSeed	2,770	76.4%	-	-
Relations-PGT	2,587	83.8%	25.0%	38.5%
Relations-MATH	26,814	87.4%	-	-
Relations-ANIM	2,987	81.2%	-	-

These results were derived by grading randomly selected samples with the help of four human graders. The samples' size was always 500 items or more. As shown in *Precision* column, OntoHarvester provides high-quality results for all four different domains. The provided accuracy for both concepts and relations in these cases is quite steady. This in fact verifies that the OntoHarvester method is general and domain-independent, and can therefore generate high-quality ontologies for a wide range of different domains.

It is interesting to note the effect of noise in data sets on the precision of the generated concepts. The MATH and ANIM that have more than 55% unrelated content are the ones with the lower precision. On the other end, PGT has the largest precision, since not only it is very well-focused but it also mentions repeatedly the same concepts and relations in different scenarios. The latter differentiates PGT from SFF data set. As for the relations, our experiments indicate that the quality of the created relations depends mostly on the complexity of the sentences in the text. For instance, since the sentences in the SFF data set are more complex than sentences in the other three sets (SFF is written by a professional author as opposed other data sets), the extracted relations for SFF are of slightly lower accuracy.

We also evaluated the recall of the MATH experiment in a similar way to that for PGT. This time, we randomly graded 5% of Wikipedia titles (1219 titles) in the MATH

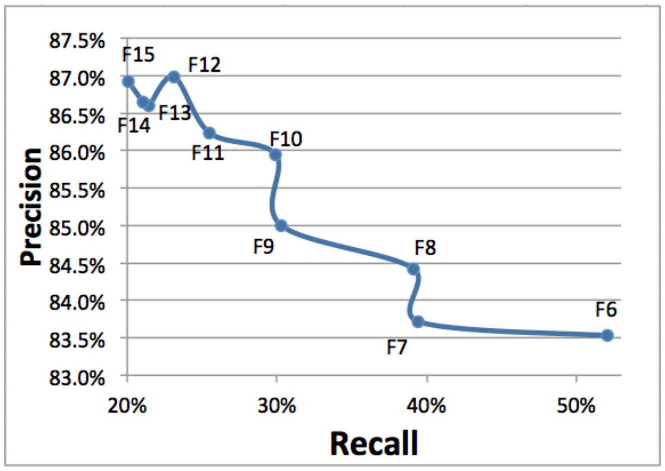


Figure 4.4: The precision/recall comparison for the concepts generated for MATH. ($Freq$= $6, ..., 15$, $Conf$=.99)

data set and found 545 mathematic concepts among them. Then, we used these 545 concepts as a benchmark to compute the coverage of the results generated by OntoHarvester. As reported in Table 4.3, 51.7% of the benchmark's concepts are extracted by OntoHarvester as well. Noting that more than 39% of the benchmark concepts are mentioned only once in the entire data set[6] and 9.5% of them have only one sentence as their abstracts, the estimated recall is actually quite satisfactory—statistical techniques are much less likely to find such low frequent concepts. Moreover, several abstracts include formulas, symbols, or very formalized definitions that pose as much a challenge for an NLP system as they do for a non-expert reader.

To study the effect of the frequency threshold ($Freq$) on the performance of OntoHarvester, we provided the precision and recall for the MATH experiments with frequency thresholds varying from 6 to 15. The results of this experiment are shown in Figure 4.4. Although, we can reach higher precision by increasing $Freq$, the recall drops quickly to less than its half. This is mainly due to the fact that many of the correctly extracted concepts are actually of very low frequency, and they are thus eliminated once $Freq$ is raised.

Finally, in Table 4.4, we provide the most frequent domain specific relation types

[6]In some cases, the concept name (i.e., the Wikipedia page's title) is not even mentioned in its own abstract.

Table 4.4: Domain specific relation types for each dataset.

Dataset	Domain-Specific Relation types (ordered by frequency)
SFF	*explains, consistOf, include, beInvestigationOf, beInstalledIn, beProvidedBy, contain, beProvidedFor, provide, beLikelyIn*
PGT	*kill, beClaimedFor, arrest, beProvidedTo, relatedTo, beConductedBy, conduct, engagedIn, claim, sentencedTo*
MATH	*explains, beCalled, include, beUsed By, use, beUsedIn, beGivenBy, beDescribedBy, beRepresentedBy, knownAs*
ANIM	*include, explains, like, consistOf, contain, beFoundIn, referTo, liveIn, beCalled, beCauseBy, beSpeciesOf*

suggested by OntoHarvester for each data sets. We excluded the generic relation types such as prepositions (*of, in, for, to, with, etc.*), conjunctions (*and* and *or*), and common verbs (*be* and *have*) from this table.

4.3.5 Application-Based Evaluation

Although previous experiments provide a good insight on the quality of the generated results, they are still subjective to the graders opinion. A more objective approach for evaluating ontologies is to use them in an application and evaluate the results of the applications. To this end, we apply the ontology generated by OntoHarvester to the problem of "Topic Identification" (TID) [SC08]. The goal in this problem is to identify the main topic(s) for a given text. Thus, we use the following algorithm, called TID, which is based on the technique proposed by Janik et al. [JK08]:

1. **Constructing Initial Semantic Graph:** For the given text τ, we create all the semantic links in τ as explained in Section 3.2. Using these semantic links, we then construct a semantic graph called G.

2. **Pruning the Graph by Ontology:** Based on the given nodes in an ontology, say O, we only keep nodes in G that are also in O. Each node, say n, is also assigned a weight (w_n) which is initiated by the degree (the number of connected links) of the node in G. Later we use these weights to suggest final topics.

3. **Populating the Graph by Taxonomy:** We add taxonomical links to the remaining nodes (concepts) in G. This way more general terms and topics are added to the graph. Let us say taxonomical link l=<n, $type\_of$, m> is used to extend the concept n in G. Thus, we add link l and node m (if it is not already there) to G and increment w_m by $c_d \times w_n$, where $c_d < 1$ is a constant decaying factor. For instance, if node "*Persian Cat*" is in the list, and O indicates "*Persian Cat*" is type of "*Cat*", we will add "*Cat*" to G as well. However, the weight of "*Cat*" is incremented only by a portion of the weight of "*Persian Cat*". This is because we do not want to give too much credit to more general topics so the final selected topics are specific enough for given text. Of course, if the text (τ) is mentioning other types of cats (e.g. Ragdoll, Munchkin, etc.) then the node "*Cat*" will receive more credit (weight) and its chance to be selected as the main topic increases. We should note that this step is repeated for any newly added nodes to G.

4. **Suggesting Final Topics:** Based on the nodes weights, we rank and report the final topics. (E.g. the node with highest weight is reported as the first-rank topic, etc.)

We employ the above algorithm to suggest topics for a benchmark data set (explained next) using our ontology and a baseline ontology. In this way, we can compare the two ontologies, by evaluating the two sets of results generated by the above algorithm for the benchmark data set using each ontology.

Baseline Taxonomy: To generate the baseline ontology, we start by 684 animal names provided by Kozareva et al. [KH10]. We refer to this list as the *initial animals list*. Next, we retrieve all the hypernyms located on the paths connecting these titles to the *animal* node in WordNet. This Ontology/Taxonomy (called *WNTax*) contains respectively 2,687 and 5,624 concepts and [*type_of*] links. We should add that, one can not simply start with the *animal* node in WordNet and find all its descendants/hyponyms to create a domain specific taxonomy. This is due to the fact that many terms have several meanings and each may have different hyponyms. Exhaustively following such hyponyms starting from a general term (e.g *animal*) would result in a very general taxonomy.

OntoHarvester Taxonomy: We use the animal ontology generated by OntoHarvester in the previous section, and considering only *type_of* links in this ontology, we create a taxonomy referred to as *OHTax*.

Benchmark Data Set: We create a simple benchmark by collecting the abstract of 440 animal names in the *initial animals list* from Wikipedia. Unfortunately, the other items in this list do not have any abstract in Wikipedia. We also assign three sets of possible topics to each abstract in the benchmark. i) *Title*: the animals name, ii) *Direct Cats.*: the direct categories provided for the *Title* by Wikipedia, and iii) *Indirect Cats.*: the indirect or secondary categories that are the ancestor of the largest number of direct categories for the abstracts. These possible topics are compared with the suggested topics by TID to evaluate the quality of taxonomies.

We feed *WNTax* and *OHTax* to the TID algorithm to suggest topics for abstracts in the benchmark. That is for each abstract, we run TID using the two taxonomies and get the ordered list of suggested topics. The result of this experiments are provided in Table 4.5. As shown in the third column of the table (in bold font), when *OHTax* is used, TID is able to find the exact title for 30.2% of the abstracts in the benchmark with its first ranked suggested topics. This is more than twice the same value for *WNTax*. Similar results hold for the other two topic sources (*Direct Cats.* and *Indirect Cats.*). In all cases, *OHTax* significantly outperforms *WNTax*, due to having more concepts related to the domain of animals.

Considering the top-20 topics suggested by TID, we compute the number of correctly identified topics from each topic source. Then for each abstract, we determine the position (rank) of the correct topic in ordered list. The average ranks is shown in

Table 4.5: Results of Topic Identification using for Taxonomies.

Taxonomy Name	**Topic Source**	**Rank-1 Topic # (%)**	**Avg. Rank**
OHTax	Title	**133** (30.2%)	**2.77**
OHTax	Direct Cats.	**78** (17.7%)	**4.47**
OHTax	Indrct. Cats.	**128** (29.1%)	**6.68**
WNTax	Title	56 (12.7%)	5.36
WNTax	Direct Cats.	35 (7.9%)	6.09
WNTax	Indrct. Cats.	97 (22.0%)	6.88

the fourth column of Table 4.5. The TID's results for *OHTax* are of higher quality than those for *WNTax*. More specifically, the average rank of the correctly reported topics using *OHTax* for the *Title* case is 2.77, which is about half of the same value for *WNTax*.

The above experiment mainly indicates that although WordNet (and similarly other general ontologies) may be of high quality, they are not comprehensive enough for domain-specific applications such as the one mentioned in this section. OntoHarvester, on the other hand, is able to use very small seed ontology (or existing ontologies) and a small set of application-specific textual documents to learn more comprehensive ontologies for the purpose of the application. This makes OntoHarvester a very powerful and practical tool for learning domain-specific ontologies.

4.4 Related Work

In the last two decades, many highly supervised approaches have been proposed to extract ontological information. In 1992, Bourigault [Bou92] used a two-phase algorithm to first analyze the corpora to find candidate terms and then parse them to find the final terminological units mainly using their grammatical structures. Based on this work, Drouin proposed a hybrid technique to extract terms from POS-tagged text and then to filter them based on some statistical techniques [Dro03]. Similar techniques were proposed by Voutilainen [Vou95] and Maedche et al. [MS00]. Unfortunately, most of these approaches are too expensive to scale up, due to the need for human validation.

A second group of related works employ statistical techniques or machine learning techniques to automatically extract ontologies; Loh et al. [LWO00] used fuzzy reasoning to calculate the likelihood of a term to be a concept. Pantel and Lin [PL01] proposed a language independent technique, in which they first extract high frequency two-word terms, and then extend them to multi-word terms using statistical techniques. In [QHF04] and [THF06], Quan et al. incorporated fuzzy logic into Formal Concept Analysis or FCA [GW99] to automatically extract ontologies. Another approach based on fuzzy logic is proposed by Lee et al. in [LKK07]. Parameswaran et al. used some variations of association rule mining techniques to find frequent terms and words from logs and tags [PGR10]. These approaches typically suffer from the following issues: 1) They need large training sets in order to provide highly accurate results, 2) they may need to be re-trained for each new domain, 3) they can not handle reordering of the words to make new terms (e.g. "*linear equation*" from "*linear or non-linear equation*") due to ignoring the morphological structures of the text, and 4) they highly rely on structured or semi-structured data sets which limits their coverage.

To address these issues, more NLP-based techniques have been proposed in recent years. Lin and Pantel [LP01] automatically found the similar paths in the dependency trees. These similar paths form a (binary) relation structure between terms and can be used in ontology generation systems. Banko et al. in [BCS07] introduced a new extraction paradigm, called OIE, in which some relations are extracted from the corpus in preprocessing time to facilitate the main process of text mining. Later, Poon and Domingos [PD10] proposed OntoUPS which is more robust to noise with respect to OIE. OntoUPS is based on a semantic parser called UPS [PD09]. UPS converts dependency trees to quasi-logical forms, and uses recursive reductions to abstract out syntactic variation. Kozareva et al. [KH10] used simple patterns to extract taxonomic relations from the Web. Later, [NVF11] improved their approaches by using more general patterns in the forms of World-Class Lattices [NV10]. Krishnamurthy and Mitchell [KM11] used the relations extracted from NELL's knowledge-base [CBK10a] to extract concepts and aliases. One of the few works showing that using *deeper* NLP-based techniques, such as parse tree mining, can provide more accurate results is CRCTOL proposed by Jiang and Tan in [JT10]. In CRCTOL, parse trees are used to find candidate terms (from NPs for instance), and other statistical techniques are employed to extract the concepts from the candidate terms.

Unlike OntoHarvester, most of these works either i) use very limited and shallow NLP-based techniques, ii) use a limited and fixed number of patterns (mostly borrowed from Hearst [Hea92]) to mine the text or parse trees, or ii) separate the concept extraction phase from relation extraction. Thus, they do not gain much over the pure statistical techniques.

We should also add that there are several systems to generate general ontologies or common-sense knowledge-bases. CYC [EG06], [GK06], FreeBase [BEP08], SOFIE [SSW09], DBPedia [BLK09], YaGo2 [HSB11], and ProBase [WLW12] are the most common such systems. However, we do not discuss them here, since they mostly work for structured data and are not designed for domain-specific ontology generation.

CHAPTER 5

IKBstore: Knowledge Integration through IBminer

As repeatedly mentioned throughout this dissertation, the Web provides a vast amount of high-quality information through structured, semi-structured, and unstructured sources. Projects such as Cyc [EG06, GK06], Kylin [WHW08], Freebase [BEP08], DBpedia [BLK09], YaGo2 [HSB11, SKW08], Probase [WLH11], and WikiData [Wik] generate a huge number of structured summaries. These techniques extract their knowledge either using highly supervised techniques or using structured or semi-structured data, and thus they are costly to scale. For instance, around 43.9% of pages in Wikipedia are missing their entire InfoBoxes and many others are missing part of their structured summaries. As a result, structured queries over individual KB will not provide accurate and complete enough results [CMH09]. This claim is also proven in Section 3.5.5, where we showed the current DBPedia is not able to answer to at least 56% of the most popular queries searched in Google. On the other hand, on-line encyclopedias, handbooks, manuals, and other semi-curated free-text corpora provide an even larger amount of knowledge in semi-structured, and unstructured (text) forms. However, due to the ambiguity and complexity of such data sources, they can not be directly used in structured querying. To make more effective structured querying and question answering systems, one need to integrated the existing sources of knowledge into a better-structured knowledge base of much improved coverage and accuracy.

Fortunately, although overlapped a lot, the mentioned KBs cover knowledge on various domains or topics. Moreover, there are several domian specific and publicly available KBs such as MusicBrainz [MUS], GeoNames [GEO], etc. that provide more information on more specific topics. In Order to unify these existing knowledge bases which are using heterogeneous formats and terminologies, we propose a knowledge integration system, called *IKBstore* [MGZ13c, MGZ13b]. *IKBstore* integrates existing KBs into a more standard KB presented with RDF triples and then completes it by mining unstructured and semi-structured data. More specifically, *IKBstore* performs the following four steps to create a knowledge base which is superior to the existing ones in coverage and consistency:

Step A: *IKBstore* first integrates the publicly existing structured data (starting with DBpedia [BLK09], WikiData [Wik], YaGo2 [HSB11], MusicBrainz [MUS], and Geonames [GEO]) into an *initial knowledge base*. To this end, *IKBstore* mostly utilizes the interlinking information available in DBpedia, and then presents the information in RDF triple format (Section 5.1.) These triples are then stored in an Apache Cassandra database.

Step B: Next, *IKBstore* uses our *IBminer* system to mine more structured information from available unstructured data in Web (Currently Wikipedia's text) as described in Chapter 3. In this way, we extract missing structured information that should belong to InfoBoxes of Wikipedia and other semi-curated on-line collections. We shortly discuss the detail of this step in Section 5.2.

Step C: Once the initial KB is created, *IKBstore* use a *Context-aware Synonym Suggestion System* (CS^3) [MGZ13a, MGZ13b] to reconcile different terminologies used in different resources. This step, which significantly improves the consistency of the KB, is an extension to *IBminer*'s technique for suggesting secondary matches and is discussed in Section 5.3

Step D: As demonstrated in [MGZ13c], *IKBstore* provides a user-friendly interfaces to help volunteer contributors to revise and improve the final knowledge bases. This step, which is further discussed in Section 5.4, allows users to edit the KB without knowing its internal terminology. Therefore, this tool is a great asset for more effective crowdsourcing

In this chapter, we explain each of the above four steps in the context of our large experiment on the entire Wikipedia dataset. This data set contains all subjects and their plain text provided by Wikipedia, which has 4.4 Million subjects each with 18.2 sentences on average. Note that for this data set we use the entire text (up to 200 sentences per page) while for Musicians, Actors. and Institutes data sets, we only used their abstracts. The experiments were run using the Hoffman2 cluster at UCLA [Hof] with up to 256 cores each with 8GB of main memory and a machine with 64 cores of 2.3GHz and 256GB of main memory.

5.1 Step A: Integrating Knowledge Bases

In this part, we explain the initial process of collecting the existing knowledge bases and integrating them using the exiting interlinks. Each piece of information generated in this phase is tagged with its provenance which is mainly the source names including that piece. This may later be used to rank the users browsing and querying results.

5.1.1 Data Collection

We will consider several publicly available knowledge bases (Table 5.1) and integrate them as explained later in this section. Among the introduced knowledge bases, there are some domain specific ones (e.g. MusicBrainz, Geonames, etc.), and some domain independent ones (e.g. DBpedia, YaGo2, etc.). Although pieces of knowledge in these sources are represented in various ways, we represent them in the form of triples <*Subject*, *Attribute*, *Value*> and store them in Apache Cassandra which is designed for handing very large amount of data. We recognize three main types of triples:

InfoBox triples: These triples provide information on a known subject in the subject/attribute/value format. For easing our discussion we refer to these as InfoBoxes

Name	**Size (GB)**	**# of Entities** (10^6)	**# of Triples** (10^6)
ConceptNet [SLM02]	3.0	0.30	1.6
DBpedia [BLK09]	43.9	3.77	400
FreeBase [BEP08]	85.0	≈25	585
Geonames [GEO]	2.2	8.3	90
MusicBrainz [MUS]	17.6	18.3	≈131
NELL [CBK10b]	1.3	4.34	50
OpenCyc [OPE]	0.2	0.24	2.1
YaGo2 [HSB13]	19.8	2.64	124
WikiData (Eng.) [Wik]	8.5	4.4	12.2

Table 5.1: Public Knowledge Bases used in IKBstore

(<*J.S. Bach*, *PlaceofBirth*, *Eisenach*>).

Subject/Category triples: They provide the categories that a subject belongs to in the form of *subject/link/category* where, *link* represents a taxonomical relation (<*J.S. Bach*, *isA*, *German Composers*>).

Category/Category triples: They represent taxonomical links between categories(<*German Composers*, *isA*, *German Musicians*>).

Currently, we have converted all the knowledge bases listed in Table 5.1 into the above triple formats.

5.1.2 Data Integration

Knowledge bases introduced in Table 5.1 contain a wealth of knowledge as the numbers indicate. However, lack of a standard ontology in these knowledge bases makes them very challenging to integrate. Our main goal is to tackle this challenge and discover the initial interlinks between subjects, attributes, and categories in order to eliminate duplication, align attributes, and improve consistency.

Interlinking Subjects: Fortunately, the same names are often used to denote the same subjects in different databases. Moreover, DBPedia is interlinked with many existing knowledge bases, such as YaGo2 and FreeBase, which can serve as a source of subject interlinks. For the knowledge bases which do not provide such interlinks (e.g. NELL), in addition to exact matching, we use both synonym matching and attribute similarity. Synonyms can be obtained from links such as *redirect* and *sameAs*, WordNet, or using our recently proposed ontology generator OntoHarvester [MKI13b]. For subjects with matching names, we only merge them if they have a predefined number of attribute names in common.

Interlinking Attributes: Similar to the subject interlinking, we use exact matching and synonym matching for finding initial attribute interlinks. In Section 3.3.5, we explain how one can find synonym attributes and interlink different aliases for the attribute names used in different knowledge bases.

Interlinking Categories: In addition to exact matching, we compute the similarity of the categories in different knowledge bases based on their instances. Consider two categories c_1 and c_2, and Let $S(c)$ be the set of subjects in category c. The similarity function for categories interlink is defined as $Sim(c_1, c_2) = |S(c_1) \cap S(c_2)|/|S(c_1) \cup S(c_2)|$. If the $Sim(c_1, c_2)$ is greater than a certain threshold, we consider c_1 and c_2 as aliases of each other, which simply means that if the instances of two categories are highly overlapping, they can be assumed to represent the same category.

After retrieving these interlinks, we will merge similar triples based on the retrieved interlinks. Currently this task is finished for Geonames and MusicBrainz and partially done for DBpedia and YaGo2. We are quickly covering more of these two knowledge bases as well as the remaining ones in Table 5.1. All triples are also assigned with accuracy confidence and frequency values. As explained in Section 2.3.2, more evidence supporting the same piece of information will increase its confidence and frequency values. Observe that the sources from which the triples are generated are also stored for the provenance purposes.

Integrating current knowledge bases using the techniques explained above has several advantages discussed next. A) The integrated knowledge base covers more structured summaries, which results in richer and more confident patterns for *IBminer*. B) More attributes are encountered, since the focus on different sources are different. This also improves *IBminer*'s performance as explained in the next section. C) The use of multiple knowledge bases represents an effective way to validate preexisting structured summaries and those newly generated from text. Using these techniques, we can also evaluate the quality of the textual part of a page.

5.2 Part B: Improving the Knowledge Base using Text Mining

Once the initial knowledge base is created, we use *IBminer* to improve the knowledge base by mining the available text in Wikipedia as explained in Section 3. This time we use the all subjects and their entire text in Wikipedia. In this way, many of the missing structured summary information in current knowledge bases will be added to the *IKBstore* which significantly improves the coverage of the knowledge base. Similar to the previous case, each newly generated triple is tagged with provenance information for later ranking purposes.

5.2.1 Generating new Structured Summaries

We set N_C=20 and L=3 which are slightly lower than what suggested in Section 3.5.3, since this data set is a lot larger and much more examples can be found for each categories. To generate the TextGraphs and semantic links (Phase I), we distributed the task over up to 256 cores in Hoffman2 cluster [Hof]. This task, which took around a month, has resulted in 4.5 Billion semantic links out which 251 Million have subjects matching their page title. To control the size of the experiment, we only used these 251 Million links for the rest of the experiment ($|T_n|$=$251M$). Out of these semantic links 8.2 Million match the existing InfoBox triples ($|T_m|$=$8.2M$) which are used to generated the PM and PAS structures using our 64-core machine. Finally, we used these two structures to suggest the final InfoBox triples.

Similar to Section 3.5.2, we graded around 50K of the generated triples and provide the precision/recall diagrams for Best and Secondary Matches respectively in Figures

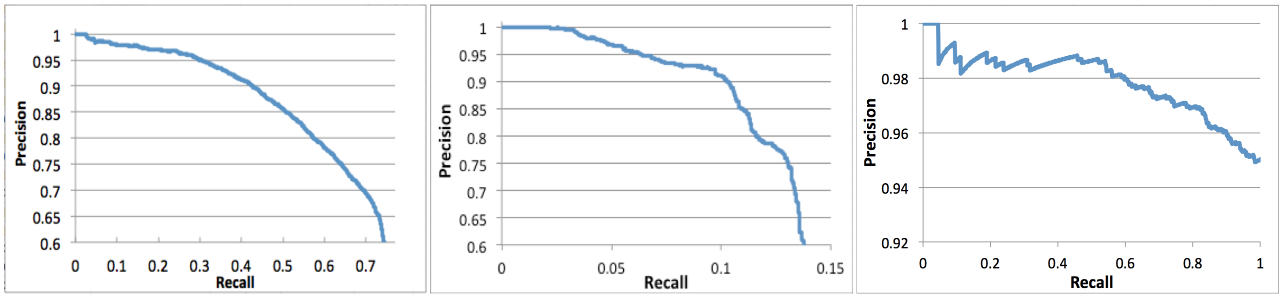


Figure 5.1: Evaluation of (a) best matches, and (b) secondary matches (c) attribute mapping for the entire Wikipedia.

5.1.(a) and 5.1.(b). Again in both diagrams, the normalized threshold (τ) decreases from left to right. As compared to the Musicians case, the precision in this case starts to drop earlier mainly due to lower quality of the information in the entire Wikipedia compared to that of Musicians. Notice that concepts related to Musicians are generally more popular and receive more attention from Wikipedia's users. This results in higher quality of text and InfoBoxes in Musicians pages. Nevertheless, *IBminer* is able to generated 3.9 Million triples with 95% accuracy. With 90% accuracy we are able to recall up to 7.1 Million triples.

To evaluate the quality of the PM structures independent from the errors introduced by the NLP parsers and the SemScape system, we also evaluated the triples generated from the T_m set. Remember that T_m includes semantic links such as $<s, l, v>$ for which there is at least one triple in exiting InfoBoxes (T_i) such as $<s, a, v>$. That is, subject s and value v are most probably extracted correctly by the NLP component, and the only thing needs to be done is to map l to a or one of its synonyms. This time, we graded a sample of 2K translated triples from T_m and reported the precision/recall diagram in Figure 5.1(c). As can be seen, the mapping results for all such cases is performed with 95% accuracy. It is interesting to note that mapping precision for the 7.1 million newly generated triples is above 98%, which is very significant.

We should mention that the number of results we generated for Musicians was higher than the other domains due to the higher quality of the information in their pages as discussed in the previous subsection. As mentioned earlier for the entire Wikipedia, we were able to generate up to 7.1 Million InfoBox triples. Noticing that these triples are generated mostly from the pages with "*useful*" text, *IBminer* adds 2.8 new triples per page. By "useful" text, we mean the text for which SemScape is able to generate at least one semantic link. Only 58% of pages in Wikipedia have "useful" text, and the other pages either are empty, contain only tabular information, or are redirect or disambiguation pages.

To understand how these new triples will impact the quality of search over DBpedia, we performed an application-based evaluation in Subsection **??**. The experiment showed that by adding *IBminer*'s triples to DBpedia, we observe 53.3% improvement in the number of results for the users query, although the increase in DBpedia's actual size is much less ($\approx$21%). This mainly implies that *IBminer* is providing higher quality and query-relevant information than DBpedia. To understand why, we studied DBpedia and summarized some interesting facts next. DBpedia suggests 44K attribute names out of which 27K are used less than 10 times and thus do not provide any use for structured querying. These attributes are mostly introduced by the naive and imprecise technique DBpedia uses to convert tables into triple format. The very same technique also has introduced many *unimportant triples* ($>17\%$) (e.g. file names, image names, formats, size, or alignment), duplicate triples ($>7\%$), wrong triples ($>4\%$), and incomplete triples[1] ($>10\%$). The reported percentages here are estimated using a small sample of DBpedia. More accurate statistics may be estimated by a larger sample, however the numbers already prove the low quality of the triples in DBpedia. In fact, out of 54 Million triples in DBpedia only 12.2 Millions are generated from Wikipedias InfoBoxes (according to WikiData [Wik]). The rest is largely generated from other structured information in Wikipedia and is not reliable. Briefly, a very optimistic estimation indicates that more than 38% of triples in DBpedia are not suitable for structured queries.

[1]These are triples that have lost their meaning due to being separated from other values in the same row of a table.

Although we used 90% accuracy in this study, much lower precisions in knowledge bases are acceptable in practice for the following reasons. First, many of the wrong results are for less important attributes (e.g. *name* and *caption* attributes) or unrelated values. These cases are usually less probable to be appeared in users' queries. Second, the main priority of a knowledge based for structured querying is the coverage (not accuracy). In fact, structured queries on the extended DBpedia are providing more precise results than current keyword-based search engines. Third, the quality of our results is completely comparable (if not better) with many current KBs, including those which have been work on for more than forty years [EG06].

5.2.2 Verifying Existing Structured Summaries

More evidence for the same piece of information is a good indicator of its correctness. However, the knowledge from different sources is usually represented in different ways. Therefore, *IKBstore* assigns initial weights to different sources, combines their triples, and calculates the significance and accuracy values of the triples based on their initial weights, evidence frequency, the patterns correctness confidence, etc. If the same piece of information is generated from the text, *IKBstore* accordingly updates its significance and accuracy.

We also try to find mismatches between items generated by *IBminer* and those were already part of the initial knowledge base. We say two triples $<s_1$, l_1, $v_1>$ and $<s_2$, l_2, $v_2>$ mismatch if $s_1{=}s_2$, $v_1{=}v_2$, $l_1{\neq}l_2$, and l_1 and l_2 are not synonyms. These mismatches are reported as incorrect summaries if their normalized wight is above a predefined threshold. Our experiments indicate that *IBminer* reports 1.2% of the existing summary items as potential incorrect items, whereas only 57% of those are actually incorrect (a false negative rate that is under $1.2 \times 0.57 = 0.52\%$).

5.2.3 InfoBox Templates Suggestion

Finding a right InfoBox template (or simply a list of relevant attributes) for a subject can be a very time consuming task for typical Web users. To alleviate this issue, *IBminer* (through IBE as explained in Section 5.4) suggests the most relevant attribute names for a given subject. To this end, *IBminer* first finds the most popular attributes currently used for the subjects in each category in our knowledge base. Then, for a subject of interest, all popular attributes from the categories listed for the subject are suggested as relevant attributes. For each attribute, say α_1, we also find the most popular attributes used with α_1 in the same InfoBox (i.e. for the same subject). Based on this co-occurrence data set, we suggest missing attributes which their counterpart is already in the current attributes for the subject. With this feature, users would use more standard attribute names and are more likely to enter structured information.

5.3 Step C: Context-aware Synonyms to Improve Consistency

The integrated knowledge base so obtained will represent a big step forward, since it will (i) improve coverage, quality, and consistency of the knowledge available to Semantic Web applications and (ii) provide a common ground for different contributors to improve the knowledge base in a more standard and effective way. However, a serious obstacle in achieving such a desirable goal is that different systems do not adhere to a standard terminology to represent their knowledge, and instead use plethora of synonyms and polynyms.

Thus, we need to resolve *synonyms* and *polynyms* for the entity names as well as the attribute names used in these knowledge bases. For example, by knowing '*Johann Sebastian Bach*' and '*J.S. Bach*' are synonyms, the knowledge base can merge their triples and associate them with one single name. As for the polynyms, the problem is even more complex. Most of the time based on the context (or popularity), one should decide the correct polynym of a vague term such as '*JSB*' which may refer to

"*Johann Sebastian Bach*", '*Japanese School of Beijing*', etc. Several efforts to find entity synonyms have been reported in recent years [CCC12, CGX09a, CGX09b, CLP12, PCB09]. However, the synonym problem for attribute names has received much less attention, although they can play a critical role in query answering. For instance, the attribute '*birthdate*' can be represented with terms such as '*date of birth*', '*wasbornindate*', '*born*', and '*DoB*' in different knowledge bases, or even in the same one when used in different contexts. Unless these synonyms are known, a search for musicians born, say, in 1685 is likely to produce a dismal recall. WikiData [Wik] is one of the pioneer approaches for recognizing such synonyms in large scales, but it unfortunately is highly supervised and mangle is designed only for InfoBoxes in Wikipedia.

To address these issues, we proposed a Context-aware Synonym Suggestion System (CS^3 for short) [MGZ13a, MGZ13b], which is used to improve the consistency of *IKB-store*. CS^3 learns attribute synonyms by matching morphological information in free text to the existing structured information. Similar to *IBminer*, CS^3 takes advantage of a large body of categorical information available in Wikipedia, which serves as the contextual information. Then, CS^3 improves the attribute synonyms so discovered, by using triples with matching subjects and values but different attribute names. After unifying the attribute names in different knowledge bases, CS^3 finds subjects with similar attributes and values as well as similar categorical information to suggest more entity synonyms. Through this process, CS^3 uses several heuristics and takes advantage of currently existing interlinks such as DBpedia's *alias*, *redirect*, *externalLink*, or *sameAs* links as well as the interlinks provided by other knowledge bases.

Synonyms are terms describing the same concept, which can be used interchangeably. According to this definition, no matter what context is used, the synonym for a term is fixed (e.g. '*birthdate*' and '*date of birth*' are always synonyms). However, the meaning or semantic of a term usually depends on the context in which the term is used. The synonym also varies as the context changes. For instance, in an article describing IT companies, the synonym of the attribute name '*wasCreatedOnDate*' most probably is '*founded date*'. In this case, knowing that the attribute is used for the name of a company is a contextual information helping us find an appropriate synonym for '*wasCreatedOnDate*'. However, if this attribute is used for something else, such as an invention, one can not use the same synonym for it.

Being aware of the context is even more useful for resolving polynymous phrases, which are in fact much more prevalent than exact synonyms in the knowledge bases. For example, consider the entity/subject name '*Johann Sebastian Bach*'. Due to its popularity, a general understanding is that the entity is describing the famous German classical musician. However, what if we know that for this specific entity the birthplace is in '*Berlin*'. This simple contextual information will lead us to the conclusion that the entity is referring to the painter who was actually the grandson of the famous musician *Johann Sebastian Bach*. A very similar issue exists for the attribute synonyms. For instance, '*birthdate*' can be a synonym for '*born*' when it is used with a value of type '*date*'; but if '*born*' is used with a value indicating locations, '*birthplace*' should be considered as its synonym.

CS^3 constructs a structure called *Potential Attribute Synonyms* (*PAS*) to extract attribute synonym. In the generation of *PAS*, CS^3 essentially counts the number of times each pair of attributes are used between the same subject and value and with the same corresponding semantic link in the TextGraphs. Similar to *IBminer*, the context here is considered to be the categorical information for the subject and the value. These numbers are then used to compute the probability that any given two attributes are synonyms. Next subsection describes the process of generating *PAS*. Later in Subsection 5.3.2, we will discuss our approach to suggest entity synonyms and improve existing ones.

5.3.1 Generating Attribute Synonyms

Intuitively, if two attributes (say '*birthdate*' and '*dateOfBirth*') are synonyms in a specific context, they should be represented with the same (or very similar) semantic links in the TextGraphs (e.g. with semantic links such as '*was born on*', '*born on*', or '*birthdate is*'). In simpler words, we use text as the witness for our attribute synonyms. Moreover, the context, which is defined as the categories for the subjects (and for the values), should be very similar for synonymous attributes.

More formally, let attributes α_i and α_j be two matches for link l in initial triple $<s, l, v>$. Let $N_{i,j}$ $(= N_{j,i})$ be the total number of times both α_i and α_j are the interpretation of the same link (in the initial triples) between category sets C_s and C_v. Also, let N_x be the total number of time α_x is used between C_s and C_v. Thus the probability that α_i (α_j) is a synonym for α_j (α_i) can be computed by $N_{i,j}/N_j$ $(N_{i,j}/N_i)$. Obviously this is not always a symmetric relationship (e.g. '*born*' attribute is always a synonym for '*birthdate*', but not the other way around, since '*born*' may also refer to '*birthplace*' or '*birthname*' as well). In other words having N_i and $N_{i,j}$ computed, we can resolve both synonyms and polynyms for any given context (C_s and C_v).

With the above intuition in mind, the goal in *PAS* is to compute N_i and $N_{i,j}$. Next we explain how CS^3 constructs *PAS* in one-pass algorithm which is essential for scaling up our system. For each two records in *PM* such as $<c_s$, l, $c_v>$: α_i and $<c_s$, l, $c_v>$: α_j respectively with evidence frequency e_i and e_j $(e_i \leq e_j)$, we add the following two records to PAS:

$$<c_s, \alpha_i, c_v>: \alpha_j$$

$$<c_s, \alpha_j, c_v>: \alpha_i$$

Both records are inserted with the same evidence frequency e_i. Note that, if the records are already in the current *PAS*, we increase their evidence frequency by e_i. At the very same time we also count the number of times each attribute is used between a pair of categories. This is necessary for estimating N_i and computing the final weights for the attribute synonyms. That is for the case above, we add the following two *PAS* records as well:

$<c_s, \alpha_i, c_v>$: '' (with evidence e_i)

$<c_s, \alpha_j, c_v>$: '' (with evidence e_j)

Improving *PAS* with Matching InfoBox Items: Potential attribute synonyms can be also derived from different knowledge bases which contain the same piece of knowledge, but in different attribute names. For instance let $<J.S.Bach$, $birthdate$, $1685>$ and $<J.S.Bach$, $wasBornOnDate$, $1685>$ be two InfoBox triples indicating $bach$'s birthdate. Since the subject and value part of the two triples matches, one may say $birthdate$ and $wasBornOnDate$ are synonyms. To add these types of synonyms to the *PAS* structure, we follow the exact same idea explained earlier in this section. That is, consider two triples such as $<s$, α_i, $v>$ and $<s$, α_j, $v>$ in which α_i and α_j may be a synonym. Also, let s and v respectively belong to category sets $C_s = \{c_{s1}, c_{s2}, ...\}$ and $C_v = \{c_{v1}, c_{v2}, ...\}$. Thus, for all $c_s \in C_s$ and $c_v \in C_v$ we add the following triples to *PAS*:

$<c_s, \alpha_i, c_v>$: α_j (with evidence 1)

$<c_s, \alpha_j, c_v>$: α_i (with evidence 1)

This intuitively means that from the context (category) of c_s to c_v, attributes α_i and α_j may be synonyms. Again more examples for these categories and attributes increase the evidence which in turn improve the quality of the final attribute synonyms. Much in the same way as learning from initial triples, we count the number of times that an attribute is used between any possible pair of categories (c_s and c_v) to estimate N_i.

We should also note that with this simple extension any other knowledge bases (e.g., Wikidata) can be added to *IKBstore* and our CS^3 system will interpret its terminology to that of *IKBstore* using the matching triples between the two KB.

Generating Final Attribute Synonyms: Once *PAS* structure is built, it is easy to compute attribute synonyms as described earlier. Assume we want to find best synonyms for attribute α_i in InfoBox Triple t=<s, α_i, v>. Using *PAS*, for all possible α_j, all $c_s \in C_s$, and all $c_v \in C_v$, we aggregate the evidence frequency (e) of records such as <c_s, α_i, c_v>: α_j in *PAS* to compute $N_{i,j}$. Similarly, we compute N_j by aggregating the evidence frequency (e) of all records in form of <c_s, α_i, c_v>: ‘’. Finally, we only accept attribute α_j as the synonym of α_j, if $N_{i,j}/N_i$ and $N_{i,j}$ are respectively above predefined thresholds τ_{sc} and τ_{se}. At the end, a very same normalization technique explained in Section 3.3.3 is used to perform a better filtering.

Completing Knowledge by Attribute Synonyms: In order to compute the precision and recall for the attribute synonyms generated by CS^3, we use our large initial knowledge bases and the PM structure generated for the entire Wikipedia text to construct the PAS structure as described above. In total, this PAS contains more than 89 million potential attribute synonym records, which are used to find synonymous attributes for all the triples in our initial knowledge base. Note that these synonyms are for already existing InfoBox triples, and thus different from the secondary matches explained in Chapter 3. Similar to our evaluation for the Best and Secondary matches in Section 3.5.2, we graded 100K of the resulted synonymous triples and provide the Precision/Recall diagram in Figure 5.2. The normalized threshold (τ) in this digram increases from left to right. As can be seen in this figure, CS^3 is able to find more than 55% of all possible synonyms with more than 90% accuracy. In fact, this is a very big step in improving structured query results, since it adds ≈5.7 million triples to *IKBstore*. These triples improve the consistency of the knowledge base by providing more popular (common) synonymous attributes for those used in different knowledge bases.

5.3.2 Generating Entity Synonyms

There are several techniques to find entity synonyms. Approaches based on the string similarity matching [Nav01], manually created synonym dictionaries [SR98], automatically generated synonyms from click log [CLP10, CLP12], and synonyms generated by other data/text mining approaches [Tur01, MKI13b] are only a few examples of

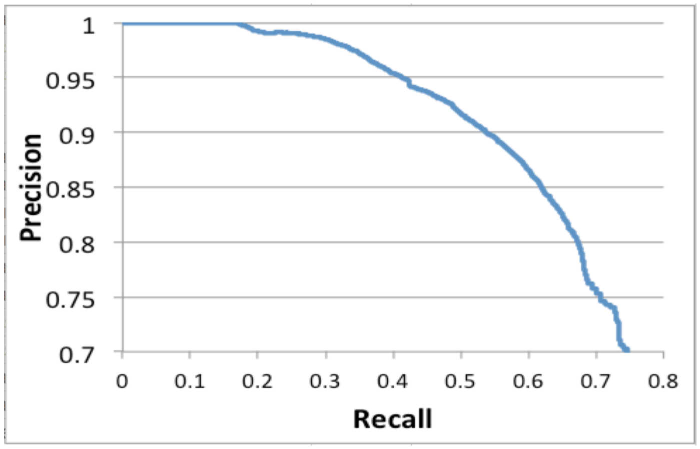


Figure 5.2: Evaluation of attribute synonyms for existing InfoBoxes in the initial knowledge base.

such techniques. Although performing very well on suggesting context-independent synonyms, they do not explicitly consider the contextual information for suggesting more appropriate synonyms and resolving polynyms.

Very similar to context-aware attribute synonyms in which the context of the subject and value used with an attribute plays a crucial role on the synonyms for that attribute, we can define context-aware entity synonyms. For each entity name, CS^3 uses the categorical information of the entity as well as all the InfoBox triples of the entity as the contextual information for that entity. Thus to complete the exiting entity synonym suggestion techniques, for any suggested pair of synonymous entities, we compute entities context similarity to verify the correctness of the suggested synonym.

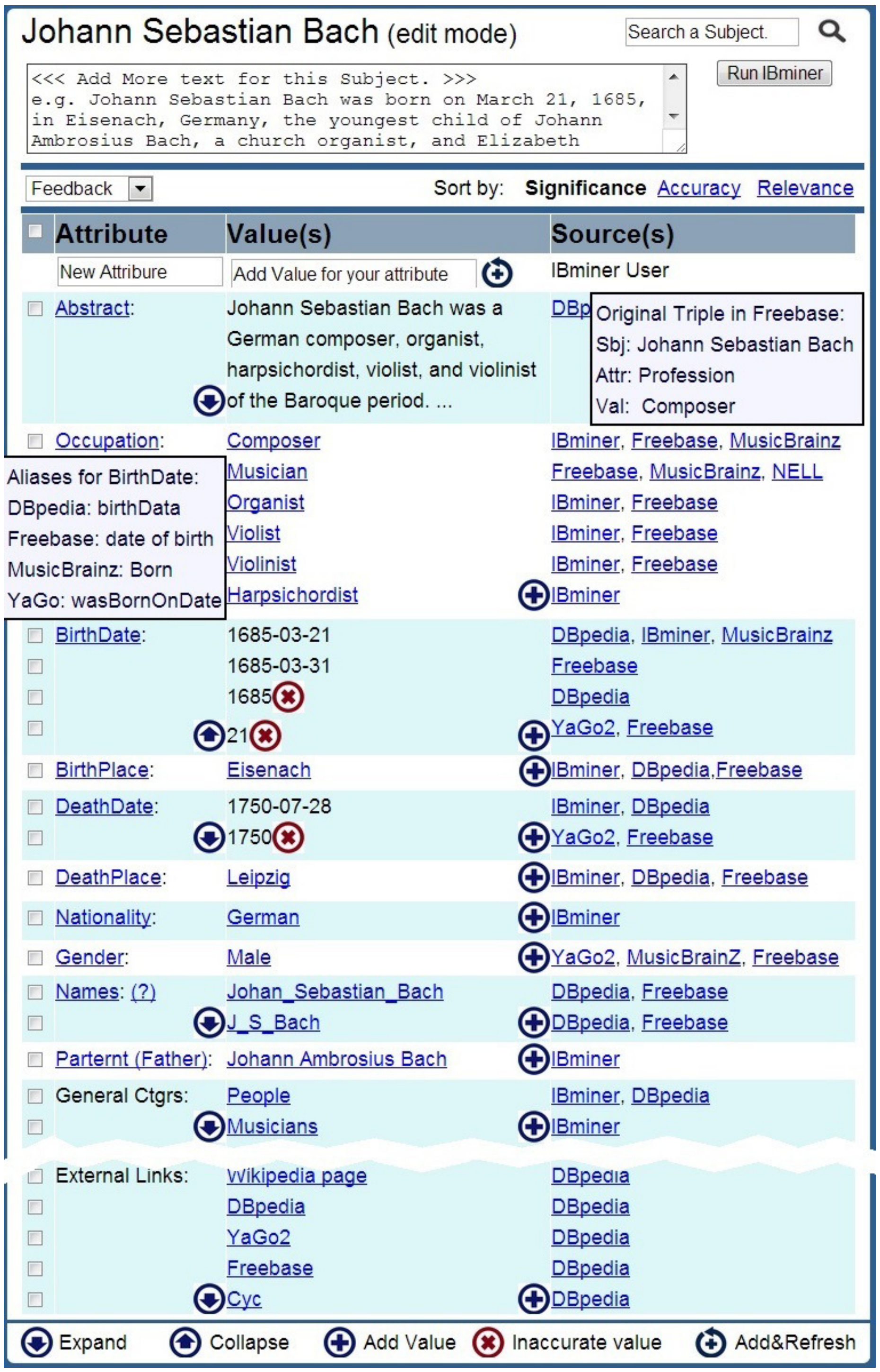
It is important to understand that this approach should be used as a complementary technique over the existing ones for two main reasons. First, context similarity of two entities does not always imply that they are synonyms specially when many pieces of knowledge are missing for most of entities in the current knowledge bases. Second, it is not feasible to compute the context similarity of all possible pairs of entities due to the large number of existing entities. In this work, we use the OntoMiner system [MKI13b] in addition to simple string matching techniques (e.g. Exact string matching, having common words, and edit distance) to suggest initial possible synonyms.

Let '*Johann Sebastian Bach*' and '*J.S. Bach*' be two synonyms that two different knowledge bases are using to denote the famous musician. A simple string matching would offer these two entity as being synonyms. Thus we compare their contextual information and realize that they have many common attributes with similar values for them (e.g. same values for attributes *occupation*, *birthdate*, *birthplace*, etc.). Also they both belong to many common categories (e.g. *Cat:German musician*, *Cat:Composer*, *Cat:people*, etc.). Thus we suggest them as entity synonyms with high confidence. However, consider '*Johann Sebastian Bach (painter)*' and '*J.S. Bach*' entities. Although the initial synonym suggestion technique may suggest them as synonyms, since their contextual information is quite different (e.i. they have different values for common attributes *occupation*, *birthplace*, *birthdate*, *deathplace*, etc.) our system does not accept them as being synonyms.

5.4 Step D: Tools for Crowdsourcing

Perhaps the most successful crowdsourcing story is that of Wikipedia, in where the amount of data has grown at least 10 times in less than 10 years, making Wikipedia the most commonly used encyclopedia. However, in recent years this growth is decreasing due to two main reasons: i) the crowds are usually good at more general knowledge, but for narrower and more specific domains, only few experts can contribute, and ii) with the data getting larger, crowdsourcing and editing data require sophisticated tools to assist the contributors. In fact the latter, is one of the main reasons hindering the Semantic Web from reaching ambitious goal of annotating the Web documents. Forcing contributors to use a common terminology is another big challenge. Moreover, users usually require advance searching and browsing tools to find the right place to enter their information or to find similar cases to avoid redoing an already performed task.

Therefore, we provided two easy-to-use tools, referred to as *InfoBox Knowledge-Base Browser* (*IBKB*) and *InfoBox Editor* (*IBE*). The former is for browsing, retrieving,



Johann Sebastian Bach (edit mode)

Search a Subject.

```
<<< Add More text for this Subject. >>>
e.g. Johann Sebastian Bach was born on March 21, 1685,
in Eisenach, Germany, the youngest child of Johann
Ambrosius Bach, a church organist, and Elizabeth
```

Run IBminer

Feedback

Sort by: **Significance** Accuracy Relevance

Attribute	Value(s)	Source(s)
New Attribure	Add Value for your attribute	IBminer User
Abstract:	Johann Sebastian Bach was a German composer, organist, harpsichordist, violist, and violinist of the Baroque period. ...	DBp
Occupation:	Composer	IBminer, Freebase, MusicBrainz
	Musician	Freebase, MusicBrainz, NELL
	Organist	IBminer, Freebase
	Violist	IBminer, Freebase
	Violinist	IBminer, Freebase
	Harpsichordist	IBminer
BirthDate:	1685-03-21	DBpedia, IBminer, MusicBrainz
	1685-03-31	Freebase
	1685	DBpedia
	21	YaGo2, Freebase
BirthPlace:	Eisenach	IBminer, DBpedia,Freebase
DeathDate:	1750-07-28	IBminer, DBpedia
	1750	YaGo2, Freebase
DeathPlace:	Leipzig	IBminer, DBpedia, Freebase
Nationality:	German	IBminer
Gender:	Male	YaGo2, MusicBrainZ, Freebase
Names: (?)	Johan_Sebastian_Bach	DBpedia, Freebase
	J_S_Bach	DBpedia, Freebase
Parternt (Father):	Johann Ambrosius Bach	IBminer
General Ctgrs:	People	IBminer, DBpedia
	Musicians	IBminer
External Links:	Wikipedia page	DBpedia
	DBpedia	DBpedia
	YaGo2	DBpedia
	Freebase	DBpedia
	Cyc	DBpedia

Original Triple in Freebase:
Sbj: Johann Sebastian Bach
Attr: Profession
Val: Composer

Aliases for BirthDate:
DBpedia: birthData
Freebase: date of birth
MusicBrainz: Born
YaGo: wasBornOnDate

Figure 5.3: A sample view of the InfoBox Editor (IBE) page. By hovering over the source names (as shown for *Freebase*) users can see the original triple in that source. Similarly, users can see the synonyms used for each attribute by hovering over that attribute name (as shown for *BirthDate*).

and querying the knowledge base, while the latter is implemented for enhancing the manual process of generating structured information by the users such as in Wikipedia and WikiData [Wik]. The prototype version of this tool (available at [Sem]) was demonstrated at PVLDB 2013 [MGZ13c] and attracted much attention. As opposed to the existing systems, IBE hides the internal structure of the KB from the users and implicitly leads them to use the correct terminology. In this way, users do not need to know the internal terminology of the KB in order to improve it, which will significantly encourage more users to contribute. Figure 5.3 shows the IBE user interface for the subject "*J.S. Bach*". We next explain main features of these two tools:

The InfoBox Knowledge-Base Browser (IBKB): IBKB is implemented to let users browse the current knowledge base. Its user interface is very similar to the one for IBE (Figure 5.3). For a given subject, the tool provides i) structured summary items in user specified order, ii) the synonyms found by *IBminer* for the attributes used in the summaries, and iii) wrong summary items recognized by *IBminer*. The tool can also determine the provenance of each piece of information and report the knowledge base from which it was originally taken, or that it was actually discovered by *IBminer*. By clicking on each source name, the user will be provided with the original form of the triple in that source. Each entity in the result pages is also connected to its own page to make the browsing easier for the users.

Using IBKB, users can select one or more summary items from the user interface, and provide their feedback on the correctness, relevance, and significance of the items. In addition to using such feedback to improve *IBminer*'s performance and to tune its patterns, users' feedback will be used to rank the structured summaries. The ranking mainly improves the user experience with IBKB, since many of the provided summaries are just common sense information for the users, and they usually do not want to see them on top of the list of summaries.

More importantly, we have equipped IBKB with the powerful query-by-example system, SWiPE, initially proposed by Atzori and Zaniolo in [AZ12]. Using SWiPE, users can specify their query by i) starting from a similar page to what they are looking for, ii) specifying their conditions on the appropriate attribute names shown for the selected page, and iii) click on the search button. Receiving users' queries, IBKB translates it into the SPARQL query and uses Apache Jena to answer the query over our integrated KB.

The InfoBox Editor (IBE): In addition to the browsing tool for the current knowledge base, we provide an easy-to-use tool, referred to as IBE, for enhancing the manual process of generating structured information by the users such as in Wikipedia. Figure 5.3 shows the IBE user interface for the subject "*J.S. Bach*". For the existing subjects, IBE allows users to add more textual information and structured summaries. To create a new subject, users are asked to enter the name (a descriptive subject), one or more categories for the subject, and a descriptive text for it. They can optionally add as many structured summaries as they desire. IBE specifically provides the following mechanisms for the used to improve the KB:

- IBE automatically suggests missing attribute names for subjects (as explained in Section 5.2.3), so users can fill the missing values.
- Similarly, IBE automatically suggests missing categories, and thus users only need to verify the correctness of the suggested information.
- Users will be able to provide feedback on correctness, importance, and relevance of each piece of information.
- Finally, users can insert their knowledge in free text (e.g, by cutting and pasting text from Wikipedia and other authorities), and IBE employs *IBminer* to convert them into the structured information. After users final verification, the structured information will be added to the KB. This is perhaps the most natural way for inserting structured information.

Notice that the main difference between IBE and Wikipedia is that IBE's focus is on generating structured summaries for machine use that requires a standard ontology, while Wikipedia is mainly designed for human readers. Moreover, IBE is able to automatically generate structured summaries and suggest InfoBox templates so users can provide structured summaries more efficiently.

Semantic Web Annotation: A final feature of IBE will enable the Semantic Web users to annotate their documents in a semi-supervised approach. In other words, instead of manually annotating the documents, IBE will perform the annotation task and users only need to verify the final annotations. In this way, not only we reduce typical Web users task to publish their Semantic Web-friendly documents, but we also provide a set of more standard annotations which consequently reduce the efforts required for sharing, reusing, and querying the contents.

CHAPTER 6

Synopses and Summarization Techniques on massive data

In this section, we provide some preliminary and background information on synopses and summarization techniques in data streams or massive data sets. For many resource-intensive applications, it is acceptable to provide approximate results to reduce time and space requirements of the algorithms. This is true for applications on both static data sets and data streams. In many cases, there is no other choice than estimating the final results due to time and space constraints, especially for data streams in which it is crucial to provide online results. Thus, we need some light structures to help maintain approximate results for the new-coming data items and avoid passing the whole data set more than once [GM99].

In data streaming environment, as already discussed, we need to continually provide the results in an online fashion [TGN92, BW01]. This basically means that for each new-coming piece (or chunk) of data, the system should recompute (or update) the results of the query. Thus, another important reason for designing synopses and looking for more efficient summarization techniques is to be able to provide faster continuous results.

One of the important uses of synopses is on query optimization. Query Optimizers need different statistics to be able to estimate the selectivity of certain operations in order to plan the execution of complex queries. In data streams, this issue will be more important since the process might take longer and the results should be provided instantly (in an online fashion.) In Some application such as those in OLAP and data mining environment, the absolute size of intermediate results are needed to ensure that the results are correctly extracted.

In many applications, processing time of a query would not be an issue if one can have the whole data set in the memory (e.g aggregates such as min/max in sliding window). However, the whole data set may be too large to fit into memory. Therefore, synopses are a good solution for reducing the memory requirement of the algorithms at the cost of providing approximate results instead of the exact ones [GM99]. Reducing storage requirements for massive data sets and data streams is also an important practice in reducing the transferring time of the data in some distributed environments.

Although synopses are very useful and inevitable structures in massive data sets and data streaming systems they have some known drawbacks as well [CM05]:

- They may have $\Omega(1/\epsilon)$ or more multiplicative factor in their space requirements, where ϵ the approximate rate.
- The maintenance of the synopses for new, updated, and/or expired items may take linear time to the size of the synopses, which is not acceptable for many data streaming environments.
- Most synopses are useful for a limited class of queries, and there are not enough works on designing synopses for semi-structured and unstructured data sets.
- The computation delay for these structures may contain large multiplicative constant factors.
- Although synopses are useful for both databases and data streams, in the context of data streams, they need to handle additional issues such as item expiration in sliding windows case, more strict limits on running time, etc.

Next, we introduce different types of data sets and data streams as well as possible types of queries over such data sources. Finally, we shortly discuss the most important type of synopses and few general summarization techniques.

6.1 One-Pass Data Sets and Data Streaming Models

Based on the application type on data sets and data streams may be defined via different models. Note that since for both massive data sets and data streams we need one-pass algorithms, we may use the term *data streams* for both of them in this study. Moreover, all of these models have some features in common; i) the sequence of data is too long that it can not be stored completely in the memory (or even if it is stored it cannot be processed in online fashion), and ii) the results of the queries need to be continuously re-evaluated on the fly. In this section, we list the existing data streaming models in the literature of the data streaming systems.

- **Add-Only Data Streams:** This is the simplest data streaming model, in which data items or tuples will be added to the rest of data set. No update or deletion is considered for this model.
- **Data Streams with Windows:** Sliding window is a more commonly used model to handling data expiration. In these models, only a chunk of data items from the most recent part of the data stream is considered [LMT05].
  - **Physical and Logical Windows:** Two main types of sliding windows are considered commonly in the literature; *Physical* windows which contains the most recent N tuples for some N, and *Logical* or *Time-based* windows which contains those tuples that have arrived in last T units of time for some T. There is another type of windows which is useful for data sets with various data item size. In such environments, it is impossible to pick an appropriate N or T that works for all the situations. To resolve this issue, a good solution is to use a fixed-size window in terms of space. That is the most recent set of data items that in aggregate occupies a certain amount of space is considered in the current window.
  - **Sliding and Tumbling Windows:** Another important issue regarding the

sliding window model is the way the window should slide. The simplest method is to slide the window every time a new tupple arrives. However this approach is not very time-efficient. A better alternative is to slide once a certain number of items, say S, arrive. This new portion of data stream is called a *slide*[1]. When the size of the slides are the same as window size, we say the window is tumbling.

  – **Updatable vs Fixed Tuples:** Another variation of this model is allowing for the updates on data items in the middle of the sliding windows. We should mention that usually these updates are ignored for the sake of better performance, however in some applications this may not be possible due to the high dynamicity of the data set.

- **General Update Model:** In the *general update model*, data stream is defined as a sequence of records in the form of (ts, i, Δ), where ts indicates the time-stamp or position of the data item in the sequence, i is the items index from FIXED domain $[n] = 1, 2, ..., n$, and Δ is an integer number indicating the change in the frequency of item i [GM07].
- **Strict Update Model:** This model is a special case of the general update model in which the frequency of items are non-negative [GM07].
- **Insert-Only Model:** In this model no deletion is allowed. That is Δ can not be a negative number [GM07].
- **Textual Data Streams:** Those data streams in which the data is a text are referred to as *Textual Data Streams* or *Topic Streams* [Kle02, HP10].

[1] A similar idea is being used in logical windows.

6.2 Essential Queries in Massive Data Sets and Data Streams

Queries can be categorized into different classes: First, queries may be one-time or continuous. Queries in data streams are mostly continuous, while they are often of one-time type in databases [TGN92][BW01]. One-time queries are those needing to be evaluated once. This type of queries can be blocking; that is to report the results they need to see the entire data set [BW01]. Second important distinction is between predefined queries and ad hoc (online) queries [BBD02]. Online queries are generally harder to deal with, since the query optimizer does not know anything about the nature of the query in advance. Additionally, in data streams, to answer ad hoc queries one may need to access part of data that have previously been arrived. This contradicts the one-pass processing concept in data streaming environments.

In many applications, queries are biased toward some points of interest in the data distributions, usually the tails. For example, in analyzing round trip time (RTT) of network packets the queries are biased toward the longer RRTs. However, in many other applications queries interest is uniformly distributed over the entire data set (data stream). These in fact shows that the nature of the queries plays a crucial rule in the synopses design for the data set. We discuss more on this in Section 8.

With this introduction, we provide a list of the most important type of queries in data streams:

- **Point Query ($\theta(i)$):** Returns an approximation for the frequency of item i in the data stream.
- **Range Query ($\theta(l, r)$):** Returns an approximation for the sum of the frequency of items form index l to index r ($\sum_{i=l}^{r} a_i$).
- **Selectivity Query** ($S(C)$)**:** Computes the number of items having a given condition C [SAC79], [PC84]. Note that this query contains both of the above queries.
- **Inner Product Query ($I(a, b)$):** Returns the inner product of two data streams

a and b with size n ($a \odot b = \sum_{i=1}^{n} a_i b_i$). The inner product is the basic operation for joining data sets or data streams, which is a very costly action. Thus, query optimizers usually need to estimate the cost of each join before they let the join operation starts.

- **Frequency Moments and L_p Norms:** For a given set of frequencies, a_i, the kth frequency, F_k moment is defined as $\sum_i a_i^k$. Obviously, F_0 is the number of distinct items and F_1 is the total number of items [FM85, AMS99]. Norms function has a similar definition. In fact, the L_p norm for $p > 0$ can be defined as $L_p = F_k^{1/p}$.

- **ϕ-Heavy Hitters:** Return the items with frequency of more than the fraction ϕ of the entire data set size. Heavy Hitters may also be referred to as the *Hot List* [GM99, CKM08] or iceberg queries [FSG98].

- **Top-k Queries:** Given set P of d-dimension points in $\Re^d$, top-k query with respect to function $score$ from a point to a real value returns k points with the largest $score$s. Top-k results are useful in many applications such as communications and sensor networks, stock market trading, profile-based marketing, rating/ranking systems, and multi-criterion decision making systems.

- **Skylines:** Given set P of d-dimension points in $\Re^d$, the $skyline$ of P is the set of points in P which are not dominated by any other points in P. Note that point p dominates q iff they are not equal and p values in all the d dimensions are less than or equal to q value in the corresponding dimensions [BKS01], [PTF05], and [CGG03]. Skylines are closely related to top-k queries, and have the same type of applications. Some techniques use skylines in order to maintain top-k results over sliding windows of data streams [MBP06].

- **Frequent Itemsets**: In shopping basket data sets, the set of items that have been appeared in more than a certain number of tuples is called a *frequent itemset*. Frequent itemsets can also be used to mine association rules [AS94], and [SA96].

- **Other Aggregate Function**: Max/Min, Entropy, Mean, Variance, Median, k-Median, etc. [HHW97, FKS99, SBA04].

Note that many other aggregate functions such as *Distinct Counts*, *Maximum/Mininimum*, *Entropy*, *Mean*, *Variance*, *Median*, *k-Median*, etc. can also be considered as query types. However since they can usually be estimated using above queries, we do not consider them in our list of queries. For more information on these sorts of functions, readers are referred to [HHW97], [FKS99] and [SBA04].

6.3 Existing Synopses

To approximately estimate the answers for the previously mentioned queries, the most common way is to employ an appropriate sketch or synopsis. In this section, we provide a list of most commonly used synopses in the literature.

- **ϕ-Quantiles:** They can provide essential information for many applications such Query optimization in commercial DBMSs, splitters in parallel data bases [PI96], association rule miners in data mining applications [SA96], and data cleaning through similarity checks [DJM02].
- **Equi-Width Histograms:** Equi-Width Histograms are the traditional histograms, in which the domain interval is divided into B equal-sized intervals and the number of items in each of those intervals are computed.
- **Equi-Depth Histograms:** Equi-depth histogram computation is akin to quantile computation. More specifically, an equi-depth histogram tries to partition the ordered list of the input data set into into B buckets so that the number of items in the buckets are (roughly) the same. [PC84]
- **Compressed Histograms:** Compressed Histograms are very similar to equi-depth Histograms. The only difference is that in compressed histograms items with very high frequency (for example, those that their frequency is larger than

the bucket size) are considered in singleton buckets and an equi-depth histogram will be used to summarize the rest of the data streams.

- **V-Optimal Histograms:** V-Optimal histograms try to approximate the ordered list of input with a B-step Function [JKM98, GKS01].
- **MaxDiff Histograms:** MaxDiff histograms [PIH96] aim to find the $B-1$ largest gaps (boundaries) in the sorted list of input.
- **Wavelets:** Wavelets try to decompose the given data set into a series of coefficients using the trends in data. This way, the most insignificant coefficients can be ignored to make an approximate sketch of the whole data set. Perhaps the most important issue is that in order to be able to answer many types of queries, the data should be re-composed from the coefficients which is not very time-efficient. Essentially, maintaining the set of significant transform values in wavelet generation is similar to that for top-k queries. However, depending on the transformation type it may be significantly harder than top-k queries[GKM03].
- **Other Counting Sketches:** There are several other sketches that are also useful to estimate the frequency of items. These sketches can be also used in more complex sketches. Exponential Histograms [DGI02a], AMS [AMS96], Smooth Histograms [BO07], and FP-Trees [HPY00] are a few example of these sketches. We will discuss them in more details later in this dissertation.

6.3.1 Quantiles and Equi-Depth Histograms

Equi-Depth Histograms, [Gre96][MD88] (also known as equi-height or equi-probable) seek to specify boundaries between buckets such that the number of tuples in each bucket is the same. Histograms of this type are more effective than equi-width histograms, particularly for data sets with skewed distributions [Ioa03].

The equi-depth histogram problem is obviously akin to that of quantiles [GK01b], which seeks to identify the item that occupies a given position in a sorted list of N

items: Thus given a ϕ, $0 \leq \phi \leq 1$, which describes the scaled rank of an item in the list, the quantile algorithm must return the $\lceil \phi N \rceil$'s item in the list. For example, a 0.5-quantile is simply the median. Therefore, to compute the $B - 1$ boundaries of any equi-depth histograms, we could employ a quantile structure and report ϕ-quantiles for $\phi = \frac{1}{B}, \frac{2}{B}, ..., \frac{B-1}{B}$. However, this solution suffers from the following problems:

- Quantile computation algorithms over sliding windows are too slow, since they must derive more information than is needed to built the histogram [PIH96].
- Quantiles algorithms that are used to construct histograms focus primarily on minimizing the rank error, while in equi-depth histograms we need to focus on the size of individual buckets and minimize the bucket size error.

Compressed histograms [PIH96] are a variation of equi-depth histograms, that place the highest frequency values in singleton buckets and use equi-depth histogram for the rest of input data. This type of histograms usually provide more accurate results and can also be used to construct biased histograms [CKM06]. However, we should also note that compressed histograms are computationally harder to design than equi-depth histograms, since they require to keep track of frequent data items.

There are many works on designing quantiles in databases [PC84], [GKM02], [GKM02]. Most of these structures pre-compute the quantiles and try to maintaining it once any pieces of data is manipulated. A commonly used technique in some of these approaches is to recompute the quantiles once in while as well.

In 1984, Shapiro and Connel introduced a method to estimate the selectivity of conditions in form of $attribute\ \theta\ constant$ in a database system where θ can be one of $=$, $<$, $>$, $\leq$, and $\geq$ [PC84]. Obviously this is very akin to the concepts of quantiles and equi-depth histograms.

In 2002, Gilbert *et. al.* used *Random Subset Sums* (RSSs) as an sketch to store summarized information about the whole database and estimate the quantile with a one-pass algorithm [GKM02].

Gibbons *et al.* presented a sampling-based technique for maintaining the approximate equi-depth histograms on relational databases [GMP97]. This work is of our interest mainly because (i) it has considered a stream of updates over database, which may make the approach applicable for the data stream environment as well, and (ii) it has employed a split/merge technique to keep the histograms up-to-date, which was inspiring for us. However in their work, the stream of updates is taken from a Zipfian distribution which means some records may be updated several times, while many of the other records remain unchanged; this is not the case in a data streaming environment where records enter windows once and leave in the same order in which they arrived.

Most past work, in the area of data streams, has focused on the related problem of quantiles. It has been proven [MP80] that single-pass algorithms for computing exact quantiles must store all the data; thus, the goal of previous researches was to find approximate quantile algorithms that have low space complexity (i.e., low memory requirements). For this reason, Manku *et al.* introduced an ϵ-approximate algorithm to answer any quantile query over the entire history of the data stream using $O(\frac{1}{\epsilon}log^2(\epsilon N))$ space [MRL98]. Later, they improved their quantile computation for the case of not knowing the data set size in advance using a non-uniform random sampling [MRL99]. Greenwald and Khanna, in [GK01b], improved the memory usage of the previously mentioned approach to $O(\frac{1}{\epsilon}log(\epsilon N))$, and also removed the restriction of knowing the size of the stream (N) in advance. Their work has been influential, and we refer to it as the GK algorithm. For instance, GK was used in [LLX04] and [AM04] to answer quantile queries over sliding windows. The algorithm proposed in [LLX04] has space complexity $O(\frac{1}{\epsilon^2}log^2(\epsilon W))$, where W is the window size and it is unknown a priori. In [AM04], Arasu and Manku proposed an algorithm which needs $O(\frac{1}{\epsilon}polylog(\frac{1}{\epsilon}, W))$ space. We will refer to this algorithm as AM.

AM runs several copies of GK over the incoming data stream at different levels. At these levels, the data stream is divided into blocks of size $\epsilon W/4$, $\epsilon W/2$, ϵW, etc. Once GK computes the results for a block in a level, AM stores these results until that block expires. In this way, they can easily manage the expiration in the sliding window. To report the final quantile at each time, AM combines the sketch of the unexpired largest blocks covering the entire window. Similar to AM, the algorithm proposed in [LLX04] also run several copies of the GK algorithm. Therefore, both of them suffer from higher time complexity, as was shown in [ZW07b]. In fact, Zhang *et al.* also proposed a faster approach for computing ϵ-approximate quantile using $O(\frac{1}{\epsilon}log^2\epsilon N)$ space. However, their approach works only for histograms that are computed on the complete history of the data stream, rather than on a sliding window.

Gibbons *et. al.*, in [GMP97], also provided an algorithm to maintain approximation for compressed histograms in the dynamic scenario in which items can be updated after they entered to the system.

In 2002, Qiao *et. al.* provided a new algorithm to construct a compressed histogram, called *Relaxed Histogram* or RHist, to answer multiple queries over the entire history of a data stream [QAA02]. Their algorithm dynamically adapts accordingly with changes in queries as well as with the new data items come into the system. The basic idea to adapt to new-coming queries is to used more buckets in the ranges where more queries are expects to occur. The have also used a *Workload Decay Model* to give less to older queries since new coming queries are more significant in their model.

6.3.2 Biased Histograms and Quantiles

In many environments queries are biased toward some points in data distribution usually on one or both ends of the data distributions. As an example, consider a performance monitoring system in a networking environment, which is watching the round trip time (RTT) for TCP packets. Note that RTT delays can be stretched a lot in different situations, so the distribution of RTTs is very skewed at its tail [CKM05][CKM06]. In such system, the queries are biased to the high end of the RTTs distribution. Another good example of these skewed data sets can be the distribution of people's wealth [CKM06].

There is tiny portion of people at the high end of this distribution, which attracts all the queries. As the third example, we can mention the distribution of the number of incoming (outgoing) URL in the Web's graph.

Many optimization techniques need to know biased information about the distribution of these data sets in order to perform more efficient and effective. For example, in the Web's graph, we may need a good approximation for the average in-degree of nodes with ranks between 99.9% and 99.99% as well as that for nodes with ranks between 99% and 99.9%. This may be because of that to store information for the nodes located between positions 99% and 99.9% and nodes in between position 99.9% and 99.99%, same amount of memory may be needed. This is very crucial in efficiently splitting/distributing large data sets over different servers.

This type of applications leaded researched toward the definition of biased quantiles/hostograms [CKM05, CKM06, ZW07b]. For instance, a biased quantile can consist of set of the median, 75%, 87.5%, 93.75%, ... quantiles. In other words, the interest is biased to a particular point (or points) in the distribution (usually one of the ends or both). Although the traditional quantiles are able to answer any quantile queries, they do not provide more accurate results for the points of interest in the distribution. For example, a 0.01-approximate quantile would return the same results for all ϕ-quantiles, $99\% \leq \phi < 100\%$. This is not acceptable in the biased quantiles' application. One may say that generating more accurate quantiles would alleviate this issue, but the problem is that the existing quantile computation algorithms are not only too slow, but also, their required space is proportional to $\frac{1}{\epsilon}$ or $\frac{1}{\epsilon^2}$. Moreover, for the mentioned applications there is no need to have the same approximation for every given quantile query.

To the authors' knowledge, no one has considered the problem constructing biased histograms in data streams over sliding windows. All the works in this area have considered the entire history of the data streams [CKM05, CKM06, ZW07b]. In Chapter 8, we present a new randomized algorithm called *Bar-Splitting Biased Histograms* (BSBH) to compute biased histograms over sliding windows of data streams. We also prove that under practical assumptions and slow concept shifts the algorithm can guarantee the expected ϵ-approximate results.

Cormode *et. al.* have introduced the idea of biased quantile in [CKM05]. Based on the GK algorithm, they proposed a deterministic ϵ-approximate algorithm to construct biased quantiles over the whole history of a data stream. Their approach needs $O(\frac{B \times log1/\phi}{\epsilon} log(\epsilon N))$ of space, where N is the current size of the stream and B is the number of boundaries we need to report. Note that as shown in [ZLX06], the worst case behavior of this type of algorithms are linear in the universe size U (The size of the items' domain).

Later, in [CKM06], the same authors proposed a faster and more space-efficient deterministic algorithm for the same problem, based on a binary tree structure idea borrowed from [SBA04]. Needing $O(\frac{logU}{\epsilon} log\epsilon N)$ space and an almost constant amortized cost of actions per new entry, the algorithm is best suited for high speed data streams. However, it can not be easily used for the sliding windows data streaming model. Moreover, their algorithm needs the prior knowledge of U and the bound on space requirement of the algorithm is dependent on the U which is not desirable.

Zhang and Wang have used a decomposable structure to construct ϵ-approximate biased quantiles using $O(\frac{log^3 \epsilon N}{\epsilon})$ space and $O(log(\frac{log\epsilon N}{\epsilon}))$ of time. In their paper, a naive non-uniform sampling technique is used which works as follows; first the given data set of known size N is sorted, and data items in each interval $[\frac{N}{2^{i+1}}, \frac{N}{2^i})$ is randomly sampled with rate $\frac{\epsilon N}{2^i}$ for $i = 0, 1, 2, ..., logN$. It is easy to see that the resulting sample is simply an ϵ-approximate biased quantile. The rest of their idea is partitioning the data stream into blocks of known size, using the introduced non-uniform sampling over each block, and finally merging the results derived for all the blocks into a final answer.

Unfortunately, the mentioned biased quantile computation algorithms can not be easily used for the sliding window case, due to their underneath structures which basically do not support the idea of expirations. The usual solution for this issue is to run several copies of the same algorithms for different-size chunks of the most recent part of the current window, and try to combine the results for the biggest possible parts which are not expired yet. Similar technique is used in [AM04] for regular quantiles. However, the technique is proven to provide a slow algorithms [ZW07b][MZ11b].

6.3.3 Exponential Histogram Sketch

In [DGI02b], Datar *et al.* proposed the Exponential Histograms (EH) sketch algorithm for approximating the number of 1's in sliding windows of a 0-1 stream and showed that for a δ-approximation of the number of 1's in the current window, the algorithm needs $O(\frac{1}{\delta} log W)$ space, where W is the window size. The EH sketch consists of an ordered list of buckets. In this paper, we refer to these buckets as boxes to avoid confusion since we will use the term *bucket* at another level in our approach. Every box in an EH sketch basically carries on two types of information; a time interval and the number of observed 1's in that interval. We refer to the latter as the size of the box. The intervals for different boxes do not overlap and every 1 in the current window should be counted in exactly one of the boxes. Boxes are sorted based on the start time of their intervals. Here are the brief descriptions for the main operations on this sketch:

Inserting a new 1: When at time t_i a new 1 arrives, EH creates a new box with size one, sets its interval to $[t_i, t_i]$, and adds the box to the head of the list. Then the algorithm checks if the number of boxes with size one exceeds $k/2 + 2$ (where $k = \frac{1}{\delta}$), and, if so, merges the oldest two such boxes. The merge operation adds up the size of the boxes and merges their intervals. Likewise for every $i > 0$: whenever the number of boxes with size 2^i exceeds $k/2 + 1$, the oldest two such boxes are merged. Figure 6.1 illustrates how this operation works.

Expiration: The algorithm expires the last box when its interval no longer overlaps with the current window. This means that at any time, we only have one box that may contain information about some of the already expired tuples. The third row in Figure 6.1 shows an expiration scenario.

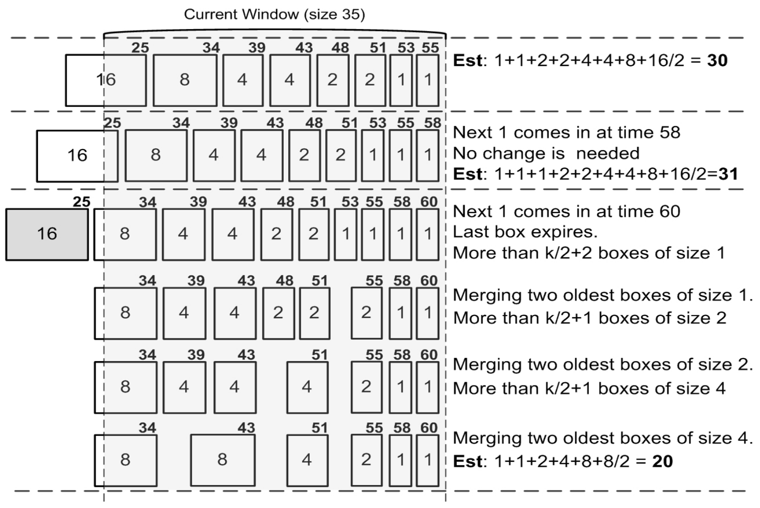


Figure 6.1: Incrementing an EH sketch twice at time 58 and 60. That is we have seen 1, 0, and 1 respectively at time 58, 59, and 60. (k=2 and W=35)

Count Estimation: To estimate the number of 1's, EH sums up the size of all the boxes except the oldest one, and adds half the size of the oldest box to the sum. We refer to this estimation as the count or size of the EH sketch.

It is easy to show that using only $O(\frac{1}{\delta}logW)$ space, the aforementioned estimation always gives us a δ-approximate number of 1's. This approach is quite fast too, since the amortized number of merges for each new 1 is only $O(1)$. It is also worth noting that instead of the counting the number of 1's in a 0-1 stream, one can simply count the number of values falling within a given interval for a general stream. For instance, to construct an equi-width histogram, a copy of this sketch can be used to estimate the number of items in each interval, when the boundaries are fixed.

6.3.4 Other Types of Histograms

Perhaps, the simplest type of histograms are the traditional *equi-width* histograms, in which the input value range is subdivided into intervals (buckets) having the same width, and then the count of items in each bucket is reported. Knowing the minimum and maximum values of the data, the equi-width histograms are the easiest to implement both in databases and in data streams. However, for many practical applications, such as fitting a distribution function or optimizing queries, equi-width histograms may not provide useful enough information [Gre96]. A better choice for these applications is an *Equi-depth* histogram which has been discussed earlier in previous section.

Other types of histograms proposed in the literature include the following: (i) *V-Optimal* Histograms [GKS01][JKM98] that estimate the ordered input as a step-function (or pairwise linear function) with a specific number of steps, and (ii) *MaxDiff* histograms [PIH96] which aim to find the $B-1$ largest gaps (boundaries) in the sorted list of input. Although these type of histograms can be more accurate than the other histograms, they are more expensive to construct and update incrementally [HKY09]. For example, current V-optimal algorithms perform multiple passes on the database, and are not suitable for most data stream applications [JKM98].

6.3.5 Other Synopses

As Already mentioned in Section 6.3.3, Datar *et. al.* presented an effective method, called *Exponential Histograms* to estimate statistics of data streams over sliding windows [DGI02a]. They showed in order to provide an ϵ-approximate results their structure needs only $O(\frac{1}{\epsilon}logW)$ of space. More detail on this method is provided in section 6.3.3. Later on, many other works have used EH sketch as their basis to provide other types of statistics [BO07], [BO10], and [MZ11b]. We already talked about works in [MZ11b], in which a fast and space efficient equi-depth histogram is provided using EH sketch.

Braverman *et. al.* in [BO07] and [BO10] improved the approximation error rate of EH sketch. Their sketch, *Smooth Histogram*, is able to provide approximate results for a larger class of function than EH called (α, β)-smooth function. These functions are namely L_p norms ($p \neq 1, 2$), frequency moments, length of increasing subsequence and geometric mean. Function f is called (α, β)-smooth, if, once $f(B)$ is a β-approximate of $f(A)$ where data set B is the suffix of A, we can guarantee that $f(B \bigcup C)$ is an

α-approximation of $f(A \bigcup C)$ for any portion of new elements in data set C.

A very costly operation in DBMSs and also in DSMSs is the join (or inner product) operation. To make join operations faster, the order of tables or data streams that are being joined is very important. Thus, an important job of query optimizers is to estimate the selectivity of each join.

FM sketch is a simple structure to compute the number of distinct items (0's frequency moment) form domain $\{0, 1, ..., M-1\}$ [FM85]. Assume hash function *h(x)* that maps incoming values, say x, in $\{0, 1, ..., M-1\}$ uniformly across $\{0, 1, 2, 4, ..., 2^{L-1}\}$, where $L = O(logM)$. Now, assume $lsb(x)$ denote the position of the least-significant 1 bit in the binary representation of x. FM sketch is an array A of L bits initialized by 0's. For each incoming value x, we set $A[lsb(h(x))]$ to 1. It is easy to show that at each moment in time the number of distinct items is approximately 2^l, where l is the position of the left most 0 in array A.

In [MBP06], authors used a k-skyline structure in order to maintain the results of top-k queries over sliding windows of a data stream. Since k-skyline is k layers of skylines over a given data set, the results of the top-k query must be among the items of these skylines. Thus it is enough to start searching the layers from the outmost one, and if top-k results are larger than the maximum value in the next layer, we can stop. Otherwise, we should search the next layer and so forth.

6.4 Sampling as the Most Common Summarization Technique

Sampling is the process of choosing a data item to be processed or not with some probability [Vit85b], [GZK05]. Sampling methods are one the most popular summarization techniques due to their low cost of generation and compatibility with most of the existing functions and applications. We usually are interested in *random sampling*, in which numbers are taken from the same probability distribution, without involving any real population.

Sampling techniques are harder to implement when the size of data streams are unknown prior to the execution of the algorithms [GZK05]; however, this is not a vital issue for case of the data streams with sliding windows. Perhaps, a more important issue in data streams with sliding windows is to avoid periodic sampling and keep the sample random. Another important consideration in sampling methods is their ability to efficiently deal with a fluctuating data rate [GZK05].

In many streaming environments with a bursty and a fast arriving data-stream, due to the existing of bottlenecks at some nodes, seldom a sequence of incoming input has to be shed. The processes of skipping a sequence of data items usually in a data streaming environment is referred to as *Load Shedding* [MBD03, MZ10]. The process of load shedding may also be referred to as *Skip Sampling* in some literature, since we sample chunk of data all at once. In addition to having the same problems as other sampling techniques, load shedding is usually difficult to be used with data mining algorithm due to dropping chunk of data items [GZK05].

Sketching techniques may also try to vertically sample the input data stream in the literature. Considering data items have several attributes or features, sketching samples those attributes which are of the most importance in some sense [BBD02], [Mut03], [GZK05].

Uniform random sampling is a very well-studied topic for data sets with known size. There are several approaches to construct Uniform random samples. However, for the cases that the data sets size are unknown in advance or the case of sliding windows over data streams, the problem becomes more challenging [Vit85b]. This is usually due to online nature of these data sets in which one-pass algorithms are the only acceptable option. One of the earliest works to address the issue of not knowing the data set size in advance is [Vit85b] by Vitter. He proposed an intuitive algorithm called *Algorithm R* which is a simple type of *Reservoir* algorithms. For generating a sample of size n in Algorithm R, first n items of the data set are selected as the initial sample and for each item number t ($t > n$) with probability n/t it will be randomly replaced by one of the already sampled items. This guarantees that at any moment in time we have an n-item uniform sample of the so far seen data set. In the same paper, Vitter proposed a faster sampling algorithm which is referred to as *Algorithm Z*. In this algorithm instead of flipping a coin for each item, we skip a random number of items and then replace the next new item to the Reservoir.

In 1998, Gibbons and Matias introduced two new sampling based summary statistics called *concise Sampling* and *counting sampling, GibbonsMatias98*. The former one is simply a uniform random sample of input such that values occurring more than once in the sample are represented with a tuple of $< value, count >$. On the other hand, *counting sampling* counts the total number of occurrences for each value selected to be in the sample. They also showed how these techniques can be used to approximate hot list queries (Heavy Hitters) in a data set.

In [GMP97] and [GM99], a uniform random sampling technique on data sets is proposed which has the capability of coping with the updates in the values of data items. This sampling technique which is called *Backing Sampling* employs a small synopsis to reduce the number of disk accesses. In this technique a reservoir-based sampling such as Algorithm Z, concise, or counter sampling is used as the basis. Once an item in the data set is deleted, two approaches can be followed; 1) the delete operation is postponed until next insert happens, or 2) a larger sample size is maintained so if the deleted item is among the sampled items, it will be replaced with another item in the reservoir.

Most of the discussed sampling techniques are limited due to lack of support for data streaming environments. In the context of sliding windows of data streams, sampling and load-shedding are often considered as duals of each other. In [BDM02], a simple approach is proposed to construct a uniform random sample of size m over the most recent n items of the data set. The whole idea of their approach is that whenever an item is added to the samples reservoir at time t, its future substitute index is also selected among the next n items $t + 1$, $t + 2$, ..., $t + n$. This way, each item in the sampled list is chain to its substitute and when ever the item is expired it would be replaced with its substitute without any further computations. This approach is called *Chain-Sampling*. They also proposed a *priority sampling* for logical window cases. In priority sampling technique each newly arrived data item gets a random priority from 0 to 1, and the item with the highest priority will be sampled and its successor is an item arrived after p and has the highest priority. A similar technique is used in [ADL05] to estimate the subset sum queries.

Aggarwal in [Agg06] introduced a new type of sampling called *biased sampling*. His idea was that in some situations users may need the results from both the entire history of data streams and the most recent items. For these situations, using sliding windows results in losing the older items and using the whole history would not be so efficient. Therefore, he proposed a biased sampling in which the sampled items bias to the more recent items.

Sampling with or without replacement from fixed-size (physical), timestamp-based (logical), or bursty windows are studied by Braverman, Ostrovsky, and Zaniolo in [BOZ07] and [BOZ09]. They also proved that the memory complexity of sampling algorithms over sliding windows is comparable to those of data streams without sliding windows. They basically showed that to generate a random sample when the window slides it is enough to replace the old slide's sampled items with a uniform random sample of the new coming slide.

The authors in [TcZ07] described a way to determine load-shedding ratios in a distributed system given the utility function that maximizes throughput. Their work however does not address important factors such as the ability to load-shed based on a utility function and computing load-shedding plans online which is essential in a data-stream environment.

In [MZ10], the load shedding problem is looked at as an optimization problem in data streaming systems in which multiple queries with specific load shedding requirements are taken into consideration. Three main strategies of load-shedding are studied, namely 1) *Uniform*, where the load is shed equally for every query, 2) *Proportional*, where the load is shed proportionally to the amount of results returned by the query, and finally 3) *Optimal*, where the shedding ratio is proportional to the amount of results returned by the query. Their proposed techniques are applicable to data-mining queries such as K-Means, decision tree classifiers, etc. This work also allows dynamic re-computation of shedding ratios when system resources change.

6.5 Other Summarization Techniques

In addition to employing synopses and sampling to summarize massive data sets or data streams, there are some other general summarization techniques that are shortly discussed next:

Clustering: Another technique which can serve as a summarization technique is *clustering*. In clustering, similar items (in some senses) are grouped together. For some applications, we may only need to store and/or process the groups information, which can greatly save memory and time. Clustering can also be beneficial in the design of more space-efficient sketches. A useful practice, for example, is to construct the sketches over the items of each group instead of the entire data sets. This may greatly reduce the total memory usage. Clustering techniques also play a crucial role in distributed systems, where the data should be partitioned and assigned to different nodes in a way that minimum inter-node communication would be required.

Compression: Although these terms may be used interchangeably in some literatures, there is a subtle difference between them. In compression techniques, the only goal usually is to reduce the size of a data set in a way that it fits in the smaller space. While, in summarization techniques, in addition to this, we need structures that allows for fast (mostly on-line) processing and query answering algorithms. This basically indicates that not all the compression techniques can serve as a summarization technique. Moreover compression is a lossless process, while summarization is not necessarily.

CHAPTER 7

A Fast- and Space-Efficient Equi-Depth Histogram

Equi-depth histograms provide statistically accurate, space-efficient synopses for very large data sets; therefore, they proved invaluable in a wide spectrum of database applications, such as: query optimization, approximate query answering, distribution fitting, parallel database partitioning, and data mining. Histograms are even more important for data streaming applications, where synopses become critical to provide real-time or quasi real-time response on continuous massive streams of bursty data and to minimize the memory required to represent these massive streams. However, finding fast and light algorithms to compute and continuously update histograms represents a difficult research problem, particularly if we seek the ideal solution to compute accurate histograms by performing only one pass over the incoming data. Data streaming applications tend to focus only on the most recent data. These data can be modeled by sliding windows [BBD02], which are often partitioned into panes, or *slides* whereby new additional results from the standing query are returned at the completion of each slide [BBD02][LMT05].

Once the number of buckets (B) is given, the equi-depth histogram algorithm seeks to find $B-1$ boundaries such that the number of tuples between two consecutive boundaries is approximately N/B, where N denotes the number of tuples in the window. For the sake of time and space issues, this goal can only be achieved within a certain approximation, and because of computational constraints and requirements of continuous queries ϵ-approximations with much coarser ϵs are expected in the data stream environment. In particular, we require that only one pass be made over the incoming data and the results must be reported each time the current window slides.

In this section, we study the problem of constructing equi-depth histograms for sliding windows on data streams and propose a new algorithm that achieves average ϵ-approximation. The new algorithm, called BAr Splitting Histogram (BASH), has the following properties: BASH (i) is much faster than current techniques, (ii) does not require prior knowledge of minimum and maximum values, (iii) works for both physical and logical windows in data streams, and (iv) leaves a smaller memory footprint for most cases. As we shall discuss in more detail later, the main idea of BASH builds on the *Exponential Histograms* technique that was used in [DGI02a] to estimate the number of 1's in a 0-1 stream over a sliding window (See Section 6.3.3).

More specifically, we make the following contributions in this section:

- We introduce a new expected ϵ-approximate approach to compute equi-depth histograms over sliding windows.
- We show that the expected memory usage of our approach is bounded by $O(B\frac{1}{\epsilon^2}log(\epsilon^2 W/B))$, where B is the number of buckets in the histograms and W is the average size of the window.
- We present extensive experimental comparisons with existing approaches: the results show that BASH improves speed by more than 4 times while using less memory and providing more accurate histograms.

In order to introduce our novel technique for generating space-efficient Equi-Depth Histograms on data streams with sliding windows, we first provide the formal definitions of equi-depth histograms and error measurements. Later, we explain the algorithm and provide our theoretical and experimental results. This section is first proposed in [MZ11b].

7.1 Definitions

Throughout this section, we will frequently use three important terms: *boxes*, *bars*, and *buckets*. *Buckets* are the final intervals that we intend to report for the histogram. On the other hand, each bucket may contain one or more *bars*, and for each bar, the associated EH sketch has a list of *boxes* which has been described in previous section. With this clarification, we can formally define an equi-depth histogram as follows:

Definition 1 *A B-bucket equi-depth histogram of a given data set D with size N is a sequence of $B-1$ boundaries B_1, B_2, ... $B_{B-1} \in D$, where B_i is the value of the item with rank $\lfloor \frac{i}{B} N \rfloor$ in the ordered list of items in D.*

Note that each data item (tuple) is simply considered as a floating point number. As in can be seen, the number of items between each two consecutive boundaries is the same (N/B). To ease this definition, B_i can be also defined as any values between the value of the items with rank $\lfloor \frac{i}{B} N \rfloor$ and $\lfloor \frac{i}{B} N + 1 \rfloor$. A similar definition can be used for the sliding window case:

Definition 2 *A B-bucket equi-depth histogram of a given data stream D at each window with size W is a sequence of $B-1$ boundaries B_1, B_2, ... $B_{B-1} \in D_W$, where D_W is the set of data in the current window and B_i is the value of the item with rank $\lfloor \frac{i}{B} W \rfloor$ in the ordered list of items in D_W.*

Note that the above definitions is valid for both physical windows in which W is a fixed value and logical windows in which W may vary through the time. However, as already discussed, computing the exact equi-depth histograms is neither time efficient nor memory friendly particularly for the data streaming case [MP80]. Thus, we need algorithms to approximate the histogram with respect to some kind of error measurements. The most natural way to define the error is based on the difference between the ideal bucket size (i.e. (W/B) and the size of constructed buckets by any algorithms.

The other way of computing this error, which is based on the error computation in quantiles, is to compute the expected error for the ranks of reported boundaries. A third approach can also be considered, which uses the differences between the ideal position (actual value) of the bucket-boundaries and the exact boundaries. Based on these three error types, the following definitions can be considered for the expected ϵ-approximate equi-depth histogram. These three error types are respectively called size error, rank error, and boundary error.

Definition 3 *An equi-depth histogram summary of a window of size W is called a size-based expected ϵ-approximate summary when the expected error on the reported number of items in each bucket s_i is bounded by $\epsilon W/B$.*

Definition 4 *An equi-depth histogram summary of a window of size W is called a rank-based expected ϵ-approximate if the expected error of the rank of all the reported boundary r_i is bounded by ϵW.*

Definition 5 *An equi-depth histogram summary of a window of size W is called a boundary-based expected ϵ-approximate if the expected error of all the reported boundary b_i is bounded by ϵS, where S is the difference between the minimum and maximum values in the current window.*

The boundary error behaves very similar to the rank error. However, boundary error is much faster to be computed, since it deals with the value of the items instead of their ranks. Throughout this section, we consider the first definition, because it is more appropriate for many applications that only care about the size of each individual bucket (not the position of the boundaries).

7.2 BAr Splitting Histogram (BASH)

In this section, we describe our BAr Splitting Histogram (BASH) algorithm which computes a size-based expected ϵ-approximate equi-depth histogram. For the rest of the section, W denotes the number of items in the current window. In other words, W is the size of the most current window, whether it is physical or logical. We should also mention that, we never assume that W is a fixed value as in physical windows. This implies that our approach works for both types of sliding windows. As already mentioned, B is the number of buckets in the histogram under construction. The pseudo-code for BASH is given in Algorithm 2, which is comprised of the following phases:

1. In the first phase, BASH initializes the structure, which may differ depending on whether the minimum and maximum values of the data stream is known or not. However, the general idea is to split the interval into at most $S_m = B \times p$ partitions (Bars), and assign an EH structure to each of the intervals ($p > 1$ is an expansion factor helping to provide accuracy guarantees for the algorithm, which will be discussed further in Section 7.4).

2. The next phase seeks to keep the size of each bar moderate. To achieve this, BASH splits bars with sizes greater than a threshold called $maxSize$. Note that $maxSize$ is proportional to current window size W, so it is not a constant value. Next section discusses this threshold with more details. Since the number of bars should not exceed S_m, BASH may need to merge some adjacent bars in order to make more room for the split operation.

3. Using the boundaries of the bars computed in the previous two phases, the third phase estimates the boundaries for the final buckets. This phase could be run each time the window slides or whenever the user needs to see the current results. To implement this phase, one could use a dynamic programming approach to find the optimum solution, but unfortunately this is not affordable under the time constraint in streaming environments, so we have to employ a simpler algorithm.

We will next describe these three phases in greater detail.

Algorithm 2 BASH()

```
BASH () {
  $initialize()$;
  while (true) {
    $next$ = next item in the data stream;
    find appropriate bar $bar$ for $next$;
    insert $next$ into $bar.EH$;
    $maxSize$ = $\lceil maxCoef \times W/S_m \rceil$;
    if ($bar.count > maxSize$)
      $splitBar(bar)$;
    remove expired boxes from all EHs;
    if (window slides)
      output $computeBoundaries()$;
  }
}
```

7.3 Bars Initialization

In many cases, the minimum and maximum possible values for the tuples in the stream are known or can be estimated. To initialize the structure for such cases, the $initialize()$ function in Algorithm 2 partitions the interval between the minimum and the maximum into $S_m = B \times p$ ($p > 1$) equal segments, that we call bars; an empty EH sketch is created for each segment. As it will be shown later, the value of our expansion factor does not need to be much larger than one. In practice, our experiments show that it is sufficient to set p value less than 10.

For those cases where there is no prior knowledge of the minimum and maximum values, $initialize()$ starts with a single bar with an empty EH. Once the first input, say $next$, comes in, the method sets the interval for this bar to $[next, next]$, and increments its EH by one. The algorithm keeps splitting the bars and expanding the interval of the first and last bars as more data are received from the stream. In this way, we can also preserve the minimum and maximum values of the tuples at each point in time.

The other critical step in Algorithm 2 is the split operation $splitBar()$. This operation is triggered when the size of a bar is larger than $maxSize = \lceil maxCoef \times W/S_m \rceil$, where $maxCoef$ is a constant factor. Obviously, $maxCoef$ should be greater than 1 since the ideal size of each bar is W/S_m and we need the $maxSize$ to be larger than W/S_m. On the other hand, after splitting a bar into two bars, the size of each bar should be less than or equal to the ideal size of bars. This implies that $maxCoef \leq 2$. This value is empirically determined to be approximately 1.7. Smaller values usually cause more splits and affects the performance, while larger values affect the accuracy since larger bars are more likely to be generated. Observe that as the window size changes, $maxSize$ also changes dynamically. This in turn implies that BASH starts splitting bars very early, when the window is empty and first starts to grow, because at that point $maxSize$ is also quite small. Starting the process of splitting the bars early is crucial to assure that accurate results are obtained from early on in the execution of the algorithm. This dynamicity also helps BASH to easily adapt to the changes in window size W for logical (i.e., time-based) windows.

Example 1: Assume that we want to compute a 3-Bucket histogram ($p = 2$, $k = 2$, and $W = 100$) and the data stream starts with 10, 123, 15, 98, ... Thus, the initialization phase starts with a single bar $B_1 = (1, [10, 10])$, which means the bar size and its interval are respectively 1 and $[10, 10]$. Once second data item, 123, enters, BASH inserts it into B_1, so $B_1 = (2, [10, 123])$. However, at this point B_1's size is larger than $maxSize$ ($\lceil 1.7 \times 2/6 \rceil$) and it should be split, so we will have two bars $B_1 = (1, [10, 66])$ and $B_2 = (1, [66, 123])$. Next data item, 15, goes to B_1, so $B_1 = (2, [10, 66])$. Again, B_1's size is larger than $maxSize$ ($\lceil 1.7 \times 3/6 \rceil$), so BASH splits it and now we have three bars: $B_1 = (1, [10, 38])$, $B_2 = (1, [38, 66])$ and $B_3 = (1, [66, 123])$. By repeating this approach for each incoming input, we obtain a sequence of non-overlapping bars covering the entire interval between the current minimum and maximum values in the window.

7.3.1 Merging and Splitting Operation

As already mentioned, for each new incoming tuple, the algorithm finds the corresponding bar and increments the EH sketch associated with that bar as, explained in Chapter 6. However, the most important part of the algorithm is to keep the number of tuples in each bar below $maxSize$, allowing for a more accurate estimate of the final buckets. In order to do so, every time the size of a bar gets larger than $maxSize$, BASH splits it into two smaller bars. However, we might have to merge two other bars in order to keep the total number of bars less than S_m. This is necessary to control the memory usage of the algorithm. Due to the structure of the algorithms, it is easier to start with the merging method.

Merging

In order to merge the EH sketches of the pair of selected bars, the EH with the smaller number of boxes is set to *blocked*. BASH stops incrementing blocked bars, but continues removing expired boxes from them. In addition, the interval of the other bar in the pair is merged with the interval of the blocked bar, and all the new tuples for this merged interval are inserted into this bar. Algorithm 3 describes this in greater detail. The details on how the bars are chosen for merging will be discussed later in this section.

When running Algorithm 3, every bar may have one or more blocked bars attached to it. BASH keeps checking that the size of the actual bar is always bigger than those of its blocked bars, and whenever this is no longer the case, BASH switches the actual bar with the blocked bar of larger size. Although this case is rare, it can occur when the boxes of the actual bar expire and the bar length decreases, while the blocked EH has not been changed.

Also, observe that every bar might have more than one blocked bar associated with it. Consider the case where we have to merge two adjacent bars which already have an associated blocked bar. To merge these two, we follow the same general approach: the longest EH is picked and all other bars (blocked or not) are considered as the blocked bars of the selected (actual) bar. In the next section, we will show that the expected number of such blocked bars is S_m.

Algorithm 3 mergeBars()

```
if (! findBarsToMerge($X_l$, $X_r$)) return false;
Initialize new bar $X$;
if ($EH_{X_l}$.BoxNo $\geq$ $EH_{X_r}$.BoxNo){
  $EH_X$ = $EH_{X_r}$;
  Add $EH_{X_l}$ to the blocked bars list of $EH_X$;
} else {
  $EH_X$ = $EH_{X_l}$;
  Add $EH_{X_r}$ to the blocked bars list of $EH_X$;
}
Add the blocked bars of both $EH_{X_l}$
  and $EH_{X_r}$ to the blocked bars list of $EH_X$;
$X$.start = $X_l$.start;
$X$.end = $X_r$.end;
Remove $X_r$ and $X_l$ from the bars list;
Add bars $X$ and to the bars list;
return true;
```

Splitting

When the size of a bar, say X, exceeds the maximum threshold $maxSize$ as shown in Algorithm 2, we split it into two smaller bars according to Algorithm 4. Before the split operation takes place, we make sure that the number of bars is less than S_m. If this is not the case, as already explained, we would try to merge two other small adjacent bars. After X is determined to be appropriate for splitting, we divide the interval being covered by X into a pair of intervals and assign a new EH sketch to each of them. Let us call these new bars X_l and X_r. X must be split in such a way that the size (count) of X_l and X_r remain equal. To this end, the algorithm first tries to distribute the blocked bars of X, denoted as BB_X, into the blocked bars of each of the new bars. The splitting algorithms tries to do this distribution as evenly as possible; however, the size (count) of bars X_l and X_r may not be equal after running this step. Thus, BASH splits the actual bar of X (EH_X) accordingly to compensate for this difference. It is easy to show that it is always possible to compensate for this difference using an appropriate split on EH_X, since the size of the actual bar is guaranteed to be greater than the size of each its blocked bars.

To split the original EH sketch, EH_X, into a pair of sketches, BASH first computes the split ratio denoted as $Ratio$ in Algorithm 4. $Ratio$ simply indicates that to compensate for the difference between the size of the two new bars, the aggregate size of boxes going from EH_X to EH_{X_r} after splitting should be $Ratio$ times the size of EH_X. In order to have such a split ratio, BASH puts half of EH_X boxes with size one into EH_{X_l}, and the other half into EH_{X_r}. Then, it replaces each of the remaining boxes in EH_X with two boxes that are half the size of the original box. Finally, based on the current ratio of the sizes, BASH decides whether to put each copy to EH_{X_r} or EH_{X_l}. In other words, if the current ratio of the size of EH_{X_r} to the aggregate size of EH_{X_r} and EH_{X_l} is smaller than $Ratio$ the copy will go to EH_{X_r} otherwise it will go to EH_{X_l}.

Example 2: Let us get back to our running example. Assume we have the following six bars at some point: B_1=$(21, [1, 43])$, B_2=$(14, [43, 59])$, $B_3 = (10, [59, 113])$, B_4=$(9, [113, 120])$, B_5 =$(29, [120, 216])$, and $B_6 = (15, [216, 233])$, and the next input is 178. BASH inserts this input into B_5, and now B_5's size is 30 which is larger than $maxSize$=$\lceil 1.7 \times 100/6 \rceil$=29. Thus, B_5 should be split. However, since we already have six ($p \times B$) bars, we need to find two candidates for merging before we can split B_5. As the bars' size offers B_3 and B_4 are the best candidates, since after merging them the aggregate size is still smaller than $maxSize$. Therefore, after this phase is done the following bars would be in the system (note that for simplicity, we assumed that nothing expires in this phase): B_1=$(21, [1, 43])$, B_2=$(14, [43, 59])$, B_3=$(19, [59, 120])$, B_4=$(15, [120, 168])$, B_5= $(15, [168, 216])$, and B_6=$(15, [216, 233])$.

Which Bars to Merge

To select two bars for merging, we first look for any two empty adjacent bars. If there is no such pair, we look for an empty bar and merge it with the smaller of its

Algorithm 4 splitBars(X)

```
if ($curBarNo$ == $S_m$ && !$mergeBars()$) return;
Initialize new bars $X_l$ and $X_r$;
$l = 0$;
//distributing blocked bars of $X$ between the new bars
$BBSize$ = Aggregate Size of the blocked bars;
for each (blocked bar $bar$ in $BB_X$) {
  if ($l$ + $bar$.size ¡ $BBSize$/2) {
    $l$ += $bar$.size;
    Add $bar$ to $BB_{X_l}$;
  } else
    Add $bar$ to $BB_{X_r}$;
}
$EH_{X_l}$.start = $EH_X$.start;
$EH_{X_l}$.end = ($EH_X$.start + $EH_X$.end)/2;
$EH_{X_r}$.start = ($EH_X$.start + $EH_X$.end)/2;
$EH_{X_r}$.end = $EH_X$.end;
$Ratio$ = (($X$.size$/2$)-$l$)$/$($X$.size - $BBSize$);
foreach (box $box$ in $EH_X$) {
   if ($box$.size == 1)
     Alternatively add a copy of $box$ to $EH_{X_l}$ or $EH_{X_r}$;
   else {
     $box$.size = $box$.size/2;
     for (2 times)
          if ($EH_{X_r}$.size $/$($EH_{X_r}$.size+$EH_{X_l}$.size) ¡ $Ratio$)
            Add a copy of $box$ to $EH_{X_r}$;
          else
            Add a copy of $box$ to $EH_{X_l}$;
   }
}
Remove $X$ from the bars list;
Add bars $X_l$ and $X_r$ to the bars list;
```

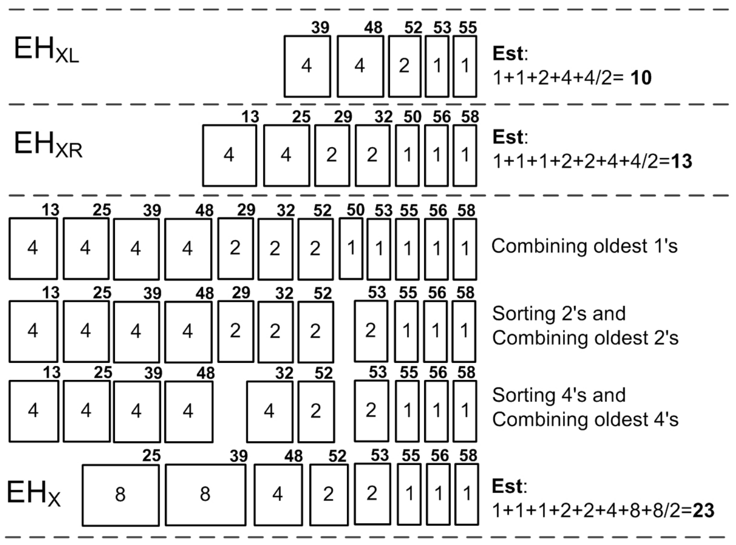


Figure 7.1: Merging two EH sketches EH1 and EH2. (k=2)

two neighbors. When there is no empty bar, then we find the two adjacent bars that have the minimum aggregate size. If this aggregate size is less than $maxSize$ (which is usually the case), we select them for merging. Otherwise, we do not perform any merge operations until some boxes expire, since we do not want to create bars with size greater than $maxSize$. Although bars may be initially empty or become empty due to the expiration of their boxes, the case in which the bars are not is obviously the most common one.

An Alternative Merging Approach

Although the aforementioned approach guarantees the approximation ratio, it is using blocked bars which may increase the memory requirement and as a result influence performance. As an alternative approach, one can use the following method, wherein the idea is to merge the boxes of the two bars, and make a new EH. The former technique is called BASH-BL, since it needs to deal with BLocked bars, and this alternative approach is referred to as BASH-AL.

To merge two EH sketches, we start by selecting boxes with size one from both EH sketches, and mingle them into a list sorted by the start time of their intervals. If the number of such boxes is more than $k/2 + 2$ ($k = 1/\delta$), we keep combining the oldest boxes in this list until the number of boxes with size one is smaller than $k/2 + 2$. The same steps would be followed for boxes with size two. We compile all boxes of size two, sort them based on their start times, and if the number of these boxes is more than $k/2 + 1$, we combine the oldest ones until the number of such boxes is smaller than or equal to $k/2 + 1$. Note that while merging boxes with size two, we may be using some boxes from the first EH, some from the second EH, and some boxes generated from combining boxes in the previous iterations. This approach is then repeated for boxes with larger size (4, 8, etc.). Figure 7.1 illustrates an example of merging two EH sketches where $k = 2$ using this alternative method.

Although, we do not have any blocked bars, the same split approach, mentioned earlier, can be used for this alternative merging technique. This time, the splitting operation in Algorithm 4 splits the only existing EH into two EHs of equal size.

7.3.2 Computing Final Buckets

So far, we showed how the intervals can be partitioned into S_m parts in a way that each part contains no more than $maxSize$ items. The last phase is to report the final buckets or boundaries of the buckets. As already noted, a dynamic programming approach cannot be used to see which bars should go into each bucket. This is due to the quadratic delay of the dynamic approach that makes it completely inapplicable. Instead, we use a linear approach as shown in Algorithm 5.

First, this algorithm computes the total number of estimated tuples in the current window by adding up the estimated number of tuples in each bar. Let us call it $total_{Est}$, which is not necessarily equal to W[1]. Therefore, the expected number of tuples in each bucket should be $idealBuckSize = total_{Est}/B$. Thus, we start from the first bar and keep summing the estimated number of items in each bar until the sum exceeds $idealBuckSize$. At this point, we report the appropriate boundary for the current bucket as shown in Algorithm 5 and continue with the next bucket.

[1]Note that the expected value of $total_{Est}$ is W, but at each point in time these two may have different values.

Algorithm 5 computeBoundaries()

```
$b = -1$; //An index over bars list
$count = 0$;
$idealBuckSize$ = $total_{Est}/B$;
for (int $i = 0$; $i < B$; $i$++) {
  while ($count \leq idealBuckSize$) {
      $b$ ++;
      $count$ += $bars[b].count$;
  }
  $surplus$ = $count - idealBuckSize$;
  $boundaries[i] = bars[b].start+$
    $bars[b].length * (1 - surplus/bars[b].count)$
  $count$ = $surplus$;
}
return $boundaries$;
```

7.4 Formal Analysis

In this section, we provide the theoretical proof of the approximation ratio, time complexity, and space usage of the BASH algorithm. All the proofs are provided for the version of BASH which uses blocking to merge the bars (BASH-BL). Similar proofs can be provided for the other version as well.

7.4.1 Approximation Analysis

First, we start by proving that the splitting operation does not change the expected error of the estimations for the number of tuples in each new bar. Assume bar X is going to be split. Let the actual number of items in X be N_X, while the sketch has estimated this number to be n_X. Because of the EH sketch approximation ratio, we know:

$$(1-\delta)N_X \leq n_X \leq (1+\delta)N_X \tag{7.1}$$

After splitting X into two smaller bars, say X_l and X_r, our estimation for each of these new bars is $n_X/2$, because of the way the split operation works. Let the actual number of items in X_l be k_l. The expected error for this new bar (X_l) is computed as the average error over all possible values for k_l. Let $P\{k_l = k\}$ be the probability that the size of bar X_l is k ($0 \leq k \leq N_X$). It is then easy to see that:

$$P\{k_l = k\} = \binom{N_X}{k}(1/2)^{N_X} \tag{7.2}$$

Without loss of generality, assume $n_x \leq N_X$ and N_X is an even number. Thus:

$$\begin{aligned} E\{err(X_l)\} &= \sum_{k=0}^{N_X} |\frac{n_X}{2} - k| P\{k_l = k\} \\ &= \sum_{k=0}^{n_X/2} (\frac{n_X}{2} - k) P\{k_l = k\} + \sum_{k=n_X/2+1}^{N_X} (k - \frac{n_X}{2}) P\{k_l = k\} \\ &= 2\sum_{k=0}^{n_X/2} (\frac{n_X}{2} - k) P\{k_l = k\} + \sum_{k=0}^{N_X} (k - \frac{n_X}{2}) P\{k_l = k\} \\ &= 2\sum_{k=0}^{N_X/2} (\frac{n_X}{2} - k) P\{k_l = k\} + \frac{N_X}{2} - \frac{n_X}{2} \end{aligned} \tag{3}$$

Now, we show that the first part of the last equation is negligible with respect to $N_X/2 - n_X/2$.

$$
\begin{aligned}
\sum_{k=0}^{n_X/2}(\frac{n_X}{2}-k)P\{k_l=k\} &\le \sum_{k=0}^{n_X/2}(\frac{N_X}{2}-k)P\{k_l=k\} \\
&\le \sum_{k=0}^{N_X/2}(\frac{N_X}{2}-k)P\{k_l=k\} \\
&= \frac{N_X}{2}\sum_{k=0}^{N_X/2}P\{k_l=k\}-\sum_{k=0}^{N_X/2}kP\{k_l=k\} \\
&= \frac{N_X}{2}\frac{1}{2}(1+P\{k_l=\frac{N_X}{2}\})-\sum_{k=1}^{N_X/2}\frac{N_X}{2}P'\{k_l=k-1\} \\
&= \frac{N_X}{4}(1+P\{k_l=\frac{N_X}{2}\})-\frac{N_X}{2}\sum_{k=0}^{N_X/2-1}P'\{k_l=k\} \\
&= \frac{N_X}{4}(1+P\{k_l=\frac{N_X}{2}\})-\frac{N_X}{2}\frac{1}{2} \\
&= \frac{N_X}{4}\binom{N_X}{N_X/2}(\frac{1}{2})^{N_X}\sim\frac{1}{4}\sqrt{\frac{N_X}{\pi}}
\end{aligned}
\tag{4}
$$

The last part is because of the direct application of Sterling's formula and $P'\{k_l = k\}$ is the probability of having k items in the left bar when you split a bar with size $N_X - 1$. Combining inequality (4) with equation (3), and using inequality (7.1), we can conclude that:

$$
\begin{aligned}
E\{err(X_l)\} &= \frac{N_X}{2}-\frac{n_X}{2}+\frac{2}{4}\sqrt{\frac{N_X}{\pi}} \\
&\le \delta\frac{N_X}{2}+\frac{1}{2}\sqrt{\frac{N_X}{\pi}} \\
&\le (\delta+\frac{1}{\sqrt{\pi N_X}})\frac{N_X}{2}
\end{aligned}
\tag{5}
$$

Note that the second part of this error shows that no matter how accurate the estimation of the size of bar X is, when we split it, we will add $1/\sqrt{\pi N_X}$ error rate on average. Fortunately in practice, this increase in error is negligible since N_X is very large. Moreover, later we show that the number of split operations is bounded such that at the next split on the same bar, this extra error will be vanished due to the expiration of old and inaccurate boxes. Thus in practice, the expected δ-approximation still holds for this new bar. This also holds for $E\{err(X_r)\}$ symmetrically. Thus, we can state that the expected error does not change by splitting a bar. Moreover, the merge operation does not change this error, since the EH sketch of the merged bars does not change at all. These two facts lead us to the following lemma:

Lemma 1 *At any moment in time, our sketch contains an expected δ-approximation of the number of tuples in each bar. Also, these estimations are not greater than $maxSize$.*

Theorem 1 *For any given $0 < \epsilon < 1$, algorithm computeBoundaries() provides a size-based expected ϵ-approximate equi-depth histogram for sliding windows on data streams.*

Proof: We need to show that the estimated number of tuples in each bucket is bounded by $\epsilon W/B$ on average. Remember that, on average every bucket consists of $S_m/B = p$ bars. According to Lemma 1, every bar also has an expected δ-approximate number of tuples in it. On the other hand, the first and last bars should be treated differently, because we may only consider a fraction of them in the bucket. Thus, in total, we would have δ error for the size estimation of the bars and $2 \times 1/p$ for the bars at the two ends of the bucket. The latter is because the entire counted number from those two bars may be on the wrong side of the boundaries we have selected. This proves that the expected total error is bounded by $\delta + 2/p$. Calling this bound ϵ, one can say that by setting $\delta = \epsilon/2$ ($k = 2/\epsilon$) and $p = 4/\epsilon$, we obtain a size-based expected ϵ-approximate equi-depth histogram. $\square$

The proof, we provided here shows that the expected approximation error is bounded, which is enough for our purpose even though better bounds may hold in theory.

7.4.2 Space Complexity Analysis

To analyze the space usage of our approach, we first prove the following lemma showing that on average, the total number of blocked bars is limited.

Lemma 2 *At any moment in time, the BASH algorithm on average generates $O(S_m)$ number of blocked bars.*

Proof: The algorithm only performs a merge operation when it has to split a bar and there is no room left for a new bar. Thus, the number of merge operations is less than the number of split operations. Moreover, for each split operation on a bar, $maxSize/2$ tuples must have been inserted into the bar since the last split on the bar. This means that the average number of splits and consequently the average number of merges is $2 \times W/maxSize$. By selecting $maxSize$ to be $O(W/S_m)$, e.g., $maxCoef \times W/S_m$ which is less than twice the ideal size for each bar, we can conclude that the average number of split operation and as a result the average number of merge operations in each window is $O(S_m)$.

To show that the average number of blocked bars in the current window is $O(S_m)$, first consider that there is no switch between any actual bar and one of its larger blocked bars. For this case, every blocked bar stays in the system, until the whole window expires. This is because when a bar is blocked, it does not get incremented anymore and after the window slides for W tuples, all its boxes will be expired. This basically means the average number of blocked bars is the same as the average number of splits. Now, consider the case that we need to switch an actual bar with one of its blocked bars which has a larger size. We show that on average the number of such switches are constant, thus the average number of blocked bars is still $O(S_m)$. Let q be the probability of switching an actual bar with one of its blocked bars at any time between two consecutive split operations over that actual bar. Also, observe that the size of blocked bars are always decreasing, and on average, the rate of expiring from both types of bars are the same. Thus, blocked bars sizes are decreasing at a faster rate than those of actual bars, since we may insert new items into the actual bars. This basically means $q < 1/2$. Keeping this in mind, we also know that in the mentioned period, the probability of having one switch is q, the probability of having two switches is q^2, etc. Thus, the expected number of switches is $\Sigma i q^i \leq q/(1-q)^2$, which in turn is smaller than 2 for $q < 1/2$. This completes our proof. □

Based on the above lemma, we can now compute the average space used by BASH. On the one hand, each EH sketch for counting X items in an sliding window with δ-approximate ($\delta = \epsilon/2$) accuracy needs $O(\frac{1}{\delta} log(\delta X))$ or equivalently $O(\frac{1}{\epsilon} log(\epsilon X))$ of space. On the other hand, the total number of bars in the system is $O(S_m)$ for the worst case (Lemma 2). Thus the total memory used is $O(S_m \frac{1}{\epsilon} log(\epsilon \times maxSize))$ or $O(B \frac{1}{\epsilon^2} log(\epsilon^2 W/B))$. This leads us to the following theorem:

Theorem 2 *The BASH algorithm computes the set of $B-1$ boundaries of the expected ϵ-approximate equi-depth histogram on a data stream using $O(B \frac{1}{\epsilon^2} log(\epsilon^2 W/B))$ space, where W is the sliding window size.*

Observe that this bound is computed for the worst case scenario. As previously mentioned, we do not need to set k $(1/\delta))$ and p to very high values in practice. Consequently, BASH practically needs even less memory space than the existing approaches.

7.4.3 Time Complexity Analysis

We will compute the expected computational cost of BASH per tuple assuming that $k = 2$: for larger k's the algorithm is faster at the cost of using more memory space. When the next data value, $next$, comes in, the cost of the various steps in Algorithm 2 can be estimated as follows:

- Finding the appropriate bar, $curBar$, for $next$ to be inserted into takes $log(S_m)$ using a simple binary search.
- Incrementing the EH sketch of $curBar$ needs one box-combining operation on

the average. To see why this is $O(1)$, observe that at most half of the increments cause two size-one boxes to get merged. From this half, half of them result in combining two boxes of size 2, and so on and so forth. Therefore, in total, for 2^i increments we need 2^{i-1} merges on boxes of size 1, 2^{i-2} of merges on box of size 2, ..., and 1 merge for boxes of size 2^{i-1}. Since this adds up to $2^i - 1$, each increment requires an average $(2^i - 1)/2^i \approx 1$ combining operations.

- The probability of needing a split operation is $1/S_m$, and to split a bar we only need to go through the box list of EH once; this takes $O(log(maxSize))$. Therefore, the average cost of a split operation is $O(log(maxSize))$ /S_m, which is practically a constant.
- The merge operation itself needs a constant amount of time, but to find the best bars to merge, we need to go through the entire list of bars of length S_m. Multiplying this by the probability of needing to merge ($1/S_m$), we conclude that the average cost for this operation is also constant.
- Computing the final boundaries requires visiting all bars ($O(S_m)$). Fortunately, we do not need to report the results for each new incoming data value. Instead, we can report the results each time the window slides (S): this reduces the complexity to $O(S_m/S)$.
- The naive implementation of the expiration phase checks the last box of each EH ($O(S_m)$). This time can be reduced to $O(log(S_m))$ using a simple heap tree structure. To keep the implementation simple, we perform the expiration phase a constant number of times in each slide: this can lower the time complexity to $O(S_m/S)$.

Considering these computations, we can conclude that the total time complexity per data item would be $O(log(S_m) + S_m/S)$ = $O(log(B \times p) + (B \times p)/S)$, where S is the slide size. In practice, this time complexity would be very close to a constant time, since S is usually much larger than S_m, and $log(S_m)$ is usually very small.

7.5 Experimental Results

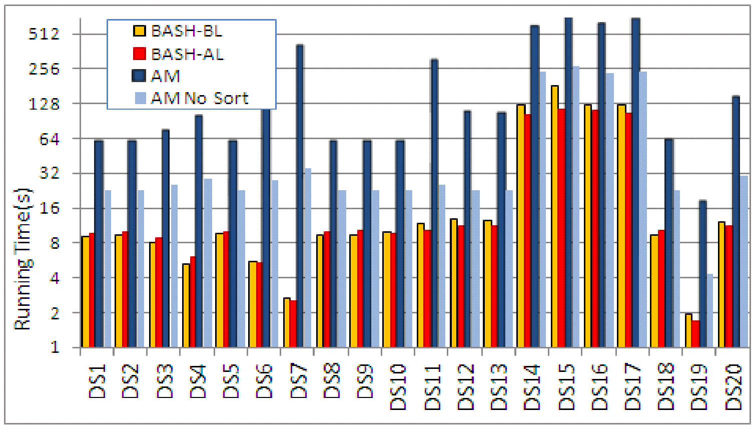
In order to evaluate the performance of the proposed algorithms, we have implemented both versions of the BASH algorithm using the C++ programming language. The version of BASH which blocks the bars in the merge operation is referred to as BASH-BL, and the alternative approach which mixes the boxes of the EHs into a single EH at the merge time is called BASH-AL. In addition to BASH, we have also implemented the AM [AM04] algorithm in the exact same environment. To be able to compare the resulting histograms from each of the mentioned algorithms, we used a simple approach which computes the exact boundaries of the equi-depth histograms. It is worthy to note that for some cases this approach needs several hours to compute the exact histograms.

For all the experiments shown in this section, we have fixed the slide size at 100 tuples. To Make the error diagrams smoother, the average error for every 20000 tuples is shown at each point. No prior knowledge of the minimum and maximum values of the incoming data is considered. $maxSize$ and B are also set to $1.7 \times W/S_m$ and 20 respectively. Similar results hold for different values of B, which are eliminated due to space limitations.[2]

Most of the comparisons included in this part are performed with $\epsilon = 0.01$ for AM algorithms. The experimental results show that to obtain this error rate for window sizes larger than 100k, it is almost always sufficient to set δ to 0.1 and p to 7. Moreover, if one wants to compete with AM with an error rate of 0.025, setting δ to $1/8$ and p to 4 would always provide more accurate results in BASH than AM. All the experiments are run on a 64bit, 2.27 GHz machine running CentOS with 4GB of main memory (RAM) and 8MB of cache.

Table 7.1: Data sets used for the experimental results

Name	**Distribution**	**Size**	**Shifts**	**Parameters**
DS1	Uniform	$1m$	0	$min0, max10k$
DS2	Normal	$1m$	0	$\mu5k$, $\sigma2k$
DS3	Normal	$1m$	0	$\mu5k$, $\sigma50$
DS4	Normal	$1m$	0	$\mu5k$, $\sigma20$
DS5	Normal	$1m$	0	$\mu5k$, $\sigma500$
DS6	Zipfian	$1m$	0	$\alpha1.1$
DS7	Zipfian	$1m$	0	$\alpha1.5$
DS8	Exponential	$1m$	0	$\lambda10^{-2}$
DS9	Exponential	$1m$	0	$\lambda10^{-3}$
DS10	Exponential	$1m$	0	$\lambda10^{-4}$
DS11	Normal	$1m$	100	$\mu10k$-$1k$, $\sigma50$
DS12	Normal	$1m$	100	$\mu15k$-$5k$, $\sigma200$
DS13	Normal	$1m$	$1k$	$\mu15k$-$5k$, $\sigma200$
DS14	Normal	$10m$	$1k$	$\mu15k$-$5k$, $\sigma500$
DS15	Normal	$10m$	$1k$	$\mu15k$-$5k$, $\sigma50$
DS16	Mix	$10m$	100	$\mu5k$, $\sigma200$, $\lambda10^{-3}$
DS17	Mix	$10m$	$1k$	$\mu5k$, $\sigma200$, $\lambda10^{-3}$
DS18	Poker-Hands	$10m$	0	-
DS19	S&P500	$165k$	-	-
DS20	Expanded S&P500	$1.03m$	-	-



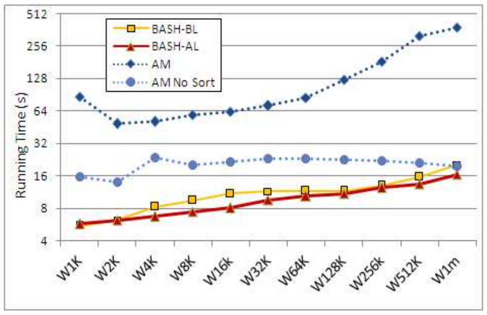


Figure 7.2: a) Execution time for all data sets ($W = 100k$), and b) The effect of changing window size from 1k to 1m on the execution time on DS13. ($k = 10$, $p = 7$, and $\epsilon = 0.01$.)

7.5.1 Data Sets

The data sets used for evaluating the results can be categorized into two main parts: synthesized data sets and real-world data sets. Table 7.1 summarizes some information about these data sets.

Synthesized data sets: We have used a number of different synthesized data sets in our experiments:

- Uniform, Normal, Zipfian [Zip49], and Exponential distributions with different settings for their parameters and without any concept shift (DS1 through DS10).
- Normal distributions with many shifts on the value of the mean parameter. (DS11 to DS15)
- Random shifts on random distributions (DS16 and DS17): Just as in the previous data sets, the distribution of data in these data sets encounters several shifts. However, this time at each shift, we randomly select between normal and exponential distributions. If the choice is the normal distribution, with probability 0.5, we increase the mean by 50, otherwise we decrease it by 50. In the same way, we change the standard deviation by ± 5. On the other hand, if the exponential distribution is selected at the current shift, the rate parameter (λ) is randomly updated by $\pm 10\%$. The initial values of the parameters are shown in Table 7.1.

Real-world data set: One of the most important applications of the histograms is for those cases in which the distribution of the data is unknown or can not be simply modeled. To evaluate the performance of BASH algorithms over this kind of data, we have considered the following real-world data sets:

- *Poker-hands (DS18)[web10b]*: Each row in this data set contains 10 values ranging from 1 to 12. For this data set, we have computed the histogram over the multiplication of all these values in each row. Note that this data set contains no

[2]For more results, see our more extensive report in [MZ11b].

concept shift, since the probability of each outcome at each point is computable.

- *S&P500 (DS19)[web10a]*: This data set contains minute-by-minute prices of the S&P500 index starting from January 29, 2010 and ending on August 13, 2010. Each record contains the highest, lowest and last prices of the index in the corresponding minute.
- *Expanded S&P500 (DS20)*: For each row of the previous data set, we generated 17 more values with a normal distribution such that 95% of them are between the lowest and highest prices of the corresponding row. Combined with the existing three prices in each row, this makes a data set with more than 1.03 million records.

7.5.2 BASH Timing Results

For each of the data sets introduced in Table 7.1, the running time of the three algorithms with $\epsilon = 0.01$ and $W = 100k$ is shown in part a of Figure 7.2. Similar results are obtained for different window sizes and ϵ which is not shown in this paper due to space limitations. For further results, readers are referred to our extensive report [MZ11b].

Based on the timing results in part a of Figure 7.2, we can state that both BASH-BL and BASH-AL are at least four times faster than AM in almost all the data sets, while also providing more accurate histograms which are discussed later in this section. This improvement on execution time makes BASH much more applicable than AM. particularly for larger windows (or equivalently faster data streams.) As an example, consider DS16 in which AM and BASH-AL (BASH-BL) respectively spend 607.56 and 111.79 (124.61) seconds to compute the results. Another interpretation of these timings would be that AM and BASH-AL (BASH-BL) can respectively support 16549 and 89453 (80250) data items per second with a window size of $100k$. Therefore, AM may not be practical for many data streaming systems, while BASH is performing properly. In the next few paragraphs, we show that the problem of high delay of the AM approach gets even worse with larger window sizes.

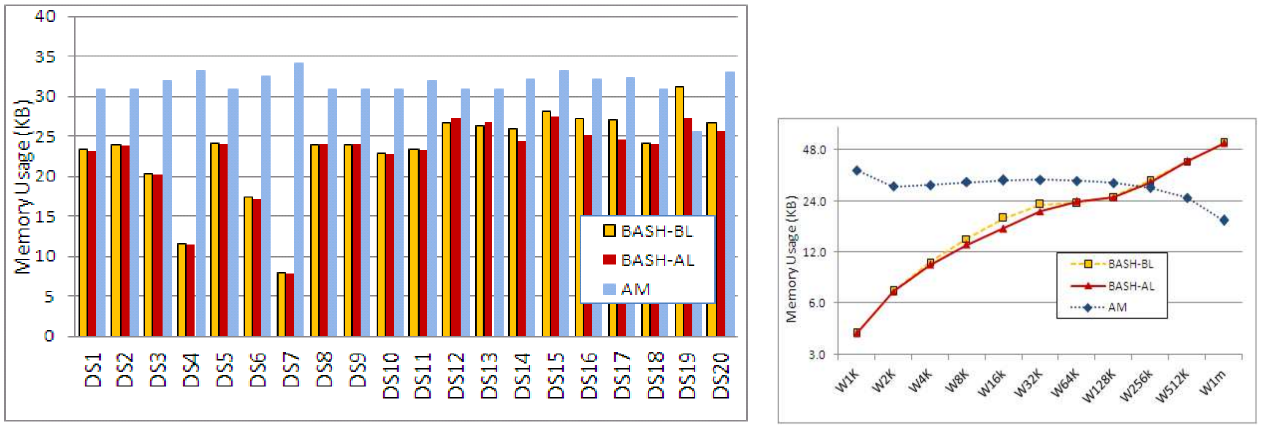


Figure 7.3: Space usage of the algorithms for a) all the data sets ($W = 100k$), and for b) different window sizes on DS13. ($k = 10$, $p = 7$, and $\epsilon = 0.01$.)

As already mentioned, AM runs several copies of the GK algorithm and employs a sorting algorithm over the output of these copies to generate the final quantiles. Thus, the timing performance of AM is hugely dependent on its sorting algorithm. In our implementation of AM, we have used quick sort, since it is one of the fastest sorting algorithms in practice. However, we have also included the running time of AM without considering the time it spends on sorting (bars titled as ”AM No Sort” in part a of Figure 7.2). Interestingly, even after eliminating the sorting time from AM, both BASH-BL and BASH-AL are still faster. Another important factor affecting the performance of the mentioned algorithms is the window size W_t. Part b of Figure 7.2 sketches how the window size affects the response time. These results are taken over data set DS13. The delay time for all the three approaches increases with respect to the window size; however, the growth rate of AM is greater than the other two. It is worthy to mention that putting the sorting time aside, the rest of the AM algorithm still needs more time than BASH algorithms for all window sizes.

7.5.3 BASH Space Usage Results

The diagram in part a of Figure 7.3 compares the space usage of the three algorithms for a window size of $W = 100k$. In this diagram, for each of the data sets, we have provided the size of the main structures in the algorithms. For all the cases (except DS19), both BASH-BL and BASH-AL need at least 20% less space than AM. However, BASH-BL uses slightly more space than BASH-AL. The reason that BASH algorithms need more memory than AM for DS19 is the small size of this data set (165K) with respect to the window size (100K). This is due to the fact that the BASH algorithms are in their initialization phase, during which reaching convergence might use more memory than other algorithms. It is also worth mentioning that the smaller the variance of data values, the less the memory usage for both BASH algorithms.

In part b of Figure 7.3, the diagram illustrates the space usage of the three algorithms for different window sizes. Both BASH algorithms use almost the same amount of memory in each window size, and that amount increases logarithmically with respect to the window size. On the other hand, memory usage of AM is almost constant for different window sizes, and it even decreases for very large window sizes. Therefore, for the window sizes larger than 256K, AM starts performing better than BASH algorithms in terms of space. This suggests that there are situations where AM performs better than BASHs with respect to memory usage. However, we also need to keep in mind the following two important points: First, as shown in the two diagrams of Figure 7.4, the histograms generated by either of the BASH algorithms over windows with size larger than 256K are at least three times more accurate than the ones computed by AM. Second, for all the cases that the memory usage of BASH algorithms is worse than AM, the running time of AM is at least 8 times worse than those of BASHs. Nevertheless, for larger window sizes, we can use smaller values for parameters k and p, and reduce the memory usage while still providing faster and more accurate histograms than AM. The current setting of k and p values is best suited for window sizes between 32K to 256K in terms of both accuracy and space requirements.

7.5.4 BASH Error Results

To evaluate the accuracy of each approach, we have used three type of errors which have been already introduced in Section 7.1: The boundary error, bucket size error (or simply size error), and rank error. For each type of errors the absolute difference of the reported value by each algorithm with the ideal value is used to avoid domination of the final error values by the large differences. The average error for each of these three error types over time are shown in Figure 7.4. Part a, b, and c of the figure respectively compare the boundary, size, and rank errors of the three algorithms. As you can see, both BASH-BL and BASH-AL algorithms outperforms AM in terms of accuracy for most of the data sets. It is also worthy to mention that although not always true, BASH-BL is slightly more accurate than BASH-AL.

The aforementioned error results are for the case in which window size is $100k$. However, this error results differ for different window sizes. To see this, we compute the boundary, size and rank errors of the histograms generated by each of the algorithms over data set DS13 (Figure 7.5). All three algorithms perform almost equally in terms of errors up to the window size of 16k. However, from window size of 32k, the error rates of BASH algorithms start to decrease, while for the AM algorithm, this rate does not change, or it even increases. As a result, in larger window sizes, both BASH-BL and BASH-AL hugely outperform AM algorithms with respect to the three error types.

To have a better understanding of the evolution of accuracy for the histograms over time, we also compare the boundary error results for AM, BASH-BL, and BASH-AL for some of the data sets. We should mention that the errors of different diagrams are not comparable with each others since they are in different scales:

Data Sets with No Concept Shifts: Figure 7.6 compares the boundary errors for the three approaches (AM, BASH-BL, and BASH-AL) for uniform, normal, and exponential distribution. The X axis shows the number of tuples that have come into the system thus far, while the Y axis indicates the average boundary errors. Both BASH

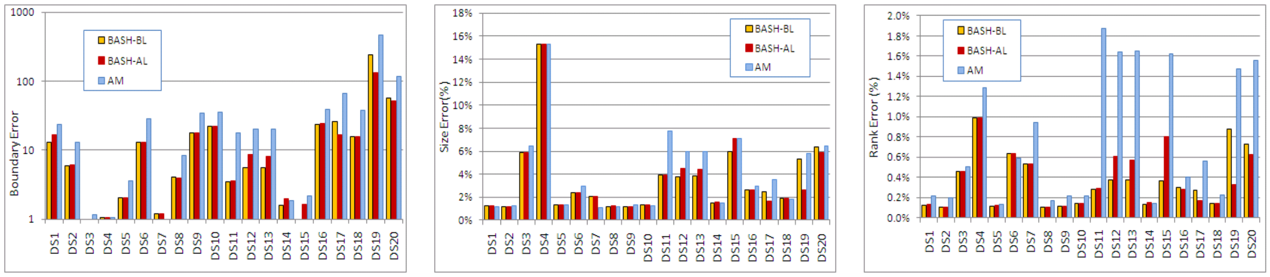


Figure 7.4: a) Boundary error, b) size error, and c) rank error for all data sets. ($k = 10$, $p = 7$, $W = 100k$, and $\epsilon = 0.01$.)

algorithms perform nearly the same, and at all points in time the generated histograms by the BASH algorithm are more accurate than the ones generated by the AM method.

Data Sets with Concept Shifts: Figure 7.7 illustrates the boundary errors of AM and two versions of the BASH algorithm on three of the data sets with concept shifts. The results mainly indicate that, not only are BASH-BL and BASH-AL on average more accurate than AM, but they are also more steady. This is more apparent in part a of Figure 7.7 that sketches the boundary error results for data set DS14 in which we have a concept shift at every 10,000 data items. To see this fluctuation in the errors computed for the AM method, all three diagrams in Figure 7.7 show the error of the generated histograms from tuple 4000k to tuple 6000k. In all these three data sets, the boundary error for the AM method fluctuates at each concept shift while BASH is more stable.

Real-World Data Sets: The boundary errors for the three real-world data sets are also computed for BASH-BL, BASH-AL, and AM algorithms (Figure 7.8). Similar to most of the cases we have seen thus far, BASH algorithms are far more accurate than AM for all the three data sets.

7.5.5 Histograms vs Sampling

As already mentioned, sampling is an easy technique to be combined with other approaches, since it just decreases the volume of data and the rest of computations re-

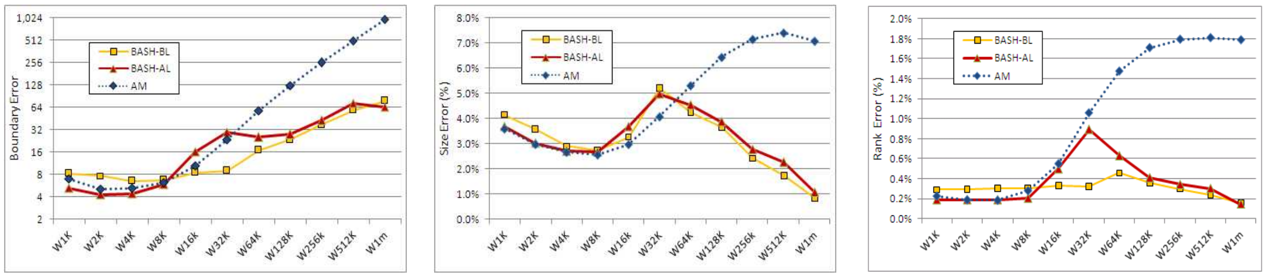


Figure 7.5: The effect of changing window size from 1k to 1m on the a) boundary error, b) size error, and c) rank error for DS13. ($k = 10$, $p = 7$, and $\epsilon = 0.01$.)

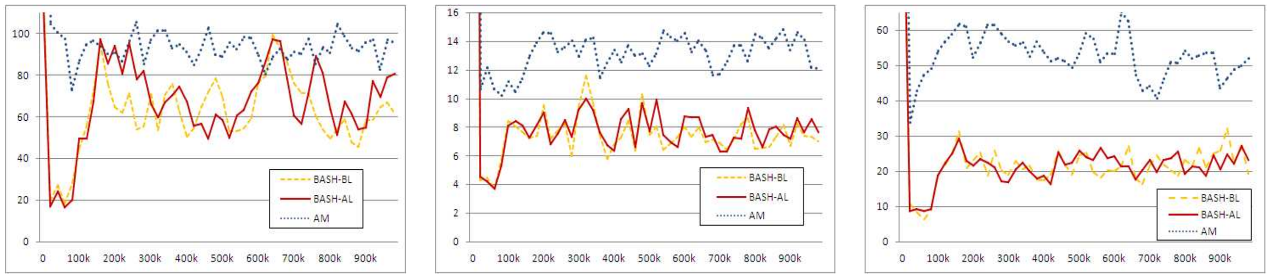


Figure 7.6: The boundary errors for data sets a) DS1, b) DS5, and c) DS8 from left to right. ($k = 10$, $p = 7$, $W = 100k$, and $\epsilon = 0.01$.)

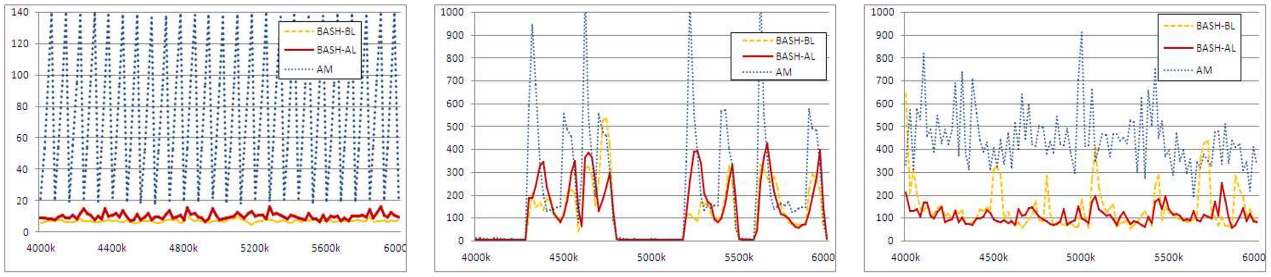


Figure 7.7: The boundary errors for data sets a) DS14, b) DS16, and c) DS17 from left to right. ($k = 10$, $p = 7$, $W = 1m$, and $\epsilon = 0.01$.)

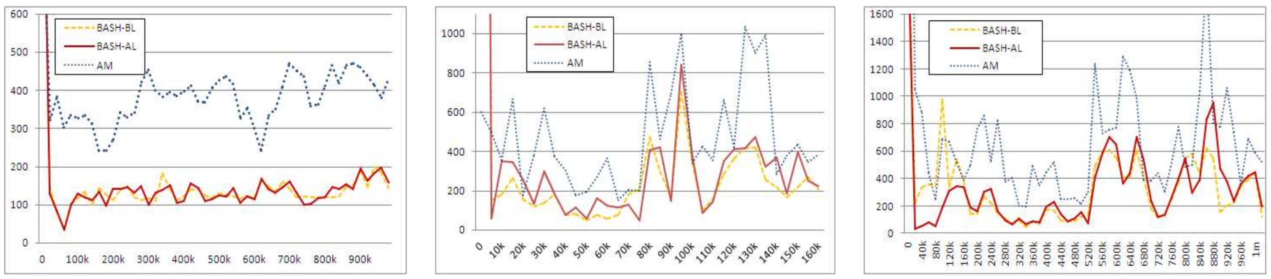


Figure 7.8: The boundary errors for data sets a) DS18, b) DS19, and c) DS20 from left to right. W_t is set to $10k$ for DS19 and $100k$ for others. ($k = 10$, $p = 7$, and $\epsilon = 0.01$.)

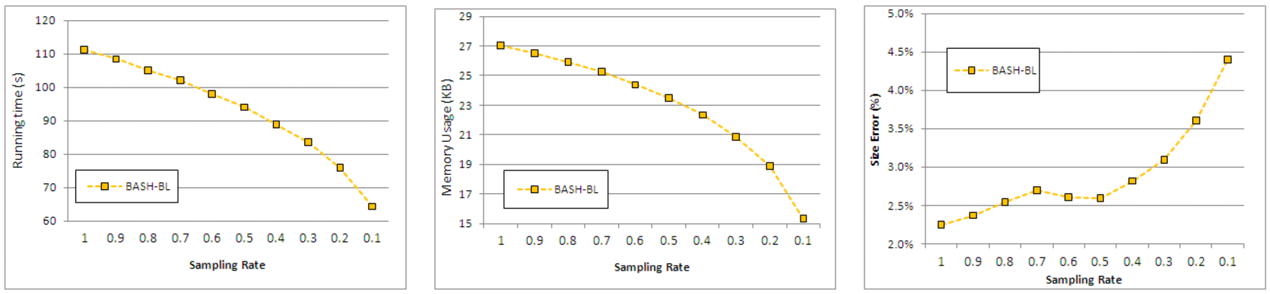


Figure 7.9: The effect of combining sampling and BASH with different sampling rates on a) running time , b) memory usage, and c) size error from left to right. The results are for data set DS17. ($W = 100k$, $k = 10$, and $p = 7$.)

mains unchanged. This way, the algorithms may be run faster using less memory with the cost of degrading the results accuracy. To see the effect sampling over histograms performance and accuracy, we picked one of our data sets (DS17) and computed the histograms, using BASH-BL algorithm, over the sampled data with different rates as it is shown in Figure 7.9. Note that to use sampling with rate $smpRate \leq 1$ in BASH, we gradually decrease the sample rate from 1 to $smpRate$ through first sliding window. This helps initiation phase to converge sooner.

As it is expected, sampling improves the performance of BASH algorithm. For example sampling with rate 10% reduces time and memory usage of BASH by 42.2% and 43.2% respectively. However, the accuracy results, in this case, are degraded with higher rate (48.8%). This may indicate that sampling over BASH algorithm can not reach a good tradeoff between efficiency and accuracy. Nevertheless, under strict time/memory constraints sampling is still an acceptable option.

Another type of sampling that we discussed in this prospectus is biased sampling. As Section 8.1.3, biased sampling rate μ should be greater than $\phi^{1/p}$ where ϕ is the histograms biased rate. For higher sampling rates, one should set $\mu = \phi^{1/p}$ and use flat sampling. Since this particular biased sampling rate makes all the bars to have the same size, BSBH will be identical to BASH and the above results should hold for this case. Now, the remaining question is what is the best value for μ. To answer this question we ran at the middle of running experiment to analyze the performance of BSBH with different biased sampling rate.

7.5.6 Discussion

The experimental results for the performance of the algorithms on different window sizes, shown in Figures 7.2.b, Figure 7.3.b, and Figure 7.5 reveal some interesting features of the BASH algorithms. For all the cases, BASH-AL is slightly faster and more efficient (in terms of memory use) than BASH-BL. However, for window sizes larger than 32k BASH-BL typically produces more accurate histograms. The reason behind this observation is that BASH-BL does not lose any accuracy while merging, due to the use of blocked bars. On the other hand, the size of each bar (EH sketch) increases with the increase in window size and, as a result, larger boxes may appear in each bar for larger windows. In this case, merging technique in BASH-AL loses more accuracy due to miss-ordering larger boxes. Thus, BASH-BL performs more accurately than BASH-AL in larger windows; however, it needs more time and memory to maintain the blocked bars.

On the other hand, the interpretation of the comparison results for space usage between BASH methods and AM may not be so clear cut. The memory usage of AM is almost fixed (or even decreases) for different window sizes, while it logarithmically increases in BASH algorithms. The better scaling of space usage of AM for larger windows is mainly due to the difference in the main objectives of the two algorithms. Observe that for larger windows, the allowed error bounds increase for both algorithms since the errors are proportional to the window size. While AM uses this relaxation on the error constraint to minimize memory usage, BASH seeks to maximize accuracy and minimize the running time as well as keeping memory usage in a reasonable range. This claim is apparent in Figures 7.2.b, Figure 7.3.b, and Figure 7.5, wherein for those cases in which memory consumption of AM is more efficient than the BASH methods, the accuracy of the histograms generated by AM as well as its running time are significantly worse than those of both BASH algorithms.

According to the timing results in Figure 7.2, both BASH-BL and BASH-AL are at least four times faster than AM, while producing histograms that have almost the same or even better accuracy. A more important concern is the scalability of the algorithms with respect to running times. As part b of Figure 7.2 indicates, the execution time for both BASH-BL and BASH-AL linearly increases with the window size, while this increase for AM is much faster. This is mainly because BASH does not employ a sorting technique as opposed to AM. Moreover, the results in part a of Figure 7.2 show that the running times of BASH algorithms decreases for data sets with smaller standard deviations (such as DS4, DS7, and DS11), while AM needs more time and memory for these types of distributions.

CHAPTER 8

Biased Histograms on Data Streams

As already discussed in the previous section, histograms provide statistically accurate and memory-efficient synopses for large data sets, and thus find many important uses in database and data stream applications. Query optimization, approximate query answering, distribution fitting, parallel database partitioning, and data mining are only a few examples of such applications. However, in most of these applications, the queries may focus on some particular regions of the data distribution—typically regions located at the extremes of data distributions. For instance, consider the daily precipitation across the US in Figure 8.1[1]. This data set has a very long tail where more than 99% of the values are less than 0.1 inch/day. As shown in the figure, an equi-depth histogram (dashed line) does not provide much information on the tail of the distribution since almost all the data items at this area (tail) are assigned to one bar. The same type of problem exists for equi-width histograms.

The biased-histogram synopsis proposed in this section provides a much better alternative for such distributions. For instance, in Figure 8.1, we see that the biased histogram supports much more accurate estimations for the tail of the distribution while preserving a good estimation for the head of the distribution (shown in the box in the middle of Figure 8.1). This is achieved by letting the size of the histogram bars decrease exponentially toward the biased region. In particular, in Figure 8.1, the size of each bar is 90% of the size of the bar to its immediate left—thus, we will say that our histogram has a *bias factor* of 0.9 or 90%.

[1]Data set is taken from http://www.ncdc.noaa.gov.

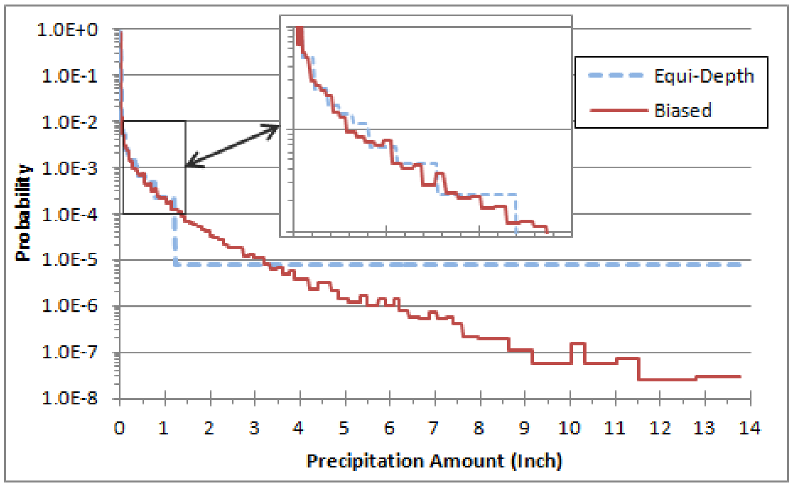


Figure 8.1: Equi-depth and biased histograms for annual precipitation in the US, with close-up magnification for the head of the distribution. (Each histogram has 100 bars.)

As another example, consider the distribution of the number of incoming (outgoing) URLs (links) in each page of the World Wide Web, which is known to be Zipfian. For this distribution, one may need reliable estimates of the average in-degree for the nodes that, say, rank in the interval 99.90% — 99.99% and for those in the interval 99.0% — 99.9%. Indeed this represents a crucial piece of information when we need to optimize the splitting/distributing of large data sets over different servers with roughly balanced load. Round Trip Times (RTTs) of the TCP packets provides another good example. Since RTT delays can stretch over long periods in various situations, the distribution of RTTs is very skewed at its tail. Therefore, performance monitoring systems need to watch the RTT distribution with a biased interest over the tail of the distribution to detect suspicious behaviors.

The histogram problem is akin to that of quantiles [GK01a], and there has also been some recent work on defining biased quantiles in data streams [CKM05, CKM06, ZLX06, ZW07a]. Quantiles are used to extract the item that occupies a given position in a sorted list of items in a data set or in a window in a data stream. Generally, a head-biased (tail-biased) quantile consists of the sequence of ϕ, ϕ^2, ϕ^3 , ... $(1-\phi$, $(1-\phi)^2$, $(1-\phi)^3$, ...) quantiles; for a given data set (or data stream) S with size N, the α-quantile $(0 \leq \alpha \leq 1)$ is defined as the value of the item at the position

$\lceil \alpha N \rceil$ in the sorted list of the items in S. Computing exact quantiles is a challenging problem even for small data sets [MP80], and thus incompatible with most data stream applications which require online response with limited memory and computational resources. As our experiments show, these problems carry over to biased quantiles. Moreover, although windows are crucial in most data stream applications, previous works on biased quantiles have always assumed that the quantiles represent the whole history of the data streams [CKM05, CKM06, ZLX06, ZW07a].

Therefore, we first define approximate biased histograms on data streams with sliding windows, and then propose a new efficient algorithm called *Bar-Splitting Biased Histograms* (BSBH) to computes approximate biased histograms over sliding windows of fast data streams. To the authors' knowledge, this is the first work on designing biased histograms over sliding windows of data streams. BSBH employs a similar structure as in our previous work on designing equi-depth histograms in [MZ11a]. BSBH also utilizes a biased sampling technique to improve the performance in terms of both CPU and memory usage. More specifically, this section makes the following contributions:

- We define the concept of *Biased Histograms* over sliding windows of data streams, and present a new algorithm called *Bar-Splitting Biased Histograms* (BSBH) to compute approximate biased histograms over fast data streams with sliding windows. (Section 8.1)
- To be able to tune the memory and CPU usage of BSBH, we propose a new biased sampling technique. Our experimental results indicate that biased sampling doubles the accuracy of the results with respect to uniform sampling while spending almost the same amount of memory and CPU. (Section 8.1.3)
- We prove that BSBH can guarantee expected ϵ-approximate biased histograms for data sets with no concept shift. We also provide theoretical bounds on both execution time and memory usage of our algorithm for this case in Section 8.2.

We use extensive experiments to evaluate the performance of BSBH and also compare it with CKMS which is one of the best existing algorithms for generating biased quantiles over the entire history of data streams [CKM06]. Our results (Section 8.3) show that BSBH outperforms CKMS with respect to execution time, memory usage, and accuracy. We have also evaluate BSBH for data streams with different rates of concept shifts on their distribution, and the results show that BSBH provides acceptably accurate results very quickly (mostly in couple of slides) after observing the concept shifts.

8.1 Biased Histogram Computation

This section first reviews the definitions of quantiles and histograms and then introduces the definition of biased histograms. The Bar-Splitting Biased Histogram (BSBH) algorithm is then thoroughly explained.

8.1.1 Definitions

Cormode *et. al.* defined *Biased quantiles* [CKM05] as follows:

Definition 6 *A Low-Biased Quantile with bias factor $\phi < 1$ for a given sequence of data items is the set of items with ranks $\lceil \phi^i N \rceil$ for i = 1, 2, ..., B=$log_{1/\phi}(N)$ in the ordered list of items, where N is the size of the data set.*

One can similarly define the *High-Biased Quantile* as well. The above definition is specifying B=$log_{1/\phi}(N)$ boundaries, which also can be seen as a B-bucket histogram. As stated in [MP80], one pass algorithms for computing the exact quantiles need to store the entire data set which is too expensive in most data streaming computations. Approximate biased quantile were thus defined in [CKM06] and [ZW07a] as follows:

Definition 7 *An approximate Low-Biased Quantile with bias factor $\phi < 1$ of a given sequence of data items, say S, is the set of items $\{v_i\} \in S$ for i = 1, 2, ..., $log_{1/\phi}(N)$, where:*

$$|rank(v_i) - \phi^i N| \leq max\{\epsilon.\phi^i N, \epsilon_{min} N\}$$

where $rank(x)$ is the position of data item x in the ordered list of items, and ϵ and ϵ_{min} are two approximation factors. The reason for needing two approximation factors is that the term $\epsilon.\phi^i N$ becomes very small for quantiles that are near the biased point (for larger i's). Thus if the term $\epsilon\phi^i N$ is used, unreasonably high accuracy would be required for quantiles near the biased point . To alleviate this problem, the alternate approximation factor (ϵ_{min}) is used instead. This essentially means that for the biased regions a flat error curve is used.

Although the above definition is useful in many applications, it is not suitable for designing biased histograms due to the following issues: First, the error is based on the rank of the reported quantiles, while in biased histograms we seek to minimize the error on the size of each bucket. Second, the definition imposes the restriction of having exactly $log_{1/\phi}(N)$ buckets or boundaries. However, many application may need to set the number of buckets based on their internal settings[2] which is usually set independent of the stream size. Third, since flat error rate (ϵ_{min}) is used for the areas near biased points, it might not provide an accurate result at those areas. This also makes the tuning of ϵ_{min} a challenging issue. To address these issues, we next define an approximate *Low-Biased Histogram* in which the goal is to keep the size of each bucket in the histogram close enough to the ideal size:

Definition 8 *An ϵ-approximate B-bucket Low-Biased Histogram (with bias factor $\phi < 1$) of a given sequence of ordered data items, say S, is the set of B buckets $\{B_i : i = 0, 1, ..., B-1\}$ partitioning S with the following invariant:*

$$|size(B_i) - \alpha_\phi \phi^i W| \leq \epsilon \alpha_\phi \phi^i W$$

where: W is the window size, $\alpha_\phi = (1-\phi)/(1-\phi^B)$ is a constant factor to make the buckets' sizes add up to W, and $size(B_i)$ is the number of items in bucket B_i. In other words, the ideal goal of the above definition is to partition the ordered data set

[2]e.g. It could be set according to the number of available processing units.

into B buckets of sizes $\alpha_\phi W$, $\alpha_\phi \phi W$, $\alpha_\phi \phi^2 W$, ..., and $\alpha_\phi \phi^{B-1} W$. Note that with this definition B is independent from the size of the stream and does not need to be exactly $log_{1/\phi}(W)$—a desirable property since W is often unknown a priori. Approximate *High-Biased Histogram* or *Targeted Quantiles* [CKM06] can be similarly defined[3].

8.1.2 Bar-Splitting Biased Histogram (BSBH)

The BSBH algorithm computes approximate B-bucket biased histograms over sliding windows of data streams. We assume the current size of the window at time t is W_t without needing to make any assumption about the type of the window (physical or logical.) As will be discussed later, BSBH maintains an estimation of W_t called W_{est}. Unlike other approaches [CKM05, CKM06], BSBH does not need the prior knowledge of U's size[4]. This section focuses on BSBH algorithm for generating low-biased histograms without using any sampling. Later in Section 8.1.3, we show how the idea of biased sampling can be added to BSBH in order to improve its time and memory performance.

BSBH Core Algorithm: The main goal in BSBH is to partition the interval between current minimum and maximum in the sliding window into $S_m = B \times p$ intervals bar_0, bar_1, ..., bar_{S_m-1}, where p is an extension factor for improving the accuracy and is determined analytically. The size of each bar is estimated using the EH structure discussed in Section 6.3.3. These bars will be later used to approximate the final $(B-1)$ buckets' boundaries.

The ideal goal of the BSBH structure is to keep the size of bar_i (i.e., $|bar_i|$) to ρ times the size of bar_{i+1} for $i = 0, 1, ..., S_m - 2$, where $\rho = \phi^{1/p}$. In other words, the ideal case is when:

$$|bar_i| = \rho|bar_{i+1}| = \alpha_\rho \rho^i W_t$$

[3] The notion of *end-biased histogram* used in [Ioa03] should not be confused with the above definition of biased histograms, since their idea is to put high-frequency items in singleton buckets.

[4] Universe U is the range of acceptable data values.

for $i = 0, 1, ..., S_m - 2$, where α_ρ is a factor that makes the bars' sizes add up to W_t (i.e. α_ρ = $(1-\rho)/(1-\rho^{S_m})$). As for W_t, BSBH uses the aggregate size of all the current bars which we will refer to as W_{est}. Due to properties of the EH structure, the expected value of W_{est} would be W_t if no sampling is used, however at each point in time these two may have different values. With such a setting, if the bars' sizes are ideal and no sampling is used, the aggregate size of bars bar_j for $ip \leq j < (i+1)p$ will be $\alpha_\phi \phi^{i+1} W$ for $i = 0, 1, ..., B-1$. These B buckets are actually what the definition of biased histograms (Definition 8) was seeking.

Algorithm 6 shows how the BSBH structure, mentioned above, is initialized and then maintained. We will next discuss how the structure is maintained once it is built, and discuss how it is initialized later in this section.

Maintenance: For each incoming data item from the data stream (say $next$), BSBH finds the appropriate bar, say bar_i, for $next$ based on the current boundaries of the bars (Line 20). This is done using a simple binary search. Then, the EH structure of bar_i is incremented by one (Line 23). While doing this, BSBH updates the current minimum and maximum values of the entire history of the data stream. At this point, if the bar_i's size is greater than a dynamically computed threshold ($maxSize_i$), BSBH splits the bar into two smaller bars (Lines 24 to 26).

While splitting bars, BSBH may need to merge two adjacent bars in order to keep the total number of bars below S_m. It is important to stress on that BSBH only merges two bars when a bar should be split and there is no room for a new bar. As opposed to BASH in which $maxSize$ is the same for all the bars at any given point in time, in BSBH, $maxSize$ is determined based on the bar's index and its closeness to the biased point. Note that, the idea is still similar to BASH; the size of bar_i should not be greater than $maxCoef > 1$ of its ideal size. In other words:

$$maxSize_i = maxCoef.\alpha_\rho.\rho^i.W_{est}$$

Although larger $maxCoef$ results in less splits, in practice we use $maxCoef = 1.7$ to reach higher accuracy as well as to accelerate the processes of stabilizing the boundaries for the initialization phase and to better cope with concept shift. Notice that having different thresholds for bars essentially allows smaller bars near the point of interest.

After assuring that the bars' sizes are in the acceptable range, if any of the boxes of the existing EHs is expired, BSBH will remove it from the structure (Line 27). Finally, after updating the structure for an incoming data item, we can compute the current boundaries for the final buckets. This is normally done at every slide (Lines 28 and 29). Next, we explain the main modules of the BSBH algorithm in more detail.

Initialization: The initialization phase plays a very important role in the BSBH algorithm. This is mostly because at the beginning, the minimum and maximum values of the data set are unknown. Thus, instead of starting with S_m bars BSBH starts with one empty bar, and keeps adding new data items into that bar until its size reaches a threshold ($maxSize_0$). At this point, the bar is full and it will be split into two bars. BSBH repeats this until S_m bars are created. After this stage for each split operation, two consecutive bars should be merged. Note that $maxSize_i$ at all time is proportional to the current window size $W_t \simeq W_{est}$; this essentially means that we allow more splits during the initiation phase of the algorithm, since W_t is small at this phase. This makes the initiation phase much faster. For instance due to our threshold-based mechanism, the first split happens usually at the second or third insert.

Merge-Bar Operation: Let two adjacent bars, say X_L and X_R, are selected to be merged into one bar called X. Later in this section, we discuss the details on how the bars are chosen for merging. As depicted in Figure 8.2, BSBH first integrates the intervals that X_L and X_R are covering. Next, for the EH sketches of bars X_L and X_R called EH_{X_L} and EH_{X_R} respectively, BSBH set the smaller one to *blocked*, and the bigger one to *active*. In other words, the new bar X now has two associated EH sketches

Algorithm 6 BSBH()

```
1 : BSBH ($\phi$, $\mu$, $B$, $p$)
2 : {
3 :     $\mu'$ = 1; /* for gradually decreasing sample rate*/
4 :     $effectSmp$ []; /* initialized with 1's */
5 :     $\rho$ = $\phi^{1/p}$;
6 :     if ($\mu < \rho$)
            $\mu = \rho$;
7 :     $initialize$();
8 :     $cnt$ = 0;
9 :     while (true)
10:     {
11:         $next$ = next item in the data stream;
12:         $cnt$ ++;
13:         //Updating the Sampling Rate
14:         if ($cnt \leq X$)
15:             if ($\mu' > \mu$)
16:                 $\mu'$ = $1 - ((1 - \mu) \times (cnt - X))/X$;
17:         if ($cnt \leq 2X$)
18:             Compute effective sampling rate for each bar and
                Update $effectSmp$ array;
19:         //Biased Sampling Phase
20:         find appropriate bar $bar_i$ for $next$;
21:         if (!Biased-Sample($next$, $i$, $\mu'$))
22:             continue; /* Skipping the item */
23:         insert $next$ into $bar_i.EH$;
24:         $maxSize_i$ = $\alpha_\rho.maxCoef.\rho^i.W_{est}$;
25:         if ($|bar_i| > maxSize_i$)
26:             $splitBar$($bar_i$, $\rho$);
27:         remove expired boxes from all EHs;
28:         if (window slides)
29:             output $computeBoundaries$($B$);
30:     }
31: }
```

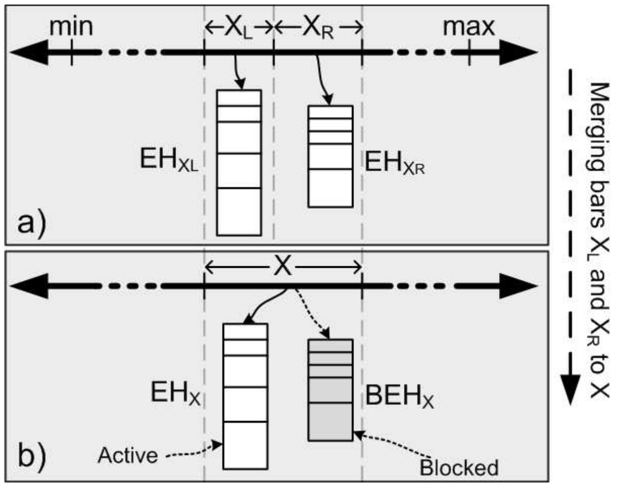


Figure 8.2: Merging two bars (X_L and X_R) into one bar (X). a) right before merge-bar operation and b) right after merge-bar operation.

but only one of them is active. This means that the incoming tuple for the X's interval will be inserted into the active EH (EH_{X_L} or EH_X in our example in Figure 8.2). As for the blocked EHs, BSBH stops incrementing them, but continues removing expired boxes from them. Thus, blocked EHs are prevented from growing, and as a result they will soon be disappeared from the structure and will no longer require memory. Also, observe that every bar may contain more than one blocked EH. For instance, consider the case where we have to merge two adjacent bars which already have an associated blocked EH (e.g. merging bar X in part b of Figure 8.2 with one of its neighbors). To merge these two, we follow the same general approach: the longest EH is set to the active one and all other EHs are considered as the blocked EHs of the new bar.

At all times, BSBH makes sure that the size of the active EH for each bar is bigger than those of the blocked EHs of the bar, and whenever this is no longer the case, BSBH switches the active EH with the blocked EH of larger size. Although this case is rare, it can occur when the boxes of the active EH expire more quickly than those of the blocked EHs. In Section 8.2, we will prove that the expected number of such blocked EHs is constant for large enough N's. Notice that since the internal structure of EHs is untouched in this merging technique, the technique does not affect the accuracy of the final results.

Split-Bar Operation: When the size of a bar, say X, exceeds its maximum threshold ($maxSize_i$, where i is the position of X in the list of current bars), we split it into two smaller bars using Algorithm 7. Before the split operation takes place, we make sure that the number of bars in the histogram is less than S_m. If this is not the case, as previously explained, we should first merge two other adjacent bars.

Let bar X need to be split to two bars called X_L and X_R. We divide the interval being covered by X into a pair of intervals (Lines 5 to 8). The goal is that $|X_R| = \rho \times |X_L|$ after the split operation takes place. To this end, the algorithm first distributes the blocked EHs of X, denoted as BEH_X, into the blocked EHs of X_L and X_R (Lines 9 to 16). The splitting algorithm tries to do this in a way that preserves the size constraint ($|X_R|/|X_L|$ = ρ); however this is not always possible. That is after splitting blocked EHs of X to X_L and X_R the ratio ρ' = $|X_R|/|X_L|$ may be different than the ideal ratio (ρ). Thus, BSBH splits the active EH of X (EH_X) to compensate for this difference ($\rho - \rho'$) (Lines 17 to 29). It is easy to show that we can always achieve this goal by using appropriate split on EH_X, given that the size of the active EH is guaranteed to be greater than the size of all the blocked EHs.

To split the original EH sketch, EH_X, into a pair of sketches called EH_{X_R} and EH_{X_L}, BSBH first computes the split ratio denoted as λ (Line 17 in Algorithm 7). λ simply indicates that to compensate for the aforementioned difference ($\rho - \rho'$), the size of EH_{X_R} after splitting should be λ times the size of EH_X (e.i. $|EH_{X_R}|$ = $\lambda|EH_X|$). In order to have such a split ratio, BSBH puts half of the EH_X boxes with size one into EH_{X_L}, and the other half into EH_{X_R} (Line 19 and 20). Then, it replaces each of the remaining boxes in EH_X with two boxes that are half the size of the original box. Finally, based on the current ratio of the sizes, BSBH decides whether to put each copy to EH_{X_R} or EH_{X_L} (Lines 23 to 27). In other words, if the current ratio of $|EH_{X_R}|/|EH_{X_R} + EH_{X_L}|$ is smaller than λ the copy will go to EH_{X_R}, otherwise it will go to EH_{X_L}. For instance if λ is 0.5, one half goes to EH_{X_R} and the other half goes to EH_{X_L}.

Algorithm 7 splitBars(X, ρ)

```
1 : if ($curBarNo$ == $S_m$ && !$mergeBars()$)
        return;
2 : Initialize new bars $X_L$ and $X_R$;
3 : $l = 0$;
4 : $|BEHs|$ = Aggregate Size of the blocked EHs;
5 : $EH_{X_L}$.start = $EH_X$.start;
6 : $EH_{X_L}$.end = ($EH_X$.start + $EH_X$.end)/(1+$\rho$);
7 : $EH_{X_R}$.start = $EH_{X_L}$.end;
8 : $EH_{X_R}$.end = $EH_X$.end;
9 : for each (blocked EH $bar$ in $BEH_X$) {
10:    if ($l$ + $|bar|$ ¡ $|BEHs|$/(1+$\rho$)){
11:       $l$ += $|bar|$;
12:       Add $bar$ to $BEH_{X_L}$;
13:    }
14:    else
15:       Add $bar$ to $BEH_{X_R}$;
16: }
17: $\lambda$ = (($|X|$/$(1+\rho)$)-$l$)/($|X|$ - $|BEHs|$);
18: foreach (box $box$ in $EH_X$){
19:      if ($|box|$ == 1)
20:         Alternatively add a copy of $box$ to $EH_{X_L}$ or $EH_{X_R}$;
21:      else {
22:         $|box|$ = $|box|$/2;
23:         for (2 times)
24:               if ($|EH_{X_R}|$ /($|EH_{X_R}|$+$|EH_{X_L}|$) ¡ $\lambda$)
25:                  Add a copy of $box$ to $EH_{X_R}$;
26:               else
27:                  Add a copy of $box$ to $EH_{X_L}$;
28:      }
29: }
30: Remove $X$ from the bars list;
31: Merge boxes of bars $X_L$ and $X_R$ if necessary.
32: Add bars $X_L$ and $X_R$ to the bars list;
```

Selecting Bars to Merge: We always merge adjacent bars that have the minimum aggregate size with respect to their positions and ideal size. Thus, we select a pair of adjacent bars that yields the least relative deviation from their ideal sizes. In other words, we pick bars bar_i and bar_{i+1} such that $|bar_i|/\rho^i + |bar_{i+1}|/\rho^{i+1}$. If this aggregate size of these two bars (e.i. $|bar_i| + |bar_{i+1}|$) is less than $maxSize_i$ (which is usually the case), we select them for merging. Otherwise, we do not perform any merge operations until some boxes expire, since we do not want to create bars with size greater than $maxSize_i$ by merging.

An Alternative Merging Approach: Similar to the BASH case (Section 7), we can merge the bars, by combining the boxes in their associated EHs. The technique which uses BLocked EHs is called BSBH-BL, and this alternative approach is referred to as BSBH-AL throughout the rest of the section. Again, notice that by merging boxes from two different EH sketches, we may lose some information about timestamps of the items which affects the accuracy of the estimations as discussed in Subection 8.3. Although, we do not have any blocked EHs for BASH-AL, the same split approach we mentioned earlier can be used for this alternative merging technique. This time, the splitting operation in Algorithm 7 splits the only existing EH into two EHs with appropriate size.

Reporting the Final Buckets: At each slide, the algorithm needs to generate a new set of boundaries/buckets. Algorithm 8 shows the pseudocode for this operation, in which the ideal goal is to find B buckets with sizes $\alpha_\phi \phi^i W_t$ for i=0, 1, ..., $B-1$. Similar to [MZ11a], Algorithm 8 passes over the list of bars once and estimates B_is boundaries assuming that the distribution of items inside each bar is uniform. Observe that the extension factor p plays an important role in this assumption. Indeed, larger values of p generate smaller intervals, and consequently make the distribution of the items in each bar closer to a uniform distribution. Since the value of W_t might not be known, we will instead use its estimate W_{est}.

Algorithm 8 computeBoundaries(B)

```
$b = -1$; //An index over bars list
$count = 0$;
$curBuckSize = \alpha_\phi \phi^{B-1} W_{est}$;
for(int $i = 0$; $i < B$; $i$++) {
    while ($count \leq curBuckSize$) {
        $b$ ++;
        $count$ += $|bars[b]|/effectSmp[b]$;
    }
    $surplus = count - curBuckSize$;
    $boundaries[i] = bars[b].start + bars[b].length \times$
        $(1 - surplus/(|bars[b]|/effectSmp[b]))$
    $count = surplus$;
    $curBuckSize = curBuckSize / \phi$ ;
}
return $boundaries$;
```

8.1.3 Biased Sampling

Sampling provides a simple technique for scaling many algorithms to larger data sets. The most common way to sample is to uniformly drop items from the input data set and reduce its volume. Although this is very easy to implement, it may not be the best practice for biased histograms, since uniformly removing items from smaller bars has a more negative effect in estimating the boundaries than in larger bars. Thus, a good sampling technique needs to sample out fewer items around the biased point(s) than from the rest of the data distribution.

Random non-uniform sampling was also used by Zhang and Wang [ZW07a] who first collect and sort the whole input and then sample from the ordered input at non-uniform rate. BSBH instead introduces a faster non-blind sampling technique, called *biased sampling*, which does not need to see the entire data set in advance. This, makes BSBH a much more practical approach for data streams. The goal in biased sampling is to sample (keep) at most σ items from each window. Thus for a low-biased histogram, knowing the new items value ($next$) and its associated bar index (i), we sample $next$

with probability μ^i, such that the following equation holds:

$$\sum_{i=0}^{S_m-1} \mu^{S_m-1-i}|bar_i| = \sigma \times W_t$$

As you can see, the sampling occurs such that we keep all the items in the smallest bar (bar_{S_m-1}), sample items from bar_{S_m-2} with rate $\mu \leq 1$, and exponentially decrease the sampling rate to a more selective one for larger bars. We refer to μ as the *biased sampling rate*. Note that it is not plausible to sample in a way that after sampling the size of bar_{i+1} becomes smaller than the size of bar_i, which means μ should be less than or equal to ρ ($\mu \leq \rho$). If smaller sample rates are desired, uniform sampling should be used. For the case of $\mu = \rho$, it is easy to see that after observing at most $2W$ items all bars will have equal sizes (which is $\alpha_\rho \rho^{S_m-1} W_t$) and the biased histogram computation is reduced to an equi-depth histogram computation. This basically means that BSBH with biased sampling rate ρ will be eventually reduced to BASH after the sampling phase is completed.

One challenge most of the sampling-based techniques need to address is the initiation phase. Since the system is initially empty, to quickly, and more accurately report the first set of results, the algorithms need to keep more data items and drop less. Later when the structures are initiated, the techniques can sample more data items to reduce the load. Moreover in our case, the sampling is not blind and needs to know the boundaries of the histogram's bars. Thus, we need to first generate the boundaries so we can start the sampling phase. In order to alleviate the effect of switching to the sampling stage, we gradually decrease biased sampling rate from 1 (no sampling) to μ for the first X data items (lines 14 to 16 in Algorithm 6). For the physical windows X could be set to the window size W. For the logical windows, X can be set to estimated size of the window at the time in which the first item expires.

Since we can not start full sampling form the beginning, the effective biased sampling rate for bar_i is not going to be μ^i for the first $2X$ data items. Thus, in lines 17 and 18 of Algorithm 6, we update the effective sampling rate for each bar in an array called $effectSmp$ based on the history of changes on sampling rates. $effectSmp$ will be used later to estimate the actual size of bars.

8.2 Formal Analysis

In this section, we prove that BSBH can provide an expected ϵ-approximate biased histogram for sliding windows on data streams with no concept shift for any given $\epsilon > 0$. We also compute the per-data item delay and space complexity of BSBH for this case. Notice that data streams with no concept shift can be modeled as the case that every incoming item from the data stream is taken form a fixed data distribution. This consequently means that items are arriving in random order. To ease our discussion, we consider that the window size is fixed at W. First, we compute the expected number of splits in each sliding window.

Let $E_{sp}\{N\}$ be the expected number of splits after processing N data items from the data streams. Observe that the expected number of splits for bar_i is $E_{sp}\{N\}/S_m$ since the probability of insert to (and expire from) bars for randomly ordered inputs are the same for each bars. Let $p_{t_1,t_2,i,j}$ be the probability that the current bar at i'th index/position (bar_i) in time t_1 is moved to position j at time t_2 without being split or merged (That is, the change of position is only due to splitting and merging of other bars). Also, let $q_{t,i}$ be the probability of having bar_i split at time t. Thus, the probability of having two consecutive splits on a bar at times t_1 and t_2 and at the starting position i and the finishing position j is $P_{t_1,t_2,i,j} = q_{t_1,i}p_{t_1+1,t_2,i,j}q_{t_2,j}$. Knowing this, we can now calculate the expected number of inserts into bar_i (called $E_{in}\{i\}$) between two consecutive splits using the following formula:

$$
\begin{aligned}
E_{in}\{i\} &= \frac{S_m}{E_{sp}\{N\}} \sum_{t}^{N} \sum_{t'=t}^{N} \sum_{j=0}^{S_m-1} P_{t-1,t',i,j}(x_{t,t',i} + m_j - \frac{m_i}{2}) \\
&= \frac{S_m}{E_{sp}\{N\}} \sum_{t}^{N} \sum_{t'=t}^{N} \sum_{j=0}^{S_m-1} P_{t-1,t',i,j}(m_j - \frac{m_i}{2}) \\
&\quad + \frac{S_m}{E_{sp}\{N\}} \sum_{t}^{N} \sum_{t'=t}^{N} \sum_{j=0}^{S_m-1} P_{t-1,t',i,j}(x_{t,t',i}) \\
&= I_i + X_i
\end{aligned}
$$

where m_i is the abbreviated form of $maxSize_i$ and $x_{t,t',i}$ is the expected number of expired items from bar_i in between the time interval $[t, t']$. Basically, for two consecutive splits on bar_i at times $t-1$ and t', we need to insert $m_j - \frac{m_i}{2}$ items to reach to the split threshold at position j if nothing expires from the bar. Otherwise, we need to insert and additional $x_{t,t',i}$ of items to compensate for the expired items from the bar. The above equation simply adds up all of such values for all possible t, t', and j. Then, the expected number of inserts among all bars between two consecutive splits can be computed with the following formula:

$$
E_{in} = \sum_{i=0}^{S_m-1} E_{in}\{i\} = \sum_{i=0}^{S_m-1} I_i + \sum_{i=0}^{S_m-1} X_i = I + X \tag{1}
$$

Let us start with I:

$$
\begin{aligned}
I &= \sum_{i=0}^{S_m-1} \frac{S_m}{E_{sp}\{N\}} \sum_{t}^{N} \sum_{t'=t}^{N} \sum_{j=0}^{S_m-1} P_{t-1,t',i,j}(m_j - \frac{m_i}{2}) \\
&= \frac{S_m}{E_{sp}\{N\}} \sum_{t}^{N} \sum_{t'=t}^{N} \sum_{i=0}^{S_m-1} \sum_{j=0}^{S_m-1} P_{t-1,t',i,j}(m_j - \frac{m_i}{2}) \\
&= \frac{S_m}{2E_{sp}\{N\}} (\sum_{t}^{N} \sum_{t'=t}^{N} \sum_{i=0}^{S_m-1} \sum_{j=0}^{S_m-1} P_{t-1,t',i,j}(m_j - \frac{m_i}{2}) \\
&\quad + \sum_{t}^{N} \sum_{t'=t}^{N} \sum_{i=0}^{S_m-1} \sum_{j=0}^{S_m-1} P_{t-1,t',j,i}(m_i - \frac{m_j}{2})) \\
&= \frac{S_m}{2E_{sp}\{N\}} \sum_{t}^{N} \sum_{t'=t}^{N} \sum_{i=0}^{S_m-1} \sum_{j=0}^{S_m-1} P_{t-1,t',i,j}(\frac{m_i + m_j}{2})
\end{aligned}
$$

In the last line of the above equations, we have used the fact that probabilities $q_{t,i}$ and $q_{t,j}$ are equal for any given t for randomly ordered inputs with no concept shift. This also indicates that $P_{t_1,t_2,i,j}$ and $P_{t_1,t_2,j,i}$ are equal. To understand this fact, let $P_{t_1,t_2,i,j}$ ¿ $P_{t_1,t_2,j,i}$ (or $P_{t_1,t_2,i,j}$ ¿ $P_{t_1,t_2,j,i}$). This means that the bars between i and j indices are constantly moving toward the index j (or i) due to splitting and merging which is in contradiction with the fact that the data stream contains no concept shift.

$$
\begin{aligned}
I &= \frac{S_m}{2E_{sp}\{N\}} \sum_{t}^{N} \sum_{t'=t}^{N} \sum_{i=0}^{S_m-1} \sum_{j=0}^{S_m-1} P_{t-1,t',i,j}(\frac{m_i + m_j}{2}) \\
&= \frac{S_m}{2E_{sp}\{N\}} \sum_{t}^{N} \sum_{t'=t}^{N} \sum_{i=0}^{S_m-1} \sum_{j=0}^{S_m-1} P_{t-1,t',i,j} m_i \\
&= \frac{S_m}{2E_{sp}\{N\}} \sum_{i=0}^{S_m-1} m_i \sum_{t}^{N} \sum_{t'=t}^{N} \sum_{j=0}^{S_m-1} P_{t-1,t',i,j}
\end{aligned}
$$

Note that $\sum_{t}^{N} \sum_{t'=t}^{N} \sum_{j=0}^{S_m-1} P_{t-1,t',i,j}$ computes the expected number of splits for

bar_i for the first N data items of the stream. As already discussed this value is equal to $E_{sp}\{N\}/S_m$. Therefore,

$$I = \frac{S_m}{2E_{sp}\{N\}} \sum_{i=0}^{S_m-1} m_i(\frac{E_{sp}\{N\}}{S_m})$$
$$= \frac{1}{2}\sum_{i=0}^{S_m-1} m_i = \frac{maxCoefW}{2} \tag{2}$$

The above formula simply indicates that the expected number of inserts per split, or E_{in}, is greater than $maxCoefW/2$. This leads us to the following lemma which is very important to prove the bound on memory requirements of BSBH.

Lemma 3 *For $N \geq W$, the expected number of blocked EHs in BSBH is $O(1)$.*

Proof: Equation 2 indicates that for $N \geq W$, the expected number of inserts before seeing the next split in BSBH is greater than $maxCoefW/2$ (equation $\sum_{i=0}^{S_m-1} m_i = maxCoefW$ only holds for $N \geq W$.) That is, in each window the number of splits is $O(1)$ on the average. On the other hand, the number of blocked EHs is proportional to the number of splits as proved in Lemma 2 of [MZ11a]. This completes the proof. □

Note that after first W items have arrived, older items start to expire with the same rate as new-arriving items. This essentially means that the expected number of items expiring in a given time interval is the same as the expected number of arriving items ($X = I$). That is after the first window:

$$E_{in} = I + X = 2I = maxCoef \times W$$

This essentially means that for $maxCoef > 1$, E_{in} would be more than W. In other words, the expected number of split in each window is less than 1.

Theorem 3 *For data stream S with no concept shift and for any given $0 < \epsilon$ and*

$0 < \phi < 1$*, the BSBH algorithm can provide a size-based expected ϵ-approximate biased histogram with bias factor ϕ for sliding windows on S.*

Proof: As stated above in each window there can be at most one split, and no matter how poorly this split operation is performed, before the next split all the wrongly split items will have been expired. Thus at any moment, only two adjacent bars may contain incorrect items due to the split operation. We refer to these two bars as the split bars. Let us first compute a bound on the expected error imposed by these two bars. The final boundaries that Algorithm 8 generates may cut at most $B - 1$ bars. We refer to the boundaries as bn_1, bn_2, ..., bn_{B-1} and to the bars they cut as the cut bars (b_1, b_2, ..., b_{B-1}). If both split bars are among un-cut bars they do not impose any error since they compensate for each other in the final bucket. Otherwise, one of them should be among b_i's. We will return to the error caused by the b_is after we consider the error caused by the second split bar. Let the second split bar be in interval [bn_i, bn_{i+1}). The maximum error it generates (called err_{sp}) can be computed as:

$$\frac{|bar_{ip+1}|}{|B_i|} \leq \frac{maxCoef.\alpha_\rho.\rho^{ip+1}.W}{\alpha_\phi.\phi^i.W} = \frac{maxCoef(1-\rho)\rho}{(1-\phi)}$$

The other part of the error is because of the the position of boundaries in the cut bars. Considering the estimated number of items at the left and right sides of boundary bn_i in bar b_i are respectively called l_i and r_i ($l_i + r_i = |b_i|$), the maximum error imposing by wrongly selecting the boundaries for B_i is then $(r_i + l_{i+1})/|B_i|$. Thus on aggregate for the cut bars error (err_{cut}), we have:

$$\begin{aligned}\sum_{i=1}^{B-1}\frac{(r_i+l_{i+1})}{|B_i|} &\leq \sum_i \frac{(r_i+l_i)}{|B_i|} = \sum_i \frac{|b_i|}{|B_i|}\\ &\leq \sum_{i=1}^{B-1}\frac{maxCoef.\alpha_\rho.\rho^{ip}.W}{\alpha_\phi.\phi^i.W}\\ &= \sum_{i=1}^{B-1}\frac{maxCoef(1-\rho)}{(1-\phi)}\\ &= (B-1)\frac{maxCoef(1-\rho)}{(1-\phi)}\end{aligned}$$

Knowing that each bar uses an EH sketch with approximation error δ, we can compute the following bound for the final expected approximation error for the buckets in our BSBH algorithm:

$$\begin{aligned}ErrRate &= \delta + (err_{sp} + err_{cut})/B\\ &\leq \delta + \frac{maxCoef(1-\rho)\rho}{(1-\phi)B} + \frac{maxCoef(1-\rho)}{(1-\phi)}\\ &\leq \delta + c(\frac{\rho}{B}+1)(1-\rho) < \delta + 2c(1-\rho)\end{aligned} \tag{3}$$

where c=$maxCoef/(1{-}\phi)$ is a constant. This proves that the expected approximate error is bounded. Calling this bound ϵ and noting that $\rho = \phi^{1/p}$, one can say that by setting $\delta = \epsilon/2$ ($k = \lceil 2/\epsilon \rceil$) and $p = \lceil -2c.log(\phi)/\epsilon \rceil$, we obtain a size-based expected ϵ-approximate low-biased histogram. To see this observe that:

$$p = \lceil \frac{-2c.log(\phi)}{\epsilon} \rceil \geq \frac{-log(\phi)}{\epsilon/2c} \geq \frac{log(\phi)}{log(1-\epsilon/2c)}$$

The last part of the above inequality is derived from the fact $e^x > 1 + x$ (where

x can be set to $1 - \epsilon/2c$). Now by replacing this lower bound for p in Formula 3, the proofs will be completed. □

Empirical results show that even much smaller values for k and p can provide the same accuracy level. However, it is necessary to mention that the larger the expansion factor p is, the quicker the convergence of the algorithm will be especially for those data sets with concept shifts. Lemma 3 shows that the total number of bars (active EHs plus blocked EHs) in BSBH is $O(S_m)$. Additionally, the size of all of these bars is smaller than the window size, which indicates that each bar needs at most $\frac{1}{\delta}log(\delta W)$ of space. These two facts lead us to the following bound on the expected memory usage of BSBH:

Theorem 4 *The expected memory usage of the BSBH algorithm is bounded by* $O(\frac{S_m}{\delta}log(\delta W))$ = $O(\frac{B}{\epsilon^2}log(\epsilon W))$ *, where W is the sliding window size and* ϵ *is the approximation factor of BSBH algorithm.*

We can provide a bound on the per item CPU usage in the following theorem. Since the proof of this theorem is identical to the one in the previous section, we skip it here.

Theorem 5 *BSBH on average spends* $O(log(S_m) + \frac{S_m}{S})$ = $O(log(\frac{B}{\epsilon}) + \frac{B}{\epsilon S})$ *time for each input data item, where* S *is the slide size.*

Note that in practice and as shown in the experimental results, this time complexity is very close to a constant time, since S is usually much larger than S_m, and $log(S_m)$ is usually very small.

8.3 Experimental Results

We implemented both versions of the BSBH algorithm in C++. The version of BSBH which blocks the bars in the merge operation is referred to as BSBH-BL, and the alternative approach is called BSBH-AL. We also compared BSBH with the CKMS algorithm proposed by Cormode et al. [CKM06] for the case of no sliding windows, since CKMS does not support it. All the errors were computed using average size error of the histograms' bars as defined in Definition 8. For all the experiments shown in this section, we fixed the slide size at 1000 tuples and number of bars at 20. No prior knowledge of the minimum and maximum values of the incoming data was considered. We also set δ to 0.1, p to 7, and $maxCoef$ to 1.7 which are experimentally proved to be optimum in [MZ11a]. All the experiments were run on a 64bit, 2.27 GHz machine running CentOS with 4GB of main memory (RAM) and 8MB of cache.

Table 8.1: Data sets used for the experimental results

Name	**Dist.**	**Size**	**Shifts No.**	**Parameters**
DS1	Uniform	$1m$	0	$min0, max10k$
DS2	Normal	$1m$	0	$\mu5k$, $\sigma500$
DS3	Zipfian	$1m$	0	$\alpha1.5$
DS4	Zipfian	$1m$	0	$\alpha1.8$
DS5	Exponential	$1m$	0	$\lambda10^{-3}$
DS6	Exponential	$1m$	0	$\lambda10^{-4}$
DS7	Zipfian	$1m$	100	starting $\alpha1.5$
DS8	Zipfian	$1m$	10	alt. $\alpha3$ & $\alpha1.1$
DS9	Exponential	$1m$	100	starting $\lambda10^{-3}$
DS10	Precipitation	$21m$	-	-

8.3.1 Data sets

We have considered 10 data sets wherein one is a real-world data set (DS10), and the rest are synthesized data sets with different distributions as shown in Table **??**. As you can see, the first six data sets (DS1 to DS6) contain no concept shifts which means their distribution does not change through time. In DS7 and DS9, we shift the concept by randomly increasing or decreasing the distribution's parameters (respectively α and λ) by 10% for every 10K data items. We have also generated a data set with 10 unrealistically large concept shifts, called DS8. In this data set, the distribution alternates between Zipfian with $\alpha = 3.0$ and $\alpha = 1.1$. Our real data set contains the amount of daily precipitation recorded in different stations across the United States for about 400

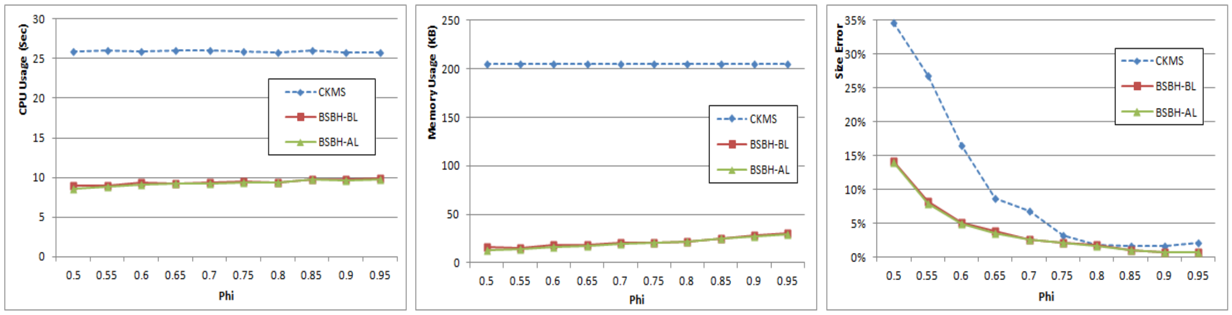


Figure 8.3: a) The execution time, b) memory usage, and c) size error of BSBH and CKMS algorithms on DS3. ($k = 10$, $p = 7$, $\epsilon = 0.01$, and $\epsilon_{min} = 0.001$.)

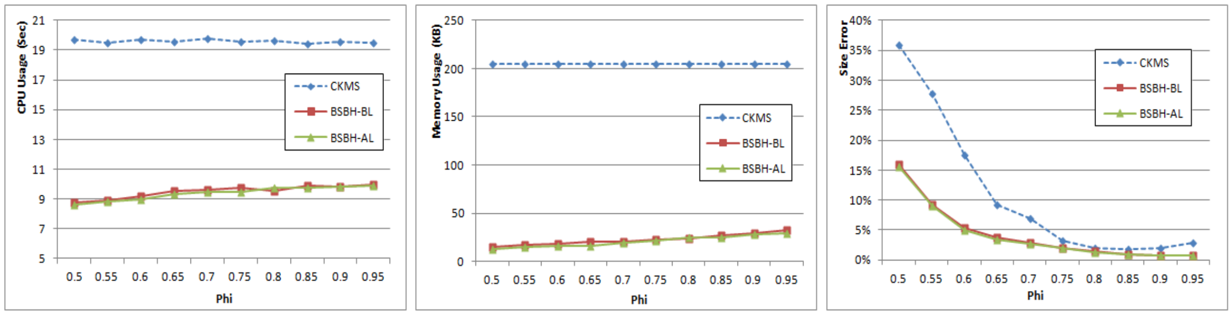


Figure 8.4: a) The execution time, b) memory usage, and c) size error of BSBH and CKMS algorithms on DS7. ($k = 10$, $p = 7$, $\epsilon = 0.01$, and $\epsilon_{min} = 0.001$.)

years[5]. Although, there are many missing records for early years, this data set contains more than 21 million records. An estimation for the distribution of data in this data set (Figure 8.1) indicates that the distribution has a very long tail.

8.3.2 BSBH versus CKMS

Figures 8.3 and 8.4 provides the execution time, memory usage, and size error of the BSBH algorithms as well as CKMS algorithm for data sets DS3 and DS7 respectively. Both of these two data sets have Zipfian distribution (with α=1.5), but DS7 contains 100 random concept shifts. For CKMS, we set ϵ and ϵ_{min} respectively to 0.01 and 0.001 as suggested in [CKM06]. As can be seen in the figure, both BSBH-BL and BSBH-AL are constantly faster, lighter (in the sense of memory usage), and more accurate than

[5]Taken from http://www.ncdc.noaa.gov.

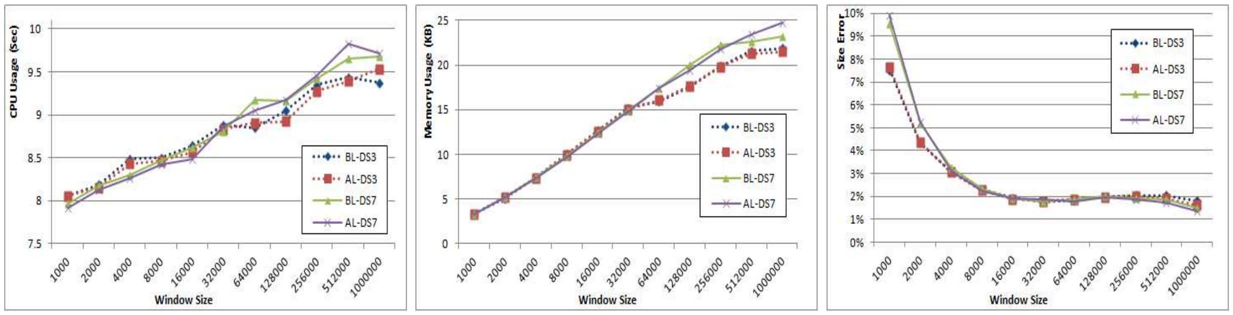


Figure 8.5: a) The execution time, b) memory usage, and c) size error of BSBH algorithms on DS3 and DS7 for different window sizes. ($\phi = .8$, $k = 10$, $p = 7$, $\epsilon = 0.01$, and $\epsilon_{min} = 0.001$.)

CKMS, even when the results for BSBH algorithms are taken over the whole history of data streams. For the current setting, BSBH is at least twice as fast as CKMS while using more than six times less memory and providing results with twice or more better accuracy. We should mention that CKMS provides a worst-case error guarantee while BSBH provides an expected error guarantee. This mainly explains the gap between memory and CPU usage of the two approaches. Note that the error increases for smaller ϕ's in both algorithms. This is mainly because for smaller ϕ's, bars closer to the biased regions get even smaller. As a result, even one misplaced item in those small bar may result in a very high error rate. This is actually why CKMS uses the flat error rate ϵ_{min} at the biased points.

8.3.3 Scalability

As discussed in previous sections, BSBH uses a constant number of Exponential Histograms and as a result, we claimed BSBH will inherit the scalability of the Exponential Histograms. To verify our claim, we tested the execution time, memory usage, and size error of our two BSBH algorithms over data sets DS3 and DS7 for different window sizes in Figure 8.5. Parts (a) and (b) of this figure depict that both CPU and memory usage of BSBH algorithms have logarithmic relation with respect to the size of the windows which proves the scalability of our structure for larger window sizes. In part (c)

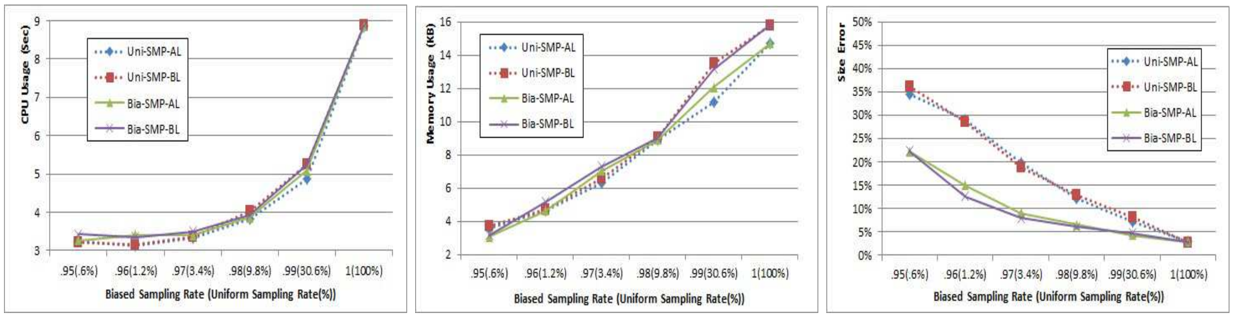


Figure 8.6: a) The execution time, b) memory usage, and c) size error of BSBH algorithms on DS7 for different sampling rates. ($\phi = .7$, $W = 100K$, $k = 10$, and $p = 7$.)

of the figure, we included the error which is steady at 2% for larger than 16K window sizes.

It is also worthy to compare the results for DS3 and DS7, in which the only difference is the presence of concept shift in DS7. For smaller window sizes, the errors (part (c) in Figure 8.5) are higher for both data sets, since there is not enough data in each bar to make a good estimation. DS7's error is even slightly higher due to several concept shifts. For larger window sizes, which are more probable in fast data streams, the accuracy for both data sets is almost the same. However, DS7 needs slightly more memory and time due to the larger number of split/merge operations. We discuss more results on data sets with concept shifts in Section 8.3.5.

8.3.4 Biased Sampling versus Uniform Sampling

To evaluate the effect of our proposed biased sampling technique, we compared the execution time, memory usage, and size error of BSBH algorithms for two types of sampling techniques; uniform sampling and biased sampling. In uniform sampling, we simply dropped items with a fixed probability. The results for data sets DS7 and DS10 are shown in Figures 8.6 and 8.7. The biased sampling rate (μ) and its equivalent uniform sampling rates (σ) are shown in the horizontal axis in the figures. As part (c)'s of the figures depict, the accuracy of the biased sampling is superior to that

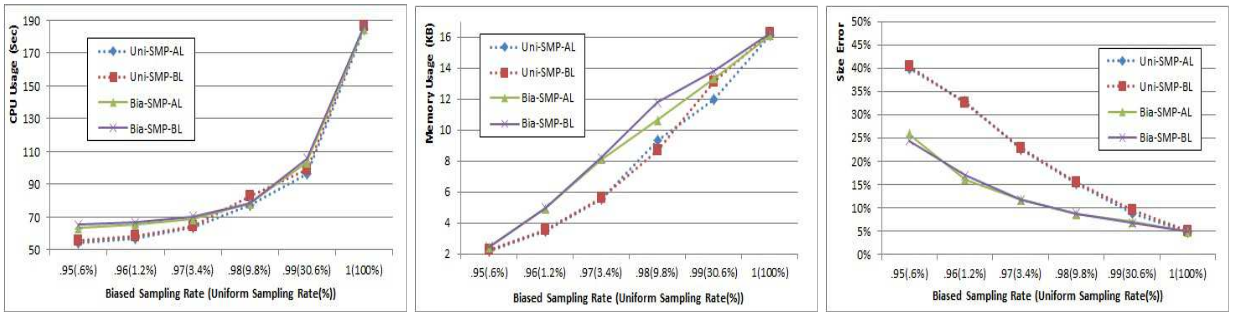


Figure 8.7: a) The execution time, b) memory usage, and c) size error of BSBH algorithms on DS10 for different sampling rates. ($\phi = .7$, $W = 100K$, $k = 10$, and $p = 7$.)

of uniform sampling, especially when the sampling rate increases. However, biased sampling needs slightly more memory than uniform sampling (part (b) of the figures) because fewer items from small bars are sampled out. This increases the number of boxes in small bars. Notice that small changes in the number of items in larger bars usually does not significantly change the number of boxes in them. As for the CPU usage s(part (a) of the figures), both techniques performs similarly. However, the interesting result is that by biasing the sampling rate (μ) to less than $.98$ none of these techniques significantly improve the CPU performance.

8.3.5 The Effect of Concept Shifts

As already shown in Figure 8.5, moderate concept shifts in the data stream have a very small effect on the performance of the BSBH algorithms. To take a closer look at the effect of concept shifts on the performance of BSBH, we compared the evolution of the size error through time for DS7, DS8, and DS9. The results (Figure 8.8) indicate that concept shifts have insignificant effect on the size error for data sets DS7 and DS9 in which we have reasonable concept shifts. This mainly shows that concept shifts of such as in DS7 and DS9 can be quickly (only in couple of slides a our experiments shows) resolved by the BSBH. However for DS8, the concept shifts effects are visible due to the unrealistic changes in the data distribution at each concept shifts. As it can be seen

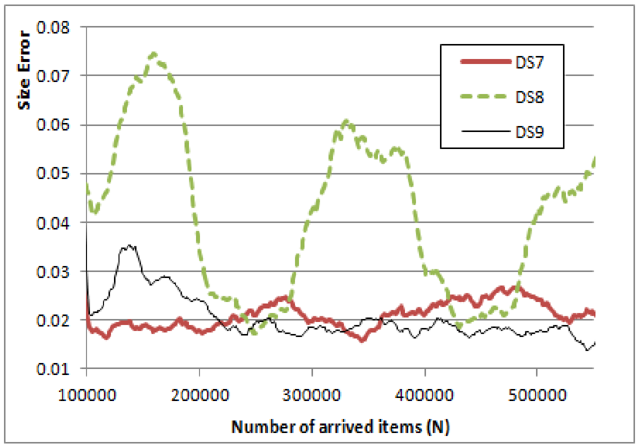


Figure 8.8: The size error of running BSBH-BL on data sets DS7 to DS9 through time. ($\phi = .7$, $W = 100K$, $k = 10$, $p = 7$)

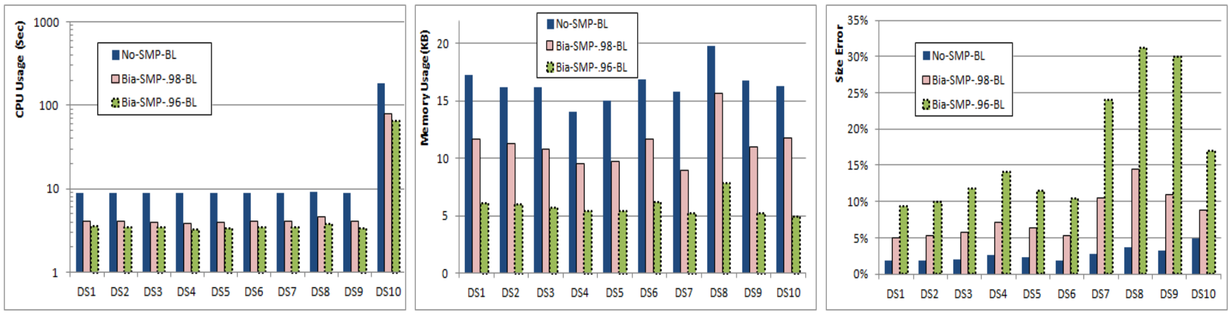


Figure 8.9: a) The execution time, b) memory usage, and c) size error of BSBH algorithms with different sampling rate for all data sets. ($\phi = .7$, $W = 100K$, $k = 10$, $p = 7$)

in the figure, the error increases right before the distribution changes from Zipf with $\alpha = 3.0$ to Zipf with $\alpha = 1.1$. The reason that the error is lower for $\alpha = 3.0$ is that BSBH receives less data items at the tail of the distribution of the current window under this case. This basically means that the boxes at the tail gradually shrink down in their size, and consequently they will not split but may merge. Since the merge operation does not impose any extra error, the overall error remains low for this case.

8.3.6 Discussion

Figure 8.9 depicts the performance of BSBH-BL without sampling as well as with biased sampling rates (μ) of 0.98 and 0.96 for all the data sets introduced in Table 8.1.

There are some important points which can be understood from the figure. Perhaps the most prominent element is that the execution time for BSBH is almost independent of the distribution of the data sets. As you can see in part (a) of Figure 8.9, the execution time of all the data sets is proportional to the size of the data sets. This verifies our Theorem 5 in previous section. As for the memory usage, part (b) of the figure suggests that the data distribution has a slight effect on the memory usage of BSBH. This is also in accordance with Theorem 4.

The other important point is that although the execution time and the memory usage of BSBH drops after sampling, this improvement is not very impressive especially for smaller sample rates. For instance with biased sampling rate of 0.96, which is equivalent to uniform sampling rate of 1.2%, we improve the memory usage and execution time at most by factors of 3 and 2 respectively, while degrading the accuracy by at least a factor of 7. This is actually not an unexpected result considering the compactness of our data structure. According to our experiments, partially shown in Figures 8.6, 8.7, and 8.9, not much is gained when we biased the sampling rate μ to be less than 0.98. Moreover, sampling on the data sets with concepts shifts has a worse effect on the accuracy of the results, as shown in part (c) of Figure 8.9.

8.4 Related Work

The problem of designing quantiles and equi-depth histograms in databases has been studied for a long time [PC84], [GMP97], [GKM02]. In 1984, Shapiro and Connel introduced a method to estimate the selectivity of conditions in the form of $attribute\ \theta\ constant$ in a database system where θ can be one of $=$, $<$, $>$, $\leq$, and $\geq$ [PC84]. In 2002, Gilbert *et al.* used *Random Subset Sums* (RSSs) as a sketch to store summarized information about the whole database and estimate the quantile with a one-pass algorithm [GKM02]. Gibbons *et al.* have also presented a sampling-based technique for maintaining approximate equi-depth histograms on relational databases [GMP97].

Existing works on designing equi-depth histograms over data streams have mainly focused on the related problem of quantiles. Since computing exact quantiles with a single-pass algorithm requires to store all the data [MP80], most of these works try to approximately answer quantile queries with low space complexity. Manku *et al.* introduced an ϵ-approximate algorithm to answer any quantile query over the entire history of the data stream [MRL98]. Later, they used non-uniform random sampling to improve their quantile computation for the case where the data set size is not known in advance [MRL99]. Greenwald and Khanna, in [GK01a], improved the memory usage of the previously mentioned approach. Their work , which is sometimes referred to as the GK algorithm, was used in [LLX04] and [AM04] to answer quantile queries over data streams with sliding windows. Both of these two approaches run several copies of the GK, and as a result they suffer from high time complexity.

Cormode *et al.* introduced the idea of biased quantiles in [CKM05]. Based on the GK algorithm, they proposed a deterministic ϵ-approximate algorithm to construct biased quantiles over the whole history of a data stream. Their approach needs $O(\frac{1}{\epsilon}B \times log(1/\phi)log(\epsilon N))$ space, where N is the data stream size and B is the number of boundaries. Note that as shown in [ZLX06], the worst case behavior of this type of algorithms is linear in the universe size (The size of the items' range). Later, in [CKM06], the same authors proposed a faster and more space-efficient deterministic algorithm for this problem, based on a binary tree structure idea borrowed from [SBA04]. Needing $O(\frac{1}{\epsilon}logUlog(\epsilon N))$ space and an almost constant amortized cost of actions per new entry, the algorithm is best suited for high speed data streams. However, it can not be easily employed in the sliding window model for data streams. Moreover, their algorithm requires prior knowledge of U and the bound on space requirement depends on the U which is undesirable. Zhang and Wang have used a decomposable structure to construct ϵ-approximate biased quantiles using $O(\frac{log^3(\epsilon N)}{\epsilon})$ space and $O(log(\frac{log(\epsilon N)}{\epsilon}))$ time. However, they used a naive non-uniform sampling technique which needs to sort thse entire data set in advance.

Unfortunately, the aforementioned biased quantile computation algorithms can not be easily used for the sliding window case, since their underlying structures do not support the idea of expirations. The usual solution for this issue is to run several copies of the same algorithms for different-sized chunks of the most recent part of the current window, and combine the results for the biggest possible parts which are not yet expired. A similar method is used in [AM04] for regular quantiles that is proven to be very slow [ZW07b, MZ11a].

CHAPTER 9

Conclusion and Future Work

Constantly growing massive data sets and data streams in the Web and various networks call for faster and more scalable accessing, querying, and analyzing approaches. Nowadays, many *Big Data* systems, such as the Word Wide Web, Web clicks, social networks, monitoring systems, stock market tickers, online encyclopedias, online reviewing systems, etc. are generating gigantic amount of data every second. For data sets of this size, traditional database management systems are ineffective, even when these systems take advantage of many well-studies summarization techniques such as sampling, histograms, and quantiles. In fact, these summarization techniques require major improvements and in some cases dramatic changes to answer the current needs in *Big Data* analysis and querying. This is mainly due to two challenging problems:

- The first challenge is that, in addition to summarization techniques for stored data, we now need online/continuous summaries for the streaming data, e.g., real-time online histograms. These summaries should be quickly updated to keep track of newly arrived data items and expiring ones to cope with fast-changing distributions of current data sets.
- The second challenge is that, the Web is storing large corpora of structured, semi-structured, and unstructured (free-text) documents, which is mostly presented in ambiguous natural languages. This situation has so far limited severely our ability of smart applications to use the information contained in Web pages, as needed to realize the grand vision of the "*Semantic Web*".

In order to address these two problems, we have introduced several efficient and effective summarization techniques for scalar and textual data. For scalar data, we have presented light and fast synopses, namely histograms, combined with various sampling approaches in order to implement more practical summarization techniques over massive data sets and data streams with sliding windows. These techniques can quickly provide a very accurate online histograms for fast data streams with sliding windows while leaving very small memory footprints.

As for the textual data, we have proposed a new knowledge-integration system, called *IKBstore*, which combines many of publicly available structured data into a superior knowledge bases. To improve the consistency of the integrated knowledge base, *IKBstore* employs a novel Context-Aware Synonym Suggestion System, CS^3, which is able to resolve many of the synonymous terms used across different knowledge bases. Since text is the most prevalent source of knowledge in the Web, *IKBstore* also completes structured summaries by mining free text. This task is performed by our new NLP-based system called *InfoBox Miner* or *IBminer*. The structured summaries so produced are in the RDF form of <*subject*, *attribute*, *value*> triples specifying one or more *value*s for each *attribute* of a *subject*. In addition to possible reduction of the volume of the data, these triples provide the knowledge in a structured format that can be effectively used in semantic search and question answering systems through structured queries, as exploited in this dissertation.

9.1 Future Research Directions

Toward Domain-Specific Knowledge Base Management Systems: So far, we showed how our project aims at building a superior system for providing typical Web users with high quality knowledge bases which can be used for precise querying. Here, we discuss an important line of future work and explain how our project can help expert users in advanced applications focused on more specific domains. In such applications, users and curators need to manage datasets that might contain structured, semi-structured, and unstructured data, and cover deeper knowledge on more specific topics than what a general encyclopedia such as Wikipedia can cover. For instance, consider a medical center where information about patients, physicians, staff, drugs and treatments is usually available in many different formats such as plain text, forms, images, tables, and structured information. In addition to the many sources of information, a tremendous number of medical articles are being published every year in different journals. Although this amount of information provides a significant opportunity for advancing medical research and patients' treatment, the complexity and heterogeneity of the data makes this goal difficult to achieve. As a matter of a fact, researchers often end-up redoing the very same study, since they could not easily identify previous work through the query systems and the search engines they have currently available. Moreover, it is easy to recognize that similar scenarios occur in many other applications domains, including education, natural sciences, history [BBC12], tourism, and entertainment or in even more specific domains such as materials and their properties [ASM14], global terrorism [PGT12], gene-protein connections, naval ship handling [Bar05], musics, movies, cooking [all14, coo14], gardening[gar14], and so on.

Thus, we need to extend the document mining and querying technology we have developed on Wikipedia to provide methods and systems for building and managing domain-specific knowledge bases and applications. In many ways, this extension can be the first system for automating domain-specific knowledge base management. The basic approach consists in using available structured information to create the initial KB for the domain of interest. Fortunately, most of the domains already provide a considerable amount of structured data. Structured information can also be found in specialized encyclopedias (e.g., MedlinePlus [Med] or Disease Database [Dis]) and more general encyclopedias (e.g. those in medical-related pages of Wikipedia). Once the initial KB is ready, we will employ *IBminer* to annotate and extract structured information from the text at hand (e.g., documents on patients, diseases, treatments, etc.) as discussed in Chapter 3. For domain specific applications, *IBminer* will be assisted by domain-specific ontologies that will improve its focus and performance on the domain of interest. Rich ontologies can be generated by starting from seed ontologies in the domain of interest (even small ones can do) and then building them up using text-mining, by the techniques outlined in Chapter 3.

Relying on ontologies and its relation-extraction patterns, *IBminer* can also annotate the documents by recognizing entities and their relations in the text. As described in Section 5.4, this process significantly reduces the human effort needed for structuring documents—giving users more time to improve their knowledge bases through the facilities provided by IBE and its user-friendly interface. Thus, it is quite reasonable to expect that users and authors will start providing structured summaries of their documents or articles by first feeding them to an *IBminer*-like system, and then verifying and completing the results so generated in the form of InfoBoxes or annotations. This will significantly speed-up the knowledge extraction&summarization process even for large corpora, particularly in scientific fields where authors are highly motivated to having their papers published, retrieved and read by other researchers who can then reference them (h-index, ect.). In the longer term this will teach users how to provide article (e.g. Wikipedia pages) in a way that are less ambiguous and thus more amenable to text mining approaches such as *IBminer*.

Supporting InfoBox Analytics: Although presenting the data in structured format makes its analysis much easier, expert users still need stronger analytical tools in order to perform InfoBox analytics. For instance, in our medical example above, it is very desirable to automatically classify patients based on their record history and current symptoms. Another future direction then would be to equip our system with most common analytics over the structured data. Covering classification, clustering, outlier detection, and frequent pattern extraction can be a very good start. Opinion Mining, Topic identification, document categorization, document summarization, etc. are aslo several other applications that can fit this line of future studies.

Information Confidence and the Provenance: The quality of knowledge in Semantic Web is very important for semantic applications, especially for semantic search systems. To assess the quality of knowledge, we need to archive the provenance for every fact in knowledge base. In reality, different applications may use different types of quality measures, such as the confidence value, evidence frequency, lineage, etc. Thus, we propose a provenance extension of *IKBstore* which annotates every fact with a structure called *provenance polynomial* to preserve these important properties of knowledge. There have been several models for capturing provenance in Semantic Web [TFK11, MWF07]. We extend the abstract provenance model discussed in [TFK11] to archive knowledge provenance. The process of capturing knowledge provenance consists of two main phases: (i) constructing the provenance polynomial [GKT07] for every fact; (ii) materializing the polynomial for each type of provenance. The advantage of such model is the generality. It does not only work for specific provenance, but also any provenance that can be expressed by provenance polynomial, such as confidence value and reliability used in NELL [CBK10a] and *IKBstore* system. In this way, we just need to perform a single-round evaluation to construct the expression of provenance process. Then, *IKBstore* can evaluate various types of provenance by computing the provenance value based on corresponding provenance polynomial.

Mining Semi-structured Data: Tables and lists are used frequently in documents, since they are easy for humans to interpret. They can also be easier for machines to interpret than unstructured data, if their structure is known to them. However this is not often the case, since the structure of the table is either explained in text or it is inferable by humans form context and commonsense knowledge. In this project, to be able to extract information from these sources, we use most common patterns to convert them into the triple format (similar to semantic links generated by the first step in *IBminer*'s process) and follow the exact remaining steps to translate them to our KBs terminology. Notice that structured information about entities that can not be easily found in the text (e.g., for Zip code, area code, elevation, etc. for a given location), can be extracted from some semi-structured or structured specialized sources, (e.g., telephone book). Thus, we interpret the internal structure of such sources as well as convert them into our knowledge graph.

Supports for Cross-Language Data Sets: Many data sources provide their data in different languages. For instance Wikipedia supports more than 270 languages. Recently, the WikiData projects is lounched to collect and unify the information in all languages in Wikipedia mostly with the assist of volunteer contributors [Wik]. This will improve the final knowledge base since each language may cover knowledge that others are missing. For instance, it is more probable to find useful information about Italian cities in the Italian Wikipedia rather than in Wikipedia for English or other languages. Although this is a very appealing vision and important step toward the "Semantic Wikipedia", we believe automatic techniques are necessary to further assist the contributors and expedite this unification process. To this end, we extend our context-aware synonym generation system to find attribute synonyms among different languages. In other words, we interpret the attribute translation task as a synonym matching one. In this way, our system will be able to use already matched attributes in Wikidata and learn appropriate patterns, as *IBminer* does, and use these patterns to translate more triples from different languages. Therefore, the contributing user only need to verify the suggested results and slightly improve the mistranslated ones.

Scaling the System: Although the current *IKBstore* contains several large scale KBs, e.g. DBpedia and YaGo2, we expect considerable improvements in its coverage and accuracy by analyzing more text from the Web. Currently we are in the process of mining the entire Wikipedia using available clusters at UCLA. Our current estimation is that this will at least doubles the size of the current KB. To be able to mine knowledge from even larger text corpora, we distribute *IBminer*'s steps (Chapter 3) to Map-Reduce-like jobs over several nodes in the cluster. The most important challenge is the large number of intermediate triples and patterns. We are seeking techniques for early filtering of the triples and reducing the patterns before the final aggregation.

As the KB will approach its final size, faster approaches for answering queries should be provided. Toward this goal, one should investigate the performance of distributed SPARQL search engines such as Virtuoso and Apache Jena under larger KBs and more simultaneous queries. The impact of different combinations of exiting optimization, caching, indexing, and estimation techniques on the time performance of the search engine should be carefully studied as well.

The other aspect of a scalable search engine is the number of queries it can handle at the same time. Fortunately, the users' requests are usually concentrated on a limited number of very frequent queries. Inspired from this fact, a dynamic caching technique can be designed, which learns the most frequent star-shaped subqueries from the current query logs and caches the intermediate results for these subqueries. In this way, next time a user submits a query containing one of these subqueries, the results will be on hand. The reason we select the star-shaped sub-queries is that they can be found in the query graphs in polynomial time. This also makes the problem of frequent subgraph mining a lot faster. In addition to the caching techniques, replication and partitioning approaches for supporting more concurrent queries should be studied.

References

[ADL05] Noga Alon, Nick G. Duffield, Carsten Lund, and Mikkel Thorup. "Estimating arbitrary subset sums with few probes." In *PODS*, pp. 317–325, 2005.

[Agg06] Charu C. Aggarwal. "On Biased Reservoir Sampling in the Presence of Stream Evolution." In *VLDB*, pp. 607–618, 2006.

[Agr07] C. Agrawal. *Data Streams: Models and Algorithms*. Springer Verlag, 2007.

[all14] "Allrecipes - Recipes and cooking confidence for home cooks." http://allrecipes.com/, 2014.

[AM04] Arvind Arasu and Gurmeet Manku. "Approximate Counts and Quantiles over Sliding Windows." In *PODS*, pp. 286–296, 2004.

[AM06] Rajeev Alur and P. Madhusudan. "Adding Nesting Structure to Words." In *Developments in Language Theory*, pp. 1–13, 2006.

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments." In *STOC*, pp. 20–29, 1996.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments." *J. Comput. Syst. Sci.*, **58**(1):137–147, 1999.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. "Fast Algorithms for Mining Association Rules in Large Databases." In *VLDB*, pp. 487–499, 1994.

[ASM14] "ASM Handbooks Online." http://products.asminternational.org/hbk/index.jsp, 2014.

[AT10] Witold Abramowicz and Robert Tolksdorf, editors. *Business Information Systems, 13th International Conference, BIS 2010, Berlin, Germany, May 3-5, 2010. Proceedings*, volume 47 of *Lecture Notes in Business Information Processing*. Springer, 2010.

[AZ12] Maurizio Atzori and Carlo Zaniolo. "SWiPE: searching wikipedia by example." In *WWW*, pp. 309–312, 2012.

[Bar05] James Alden Barber. *Naval Shiphandler's Guide*. Naval Institute Press, 2005.

[BBC12] Roger S. Bagnall, Kai Brodersen, Craige B. Champion, Andrew Erskine, and Sabine R. Huebner. *The Encyclopedia of Ancient History*. Wiley, 2012.

[BBD02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. "Models and Issues in Data Stream Systems." In *PODS*, pp. 1–16, 2002.

[BCS07] Michele Banko, Michael J Cafarella, Stephen Soderl, Matt Broadhead, and Oren Etzioni. "Open information extraction from the web." In *IJCAI*, 2007.

[BDM02] Brian Babcock, Mayur Datar, and Rajeev Motwani. "Sampling from a moving window over streaming data." In *SODA*, pp. 633–634, 2002.

[BEP08] Kurt D. Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. "Freebase: a collaboratively created graph database for structuring human knowledge." In *SIGMOD*, pp. 1247–1250, 2008.

[BKS01] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. "The Skyline Operator." In *ICDE*, pp. 421–430, 2001.

[BLK09] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. "DBpedia - A crystallization point for the Web of Data." *J. Web Sem.*, **7**(3):154–165, 2009.

[BO07] Vladimir Braverman and Rafail Ostrovsky. "Smooth Histograms for Sliding Windows." In *FOCS*, pp. 283–293, 2007.

[BO10] Vladimir Braverman and Rafail Ostrovsky. "Effective Computations on Sliding Windows." *SIAM J. Comput.*, **39**(6):2113–2131, 2010.

[Bou92] Didier Bourigault. "Surface Grammatical Analysis For The Extraction Of Terminological Noun Phrases." In *COLING*, pp. 977–981, 1992.

[BOZ07] Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. "Succinct Sampling on Streams." *CoRR*, **abs/cs/0702151**, 2007.

[BOZ09] Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. "Optimal sampling from sliding windows." In *PODS*, pp. 147–156, 2009.

[BW01] Shivnath Babu and Jennifer Widom. "Continuous Queries over Data Streams." *SIGMOD Record*, **30**(3):109–120, 2001.

[Cas] "Apache Cassandra." `http://cassandra.apache.org/`.

[CBK10a] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E.R. Hruschka Jr., and T.M. Mitchell. "Toward an Architecture for Never-Ending Language Learning." In *AAAI*, pp. 1306–1313, 2010.

[CBK10b] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. "Toward an Architecture for Never-Ending Language Learning." In *AAAI*, 2010.

[CCC12] Kaushik Chakrabarti, Surajit Chaudhuri, Tao Cheng, and Dong Xin. "A framework for robust discovery of entity synonyms." In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pp. 1384–1392, New York, NY, USA, 2012. ACM.

[CdJ10] Daniel Cer, Marie-Catherine de Marneffe, Daniel Jurafsky, and Christopher D. Manning. "Parsing to Stanford Dependencies: Trade-offs between speed and accuracy." In *7th International Conference on Language Resources and Evaluation (LREC 2010)*, 2010.

[CE09] Eugene Charniak and Micha Elsner. "EM works for pronoun anaphora resolution." In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, pp. 148–156, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[CGG03] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. "Skyline with Presorting." In *ICDE*, pp. 717–816, 2003.

[CGR00] Kaushik Chakrabarti, Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. "Approximate Query Processing Using Wavelets." In *VLDB*, pp. 111–122, 2000.

[CGX09a] Surajit Chaudhuri, Venkatesh Ganti, and Dong Xin. "Exploiting web search to generate synonyms for entities." In *Proceedings of the 18th international conference on World wide web*, WWW '09, pp. 151–160, New York, NY, USA, 2009. ACM.

[CGX09b] Surajit Chaudhuri, Venkatesh Ganti, and Dong Xin. "Mining document collections to facilitate accurate approximate entity matching." *Proc. VLDB Endow.*, **2**(1):395–406, August 2009.

[Cha08] "Charniak NLP Parser." `ftp://ftp.cs.brown.edu/pub/nlparser/`, 2008.

[CKM05] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. "Effective Computation of Biased Quantiles over Data Streams." In *ICDE*, pp. 20–31, 2005.

[CKM06] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. "Space- and time-efficient deterministic algorithms for biased quantiles over data streams." In *PODS*, pp. 263–272, 2006.

[CKM08] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. "Finding hierarchical heavy hitters in streaming data." *TKDD*, **1**(4), 2008.

[CL03] Joong Hyuk Chang and Won Suk Lee. "Finding recent frequent itemsets adaptively over online data streams." In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 487–492, 2003.

[CLP10] Tao Cheng, Hady Wirawan Lauw, and Stelios Paparizos. "Fuzzy matching of Web queries to structured data." In *ICDE*, pp. 713–716, 2010.

[CLP12] Tao Cheng, Hady Wirawan Lauw, and Stelios Paparizos. "Entity Synonyms for Structured Web Search." *IEEE Trans. Knowl. Data Eng.*, **24**(10):1862–1875, 2012.

[CM99] Surajit Chaudhuri and Rajeev Motwani. "On Sampling and Relational Operators." *IEEE Data Eng. Bull.*, **22**(4):41–46, 1999.

[CM05] Graham Cormode and S. Muthukrishnan. "An improved data stream summary: the count-min sketch and its applications." *J. Algorithms*, **55**(1):58–75, 2005.

[CMH09] Michael J. Cafarella, Jayant Madhavan, and Alon Halevy. "Web-scale extraction of structured data." *SIGMOD Rec.*, **37**(4):55–61, March 2009.

[CMN99] Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. "On Random Sampling over Joins." In *SIGMOD Conference*, pp. 263–274, 1999.

[Coo71] Stephen A. Cook. "The Complexity of Theorem-Proving Procedures." In *STOC*, pp. 151–158, 1971.

[coo14] "Recipe Search and More." http://www.cooks.com/, 2014.

[DG08] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Commun. ACM*, **51**(1):107–113, January 2008.

[DGI02a] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. "Maintaining Stream Statistics over Sliding Windows." *SIAM J. Comput.*, **31**(6):1794–1813, 2002.

[DGI02b] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. "Maintaining Stream Statistics over Sliding Windows." *SIAM J. Comput.*, **31**(6):1794–1813, 2002.

[Dis] "Diseases Database Ver 1.9." http://www.diseasesdatabase.com/.

[DJM02] Tamraparni Dasu, Theodore Johnson, S. Muthukrishnan, and Vladislav Shkapenyuk. "Mining database structure; or, how to build a data quality browser." In *SIGMOD Conference*, pp. 240–251, 2002.

[Dro03] Patrick Drouin. "Term extraction using non-technical corpora as a point of leverage." *TERMINOLOGY*, **9**:99–116, 2003.

[DZP10] Euthymios Drymonas, Kalliopi Zervanou, and Euripides G. M. Petrakis. "Unsupervised Ontology Acquisition from Plain Texts: The *OntoGain* System." In *NLDB*, pp. 277–287, 2010.

[ECD04] Oren Etzioni, Michael J. Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. "Web-scale information extraction in knowitall: (preliminary results)." In *WWW*, pp. 100–110, 2004.

[EG06] C. Elkan and R. Greiner. "Building large knowledge-based systems: representation and inference in the Cyc project." *Artificial Intelligence*, **61**(1):41–52, 2006.

[EG13] C. Elkan and R. Greiner. "Inside YAGO2s: A Transparent Information Extraction Architecture." *Artificial Intelligence*, 2013.

[FKS99] Joan Feigenbaum, Sampath Kannan, Martin Strauss, and Mahesh Viswanathan. "An Approximate L^1-Difference Algorithm for Massive Data Streams." In *FOCS*, pp. 501–511, 1999.

[FLG87] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. "The vocabulary problem in human-system communication." *Commun. ACM*, **30**(11):964–971, November 1987.

[FM85] Philippe Flajolet and G. Nigel Martin. "Probabilistic Counting Algorithms for Data Base Applications." *J. Comput. Syst. Sci.*, **31**(2):182–209, 1985.

[FSG98] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman. "Computing Iceberg Queries Efficiently." In *VLDB*, pp. 299–310, 1998.

[gar14] "Gardening Resources: National Gardening Association." http://www.garden.org/, 2014.

[GEO] "GeoNames." http://www.geonames.org.

[GGR02] Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi. "Querying and mining data streams: you only get one look a tutorial." In *SIGMOD Conference*, p. 635, 2002.

[Gib01] Phillip B. Gibbons. "Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports." In *VLDB*, pp. 541–550, 2001.

[GK01a] Michael Greenwald and Sanjeev Khanna. "Space-Efficient Online Computation of Quantile Summaries." In *SIGMOD Conference*, pp. 58–66, 2001.

[GK01b] Michael Greenwald and Sanjeev Khanna. "Space-Efficient Online Computation of Quantile Summaries." In *SIGMOD Conference*, pp. 58–66, 2001.

[GK06] Andrew Gregorowicz and Mark A. Kramer. "Mining a Large-Scale Term-Concept Network from Wikipedia." Technical report, Mitre, 2006.

[GKM01] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. "Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries." In *VLDB*, pp. 79–88, 2001.

[GKM02] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. "How to Summarize the Universe: Dynamic Maintenance of Quantiles." In *VLDB*, pp. 454–465, 2002.

[GKM03] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. "One-Pass Wavelet Decompositions of Data Streams." *IEEE Trans. Knowl. Data Eng.*, **15**(3):541–554, 2003.

[GKS01] Sudipto Guha, Nick Koudas, and Kyuseok Shim. "Data-streams and histograms." In *STOC*, pp. 471–475, 2001.

[GKT07] Todd J. Green, Gregory Karvounarakis, and Val Tannen. "Provenance semirings." In *PODS*, 2007.

[GM98] Phillip B. Gibbons and Yossi Matias. "New Sampling-Based Summary Statistics for Improving Approximate Query Answers." In *SIGMOD Conference*, pp. 331–342, 1998.

[GM99] Phillip B. Gibbons and Yossi Matias. "Synopsis Data Structures for Massive Data Sets." In *SODA*, pp. 909–910, 1999.

[GM07] Sumit Ganguly and Anirban Majumder. "CR-precis: A Deterministic Summary Structure for Update Data Streams." In *ESCAPE*, pp. 48–59, 2007.

[GMP97] Phillip B. Gibbons, Yossi Matias, and Viswanath Poosala. "Fast Incremental Maintenance of Approximate Histograms." In *VLDB*, pp. 466–475, 1997.

[Gre96] Michael Greenwald. "Practical Algorithms for Self Scaling Histograms or Better than Average Data Collection." *Perform. Eval.*, **27/28**(4):19–40, 1996.

[Gru93] T. R. Gruber. "A Translation Approach to Portable Ontology Specifications." *Knowledge Acquisition*, **6**:199–220, 1993.

[GW99] Bernhard Ganter and Rudolf Wille. *Formal concept analysis - mathematical foundations*. Springer, 1999.

[GZK05] Mohamed Medhat Gaber, Arkady B. Zaslavsky, and Shonali Krishnaswamy. "Mining data streams: a review." *SIGMOD Record*, **34**(2):18–26, 2005.

[HBS10] Rasmus Hahn, Christian Bizer, Christopher Sahnwaldt, Christian Herta, Scott Robinson, Michaela Bürgle, Holger Düwiger, and Ulrich Scheel. "Faceted Wikipedia Search." In *BIS*, pp. 1–11, 2010.

[Hea92] Marti A. Hearst. “Automatic Acquisition of Hyponyms from Large Text Corpora.” In *COLING*, pp. 539–545, 1992.

[HHW97] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. “Online Aggregation.” In *SIGMOD Conference*, pp. 171–182, 1997.

[HK09] Aria Haghighi and Dan Klein. “Simple Coreference Resolution with Rich Syntactic and Semantic Features.” In *EMNLP*, pp. 1152–1161, 2009.

[HKY09] Felix Halim, Panagiotis Karras, and Roland H. C. Yap. “Fast and effective histogram construction.” In *CIKM*, pp. 1167–1176, 2009.

[Hob78] Jerry Hobbs. “Resolving Pronoun References.” *Lingua*, **44**:311–338, 1978.

[Hof] “Hoffman2 Cluster, UCLA.” http://hpc.ucla.edu/hoffman2/.

[HP10] Dan He and D. Stott Parker. “Topic dynamics: an alternative model of bursts in streams of topics.” In *KDD*, pp. 443–452, 2010.

[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. “Mining Frequent Patterns without Candidate Generation.” In *SIGMOD Conference*, pp. 1–12, 2000.

[HSB11] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, Edwin Lewis-Kelham, Gerard de Melo, and Gerhard Weikum. “YAGO2: exploring and querying world knowledge in time, space, context, and many languages.” In *WWW*, pp. 229–232, 2011.

[HSB13] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. “YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia.” *Artif. Intell.*, **194**:28–61, 2013.

[HZW10] Raphael Hoffmann, Congle Zhang, and Daniel S. Weld. “Learning 5000 Relational Extractors.” In *ACL*, pp. 286–295, 2010.

[Ioa03] Yannis E. Ioannidis. “The History of Histograms (abridged).” In *VLDB*, pp. 19–30, 2003.

[IP99] Yannis E. Ioannidis and Viswanath Poosala. “Histogram-Based Approximation of Set-Valued Query-Answers.” In *VLDB*, pp. 174–185, 1999.

[Jen] “Apache Jena.” `http://jena.apache.org/`.

[JK08] Maciej Janik and Krys Kochut. “Training-less Ontology-based Text Categorization.” In *Workshop on ESAIR*, March 2008.

[JKM98] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. “Optimal Histograms with Quality Guarantees.” In *VLDB*, pp. 275–286, 1998.

[JT10] Xing Jiang and Ah-Hwee Tan. “CRCTOL: A semantic-based domain ontology learning system.” *JASIST*, **61**(1):150–168, 2010.

[KH10] Zornitsa Kozareva and Eduard H. Hovy. "A Semi-Supervised Method to Learn and Construct Taxonomies Using the Web." In *EMNLP*, pp. 1110–1118, 2010.

[KKR08] Karin Kipper, Anna Korhonen, Neville Ryant, and Martha Palmer. "A large-scale classification of English verbs." *Language Resources and Evaluation*, **42**(1):21–40–40, March 2008.

[Kle02] Jon M. Kleinberg. "Bursty and hierarchical structure in streams." In *KDD*, pp. 91–101, 2002.

[KM11] Jayant Krishnamurthy and Tom M. Mitchell. "Which noun phrases denote which concepts?" In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pp. 570–580, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[LBN10] Dustin Lange, Christoph Böhm, and Felix Naumann. "Extracting structured information from Wikipedia articles to populate infoboxes." In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pp. 1661–1664, New York, NY, USA, 2010. ACM.

[LKK07] Chang-Shing Lee, Yuan-Fang Kao, Yau-Hwang Kuo, and Mei-Hui Wang. "Automated ontology construction for unstructured text documents." *Data Knowl. Eng.*, **60**(3):547–566, March 2007.

[LL94] Shalom Lappin and Herbert J. Leass. "An algorithm for pronominal anaphora resolution." *Comput. Linguist.*, **20**(4):535–561, December 1994.

[LLX04] Xuemin Lin, Hongjun Lu, Jian Xu, and Jeffrey Xu Yu. "Continuously Maintaining Quantile Summaries of the Most Recent N Elements over a Data Stream." In *ICDE*, pp. 362–374, 2004.

[LMT05] Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, and Peter A. Tucker. "No pane, no gain: efficient evaluation of sliding-window aggregates over data streams." *SIGMOD Record*, **34**(1):39–44, 2005.

[LP01] Dekang Lin and Patrick Pantel. "DIRT @SBT@discovery of inference rules from text." In *KDD*, pp. 323–328, 2001.

[LWO00] Stanley Loh, Leandro Krug Wives, and José Palazzo M. de Oliveira. "Concept-based knowledge discovery in texts extracted from the Web." *SIGKDD Explor. Newsl.*, **2**(1):29–39, June 2000.

[LWW11] Taesung Lee, Zhongyuan Wang, Haixun Wang, and Seung won Hwang. "Web Scale Taxonomy Cleansing." *PVLDB*, **4**(12):1295–1306, 2011.

[MBD03] Brian Babcock Mayur, Brian Babcock, Mayur Datar, and Rajeev Motwani. "Load Shedding Techniques for Data Stream Systems." In *In Proc. of the 2003 Workshop on Management and Processing of Data Streams (MPDS)*, 2003.

[MBP06] Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. "Continuous monitoring of top-k queries over sliding windows." In *SIGMOD Conference*, pp. 635–646, 2006.

[MD88] M. Muralikrishna and David J. DeWitt. "Equi-Depth Histograms For Estimating Selectivity Factors For Multi-Dimensional Queries." In *SIGMOD Conference*, pp. 28–36, 1988.

[Med] "MedlinePlus - Health Information from the National Library of Medicine." http://www.nlm.nih.gov/medlineplus/.

[MGZ13a] Hamid Mousavi, Shi Gao, and Carlo Zaniolo. "Discovering Attribute and Entity Synonyms for Knowledge Integration and Semantic Web Search." *3rd International Workshop on Sematic Search over The Web*, 2013.

[MGZ13b] Hamid Mousavi, Shi Gao, and Carlo Zaniolo. "Discovering Attribute and Entity Synonyms for Knowledge Integration and Semantic Web Search." In *CSD Technical Report #130013, UCLA*, 2013.

[MGZ13c] Hamid Mousavi, Shi Gao, and Carlo Zaniolo. "IBminer: A Text Mining Tool for Constructing and Populating InfoBox Databases and Knowledge Bases." *PVLDB*, **6**(12):1330–1333, 2013.

[MKI11a] Hamid Mousavi, Deirdre Kerr, and Markus Iseli. "A New Framework for Textual Information Mining over Parse Trees." In *ICSC*, 2011.

[MKI11b] Hamid Mousavi, Deirdre Kerr, and Markus Iseli. "A New Framework for Textual Information Mining over Parse Trees." In *(CRESST Report 775). University of California, Los Angeles*, 2011.

[MKI13a] Hamid Mousavi, Deirdre Kerr, Markus Iseli, and Carlo Zaniolo. "Deducing InfoBoxes from Unstructured Text in Wikipedia Pages." In *CSD Technical Report #130001), UCLA*, 2013.

[MKI13b] Hamid Mousavi, Deirdre Kerr, Markus Iseli, and Carlo Zaniolo. "OntoHarvester: An Unsupervised Ontology Generator from Free Text." In *CSD Technical Report #130003), UCLA*, 2013.

[MML09] Olena Medelyan, David N. Milne, Catherine Legg, and Ian H. Witten. "Mining meaning from Wikipedia." *Int. J. Hum.-Comput. Stud.*, **67**(9):716–754, 2009.

[MMM06] Marie De Marneffe, Bill Maccartney, and Christopher D. Manning. "Generating typed dependency parses from phrase structure parses." In *In LREC*, 2006.

[MP80] J. Ian Munro and Mike Paterson. "Selection and Sorting with Limited Storage." *Theor. Comput. Sci.*, **12**:315–323, 1980.

[MRL98] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. "Approximate Medians and other Quantiles in One Pass and with Limited Memory." In *SIGMOD Conference*, pp. 426–435, 1998.

[MRL99] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. "Random Sampling Techniques for Space Efficient Online Computation of Order Statistics of Large Datasets." In *SIGMOD Conference*, pp. 251–262, 1999.

[MRS11] Y. Mass, M. Ramanath, Y. Sagiv, and G. Weikum. "Iq: The case for iterative querying for knowledge." In *CIDR*, 2011.

[MS00] Alexander Maedche and Steffen Staab. "Semi-automatic engineering of ontologies from text." In *Proc. of 12th Int. Conf. on Software and Knowledge Engineering*, Chicago, IL, 2000.

[MUS] "MusicBrainz." http://musicbrainz.org.

[Mut03] S. Muthukrishnan. "Data streams: algorithms and applications." In *SODA*, pp. 413–413, 2003.

[MWF07] Simon Miles, Sylvia C. Wong, Weijian Fang, Paul Groth, Klaus-Peter Zauner, and Luc Moreau. "Provenance-based validation of e-science experiments." *Web Semant.*, **5**(1), March 2007.

[MZ10] Barzan Mozafari and Carlo Zaniolo. "Optimal load shedding with aggregates and mining queries." In *ICDE*, pp. 76–88, 2010.

[MZ11a] Hamid Mousavi and Carlo Zaniolo. "Fast and accurate computation of equi-depth histograms over data streams." In *EDBT*, pp. 69–80, 2011.

[MZ11b] Hamid Mousavi and Carlo Zaniolo. "Fast and Space-Efficient Computation of Equi-Depth Histograms for Data Streams." In *EDBT 2011 Joint Conference, Uppsala, Sweden*, pp. 100–112, 2011.

[MZ13] Hamid Mousavi and Carlo Zaniolo. "Fast computation of approximate biased histograms on sliding windows over data streams." In *SSDBM*, p. 13, 2013.

[Nav01] Gonzalo Navarro. "A guided tour to approximate string matching." *ACM Comput. Surv.*, **33**(1):31–88, March 2001.

[NMI07] Dat P. T Nguyen, Yutaka Matsuo, and Mitsuru Ishizuka. "Exploiting syntactic and semantic information for relation extraction from wikipedia." In *In IJCAI07-TextLinkWS*, 2007.

[NS07] David Nadeau and Satoshi Sekine. "A survey of named entity recognition and classification." *Lingvisticae Investigationes*, **30**(1):3–26, January 2007.

[NTW11] Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum. "Scalable knowledge harvesting with high precision and high recall." WSDM '11, pp. 227–236, New York, NY, USA, 2011. ACM.

[NV10] Roberto Navigli and Paola Velardi. "Learning Word-Class Lattices for Definition and Hypernym Extraction." In *ACL*, pp. 1318–1327, 2010.

[NVF11] Roberto Navigli, Paola Velardi, and Stefano Faralli. "A Graph-Based Algorithm for Inducing Lexical Taxonomies from Scratch." In *IJCAI*, pp. 1872–1877, 2011.

[NZR12] Feng Niu, Ce Zhang, Christopher Re, and Jude W. Shavlik. "DeepDive: Web-scale Knowledge-base Construction using Statistical Learning and Inference." In *VLDS*, pp. 25–28, 2012.

[OPE] "OPENCYC." http://www.cyc.com/platform/opencyc.

[PC84] Gregory Piatetsky-Shapiro and Charles Connell. "Accurate Estimation of the Number of Tuples Satisfying a Condition." In *SIGMOD Conference*, pp. 256–276, 1984.

[PCB09] Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. "Web-scale distributional similarity and entity set expansion." In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, EMNLP '09, pp. 938–947, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[PD08] Hoifung Poon and Pedro Domingos. "Joint Unsupervised Coreference Resolution with Markov Logic." In *EMNLP*, pp. 650–659, 2008.

[PD09] Hoifung Poon and Pedro Domingos. "Unsupervised semantic parsing." In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, EMNLP '09, pp. 1–10, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[PD10] Hoifung Poon and Pedro Domingos. "Unsupervised Ontology Induction from Text." In *ACL*, pp. 296–305, 2010.

[PGR10] Aditya G. Parameswaran, Hector Garcia-Molina, and Anand Rajaraman. "Towards The Web of Concepts: Extracting Concepts from Large Datasets." *PVLDB*, **3**(1):566–577, 2010.

[PGT12] "Patterns of Global Terrorism." http://onlinebooks.library.upenn.edu/webbin/serial?id=patglobalterror, 2012.

[PI96] Viswanath Poosala and Yannis E. Ioannidis. "Estimation of Query-Result Distribution and its Application in Parallel-Join Load Balancing." In *VLDB*, pp. 448–459, 1996.

[PIH96] Viswanath Poosala, Yannis E. Ioannidis, Peter J. Haas, and Eugene J. Shekita. "Improved Histograms for Selectivity Estimation of Range Predicates." In *SIGMOD Conference*, pp. 294–305, 1996.

[PL01] Patrick Pantel and Dekang Lin. "A Statistical Corpus-Based Term Extractor." In *Canadian Conference on AI*, pp. 36–46, 2001.

[PTF05] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. "Progressive skyline computation in database systems." *ACM Trans. Database Syst.*, **30**(1):41–82, 2005.

[QAA02] Lin Qiao, Divyakant Agrawal, and Amr El Abbadi. “RHist: adaptive summarization over continuous data streams.” In *CIKM*, pp. 469–476, 2002.

[QHF04] T.T. Quan, S.C. Hui, ACM Fong, and T.H. Cao. “Automatic generation of ontology for scholarly semantic web.” *Proceedings of The Semantic Web–ISWC*, pp. 726–740, 2004.

[RLR10] Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nate Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher D. Manning. “A Multi-Pass Sieve for Coreference Resolution.” In *EMNLP*, pp. 492–501, 2010.

[SA96] Ramakrishnan Srikant and Rakesh Agrawal. “Mining Quantitative Association Rules in Large Relational Tables.” In *SIGMOD Conference*, pp. 1–12, 1996.

[SAC79] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. “Access Path Selection in a Relational Database Management System.” In *SIGMOD Conference*, pp. 23–34, 1979.

[SBA04] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. “Medians and Beyond: New Aggregation Techniques for Sensor Networks.” *CoRR*, **cs.DC/0408039**, 2004.

[SC08] Veselin Stoyanov and Claire Cardie. “Topic identification for fine-grained opinion analysis.” In *Proceedings of COLING*, pp. 817–824, Stroudsburg, PA, USA, 2008.

[Sem] “Semantic Web Information Management System (SWIMS).” http://semscape.cs.ucla.edu.

[SKW08] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. “YAGO: A Large Ontology from Wikipedia and WordNet.” *J. Web Sem.*, **6**(3):203–217, 2008.

[SLM02] Push Singh, Thomas Lin, Erik T. Mueller, Grace Lim, Travell Perkins, and Wan Li Zhu. “Open Mind Common Sense: Knowledge Acquisition from the General Public.” In *Confederated International Conferences DOA, CoopIS and ODBASE*, London, UK, 2002.

[Sno06] Rion Snow. “Semantic taxonomy induction from heterogenous evidence.” In *In Proceedings of COLING/ACL 2006*, pp. 801–808, 2006.

[SPA08] “SPARQL Query Language for RDF.” http://www.w3.org/TR/rdf-sparql-query/, 2008.

[SPA12] “SPARQL Query Language for RDF.” http://www.w3.org/TR/rdf-sparql-query/, 2012.

[SR98] Michael M. Stark and Richard F. Riesenfeld. “WordNet: An Electronic Lexical Database.” In *Proceedings of 11th Eurographics Workshop on Rendering*. MIT Press, 1998.

[SSW09] Fabian M. Suchanek, Mauro Sozio, and Gerhard Weikum. “SOFIE: a self-organizing framework for information extraction.” In *WWW*, pp. 631–640, 2009.

[Sta14] "The Stanford Parser: A statistical parser." http://nlp.stanford.edu/software/lex-parser.shtml, 2014.

[TcZ07] Nesime Tatbul, Ugur Çetintemel, and Stanley B. Zdonik. "Staying FIT: Efficient Load Shedding Techniques for Distributed Stream Processing." In *VLDB*, pp. 159–170, 2007.

[TFK11] Yannis Theoharis, Irini Fundulaki, Grigoris Karvounarakis, and Vassilis Christophides. "On Provenance of Queries on Semantic Web Data." *IEEE Internet Computing*, **15**(1), 2011.

[TGN92] Douglas B. Terry, David Goldberg, David A. Nichols, and Brian M. Oki. "Continuous Queries over Append-Only Databases." In *SIGMOD Conference*, pp. 321–330, 1992.

[THF06] Quan Thanh Tho, Siu Cheung Hui, A. C. M. Fong, and Tru Hoang Cao. "Automatic Fuzzy Ontology Generation for Semantic Web." *IEEE Trans. on Knowl. and Data Eng.*, **18**(6):842–856, June 2006.

[THP08] Yuanyuan Tian, Richard A. Hankins, and Jignesh M. Patel. "Efficient aggregation for graph summarization." In *SIGMOD Conference*, pp. 567–580, 2008.

[Tur01] Peter D. Turney. "Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL." In *Proceedings of the 12th European Conference on Machine Learning*, EMCL '01, pp. 491–502, London, UK, UK, 2001. Springer-Verlag.

[Vit85a] Jeffrey Scott Vitter. "Random Sampling with a Reservoir." *ACM Trans. Math. Softw.*, **11**(1):37–57, 1985.

[Vit85b] Jeffrey Scott Vitter. "Random Sampling with a Reservoir." *ACM Trans. Math. Softw.*, **11**(1):37–57, 1985.

[Vou95] Atro Voutilainen. "NPtool, a detector of English noun phrases." *CoRR*, **cmp-lg/9502010**, 1995.

[VW99] Jeffrey Scott Vitter and Min Wang. "Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets." In *SIGMOD Conference*, pp. 193–204, 1999.

[WAA01] Yi-Leh Wu, Divyakant Agrawal, and Amr El Abbadi. "Using the Golden Rule of Sampling for Query Estimation." In *SIGMOD Conference*, pp. 449–460, 2001.

[web10a] "Business Databases at UCLA Anderson, School of Management, http://www.anderson.ucla.edu/x14506.xml.", December 2010.

[web10b] "UCI Machine Learning Repository, http://archive.ics.uci.edu/ml/datasets.html.", December 2010.

[WHW08] Fei Wu, Raphael Hoffmann, and Daniel S. Weld. "Information extraction from Wikipedia: moving down the long tail." In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pp. 731–739, New York, NY, USA, 2008. ACM.

[Wik] "Wikidata." http://www.wikidata.org.

[Wik12] "Wikipedia." http://www.wikipedia.org/, 2012.

[WLB12] Wilson Wong, Wei Liu, and Mohammed Bennamoun. "Ontology learning from text: A look back and into the future." *ACM Comput. Surv.*, **44**(4):20, 2012.

[WLH11] W.Wu, H. Li, H.Wang, and K. Zhu. "Towards a probabilistic taxonomy of many concepts." Technical report, 2011.

[WLW12] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Q. Zhu. "Probase: a probabilistic taxonomy for text understanding." SIGMOD '12, pp. 481–492, New York, NY, USA, 2012. ACM.

[Wor12] "WordNet." http://wordnet.princeton.edu/, 2012.

[WW08] Fei Wu and Daniel S. Weld. "Automatically refining the wikipedia infobox ontology." In *WWW*, pp. 635–644, 2008.

[WW10] Fei Wu and Daniel S. Weld. "Open Information Extraction Using Wikipedia." In *ACL*, pp. 118–127, 2010.

[YCB07] Alexander Yates, Michael Cafarella, Michele Banko, Oren Etzioni, Matthew Broadhead, and Stephen Soderland. "TextRunner: open information extraction on the web." In *Proceedings of Human Language Technologies*, pp. 25–26, 2007.

[Zip49] G. K. Zipf. *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, Reading, MA, 1949.

[ZLX06] Ying Zhang, Xuemin Lin, Jian Xu, Flip Korn, and Wei Wang. "Space-efficient Relative Error Order Sketch over Data Streams." In *ICDE*, p. 51, 2006.

[ZW07a] Qi Zhang and Wei Wang. "An efficient algorithm for approximate biased quantile computation in data streams." In *CIKM*, pp. 1023–1026, 2007.

[ZW07b] Qi Zhang and Wei Wang. "A Fast Algorithm for Approximate Quantiles in High Speed Data Streams." In *SSDBM*, p. 29, 2007.