UNIVERSITY OF CALIFORNIA
Santa Barbara

# Mining Disparate Sources for Question Answering

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Huan Sun

Committee in Charge:

Professor Xifeng Yan, Chair

Professor Linda Petzold

Professor Ambuj Singh

March 2016

The Dissertation of Huan Sun is approved:

Professor Linda Petzold

Professor Ambuj Singh

Professor Xifeng Yan, Committee Chair

January 2016

Mining Disparate Sources for Question Answering

by

Huan Sun

*To my parents and elder brother.*

# Acknowledgements

The five-year Ph.D. life has turned out to be invaluable for me to reflect and rediscover myself. I am sincerely grateful to every person and every experience for what they taught me along the journey. I feel incredibly fortunate to have Professor Xifeng Yan as my advisor. Professor Yan provides tremendous advice and support, promoting me from whom I was five years ago to whom I am now. He always encourages me to step out of the comfort zone, enabling me to achieve a higher and higher level in academics. He shares with me a lot of experience and lessons in not only research but also life, benefiting me so much that I would not have grown so quickly otherwise. His broad knowledge and vision, rigorous attitude towards research, and selflessness towards students make him an absolute role model that I will always look up to in my life afterwards.

I am also in great debt to my committee members: Professor Linda Petzold and Professor Ambuj Singh. They provided me invaluable feedback at each stage of my graduate study. Thank Professor Jiawei Han from UIUC for giving me great suggestions on my work, as well as on thesis writing. I would also like to give sincere thanks to all of them for their support and encouragements during my job search process.

I am truly blessed to have great mentors and collaborators. Thank Dr. Mudhakar Srivatsa from IBM for offering a lot of help with my internship project and

follow-up work. Thank Dr. Lance Kaplan from Army Research Lab (ARL) for valuable comments on the work in Chapter 4. Thank Dr. Ananthram Swami, Dr. Brian Sadler, Dr. Hasan Cam, Dr. Michelle Vanni, and Dr. Sue Kase for sharing their insights during my visit at ARL. I would like to thank my mentors and collaborators in Microsoft Research, Redmond. Special thanks to Dr. Hao Ma who offered me wonderful opportunities to visit MSR and introduced me to exciting research topics. Many thanks to Dr. Scott Yih, Dr. Xiaodong He, Dr. Ming-Wei Chang, Dr. Li Deng, and Dr. Yi-Min Wang for their precious advice on my two summer internship projects in Chapter 2 and 3.

I am always grateful to have studied and worked in the fantastic University of California, Santa Barbara. It has been such a great pleasure to work with my friends, former and current labmates: Shengqi Yang, Bo Zong, Yang Li, Shulong Tan, Yinghui Wu, Fangqiu Han, and Yu Su. The work in this dissertation is the result of close collaboration with them. Thank them for always being super supportive to me; for fruitful discussions and debates on various projects and topics; for taking care of me like a younger sister, especially during the first few years; for memories of many late nights and weekends to catch up with a deadline, and those of splendid trips to Las Vegas and Yosemite. I wish time could slow down a bit so that I could create more memories with these adorable guys on this adorable campus. I would also like to thank former members in the lab: Dr.

Gengxin Miao, Dr. Ziyu Guan, Dr. Nan Li, and Dr. Arijit Khan. They provided tremendous help for me to successfully adapt to the Ph.D. life. Thank them for being excellent role models. I also thank other friends in and outside our lab: Honglei Liu, Semih Yavuz, Izzeddin Gur, Xiaohan Zhao, Wen Chen, and Xin Jin. Thank them very much for sharing many happy moments and enriching my life at Santa Barbara. Special thanks go to my sister and closest friend, Xintong Yang. Thank her very much for witnessing my growth along the way, for sharing a lot of philosophies about life which I truly benefited from, and for all the encouragements she gave me whenever I felt frustrated or panic.

Last but not least, I would like to express my deepest gratitude to my parents and older brother for their unconditional love and support. Thank them for always encouraging me to be a fearless dream pursuer, and for being my strongest backing during this life-long process.

The research in this dissertation is funded in part by NSF IIS 0917228, IIS 0954125, ARMY W911NF-09-2-0053, and UCSB Regents Special Fellowship.

# Curriculum Vitæ

Huan Sun

### Education

| | |
|---|---|
| 2010-2015 | Ph.D. in Computer Science<br>University of California, Santa Barbara (UCSB) |
| 2006-2010 | B.S. in Electronic Engineering and Information Science<br>University of Science and Technology of China (USTC) |

### Experience

| | |
|---|---|
| 07/2016- | Assistant Professor, CSE Dept.@the Ohio State University |
| 01/2016-06/2016 | Visiting Scholar, the University of Washington, Seattle |
| 09/2015-12/2015 | Visiting Scholar, Baidu Research, Sunnyvale |
| 09/2010-09/2015 | Research Assistant, University of California, Santa Barbara |
| 06/2015-09/2015 | Research Intern, Microsoft Research, Redmond |
| 06/2014-09/2014 | Research Intern, Microsoft Research, Redmond |
| 06/2013-09/2013 | Research Intern, IBM T.J. Watson Research Center |
| 12/2009-03/2010 | Research Intern, Microsoft Research Asia |
| 09/2008-12/2009 | Undergraduate Research Assistant, USTC |

### Selected Publications

**H. Sun**, H. Ma, X. He, S. Yih, Y. Su, X. Yan. Table Cell Search for Question Answering. In **WWW** 2016.

Y. Li, S. Tan, **H. Sun**, J. Han, D. Roth, X. Yan. Entity Disambiguation with Linkless Knowledge Bases. In **WWW** 2016.

F. Han, S. Tan, **H. Sun**, X. Yan, M. Srivatsa, D. Cai. Distributed Representations of Expertise. In **SDM** 2016.

Y. Su, **H. Sun**, B. Sadler, M. Srivatsa, I. Gur, Z. Yan, X. Yan. A Configurable Natural Language Query Benchmark for Knowledge Bases.
Submitted to **SIGMOD** 2016.

**H. Sun**, H. Ma, S. Yih, C. Tsai, J. Liu, M. Chang. Open Domain Question Answering via Semantic Enrichment. In **WWW** 2015.

Y. Su, S. Yang, **H. Sun**, M. Srivatsa , S. Kase, M. Vanni, X. Yan. Exploiting Relevance Feedback in Knowledge Graph Search. In **SIGKDD** 2015.

Z. Guan, S. Yang, **H. Sun**, M. Srivatsa, X. Yan. Fine-Grained Knowledge Sharing in Collaborative Environments. In **TKDE** 2015.

**H. Sun**, M. Srivatsa, S. Tan, Y. Li, L. Kaplan, S. Tao, X. Yan. Analyzing Expert Behaviors in Collaborative Networks. In **SIGKDD** 2014.

S. Yang, Y. Wu, **H. Sun**, X. Yan. Schemaless and Structureless Graph Querying. In **VLDB** 2014.

S. Yang, Y. Xie, Y. Wu, T. Wu, **H. Sun**, J. Wu, X. Yan. SLQ: A User-friendly Graph Querying System. In **SIGMOD** 2014 (Demo).

N. Li, **H. Sun**, K. Chipman, J. George, X. Yan. A Probabilistic Approach to Uncovering Attributed Graph Anomalies. In **SDM** 2014.

S. Tan, Y. Li, **H. Sun**, Z. Guan, X. Yan, J. Bu, C. Chen, X. He. Interpreting the Public Sentiment Variations on Twitter. In **TKDE** 2014.

**H. Sun**, A. Morales, X. Yan. Synthetic Review Spamming and Defense.
In **SIGKDD** 2013.

**H. Sun**, G. Miao, X. Yan. Noise-Resistant Bicluster Recognition.
In **ICDM** 2013.

### Awards and Honors

Outstanding Dissertation Award in CS Dept., UCSB, 2015.
Regents Special Fellowship, UCSB, 2010-2011, 2014-2015.
Ph.D. Progress Award in CS Dept., UCSB, 2014.
NSF Student Travel Award, SIGKDD 2014.
NSF Travel Award, RECOMB 2013.
Doctor Forum Travel Award, SDM 2012.
ICML registration waiver, 2011.
Excellent Graduation Thesis Award, USTC, 2010
Guanghua Education Scholarship, USTC, 2009
Excellent Undergraduate Research Project Scholarship, USTC, 2009
National Scholarship, USTC, 2008
National Scholarship, USTC, 2007

## Abstract

Mining Disparate Sources for Question Answering

by

Huan Sun

Today's paradigm of information search is in the midst of a significant transformation. Question answering (QA) techniques that can directly and precisely answer user questions are becoming more and more desired, in contrast to traditional search engines retrieving lengthy web pages. The big data age is endowed with large-scale diversified information sources, such as structured knowledge bases (KBs), unstructured texts, semi-structured tables, as well as human networks including social and expert networks. *How to mine such large-scale and disparate sources to advance question answering?* In this dissertation, we systematically investigate this problem from the perspectives of text mining, network analysis and human behavior understanding. Specifically, our research lies in:

(1) Text-based question answering. We recognize that KBs are usually far from complete and information required to answer questions may not always exist in KBs. This framework jointly utilizes web texts and knowledge bases: It

mines answers directly from large-scale web corpora, and meanwhile employs KBs as a significant auxiliary to boost QA performance.

(2) Table-based question answering. Owing to their prevalence on the Web and large topical diversity, we explore tables, which are distinctive from KBs and texts, for question answering. Specifically, we investigate the problem of given millions of tables, how to precisely retrieve *table cells* to answer a user question. We propose a table cell search framework to deal with it. The framework is compared with state-of-the-art KB-based QA systems. Experimental results validate the hypothesis that web tables provide rich knowledge missing from existing KBs, and thus serve as a good complement.

(3) Expert-based question answering. The intelligence possessed by current machines is still limited in many aspects. Human intelligence, contributed by crowdsourcing platforms and collaborative networks, should be exploited to complement machine-aided question answering and problem solving. We are among the first to quantitatively analyze task/question routing behaviors of experts in real collaborative networks, which aims at detecting the efficiency bottleneck and optimizing human collaboration.

The developed methodologies and frameworks in this dissertation hence pave the path for building intelligent systems which can utilize an array of comple-

mentary knowledge sources including knowledge bases, texts, tables, and human networks to directly answer user questions in various domains, discover novel knowledge, and thereby assist problem solving and decision making.

Professor Xifeng Yan
Dissertation Committee Chair

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

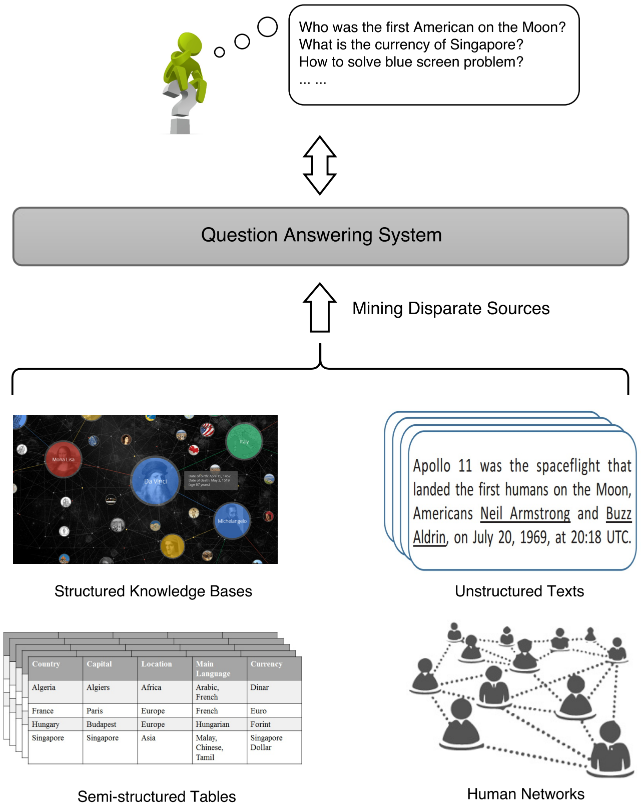
Question answering (QA) aims at directly returning exact answers to natural language questions. With the popularity of mobile devices, directly and precisely answering natural language questions greatly enhances user experience [22], in contrast to traditional information search returning lengthy documents. Apart from information retrieval, QA is also being actively investigated in various other disciplines. Database researchers have been building natural language interfaces to databases [4, 37, 39, 53], which shall shield users from understanding complex database schemas and constructing formal queries like SQL or SPARQL. Question answering based on images and videos [54, 73, 77] is becoming a more and more popular task in computer vision to demonstrate the intelligence possessed by a computer program. In the near future, not only humans use search engines, but also robots will. Search engines for robots such as RoboBrain [56] are under active construction. Given a natural language command "*Bring me sweet tea from the* *kitchen*", an autonomous robot can interact with RoboBrain to obtain knowledge necessary to execute the command, such as "*Sweet tea can be kept on a table or inside a refrigerator*" and "*Bottle can be grasped in certain ways*". QA research empowers humans to efficiently acquire knowledge and to build more and more intelligent systems for understanding big data and executing assigned tasks.

Not only connected with many research disciplines, question answering techniques are also entitled with various real-life applications in the big data age. In healthcare, given a large number of electronic health records, frontier research reports, and medical forum posts, it is unaffordable for humans to digest all such data to acquire certain knowledge. People can simply ask an intelligent QA system, e.g., "*What diseases can be induced by fever in babies*", "*How to treat a flu when pregnant*", "*What is a good meal plan for people with Anemia and wheat allergy*" to obtain precise information and formulate best and personalized treatment options. Such intelligent QA in healthcare is being pursued in many companies and clinics such as Baidu, IBM, and Mayo Clinic. Customer service is a critical part in almost every business. A service provider might need to handle, on a daily basis, thousands of queries that report various types of product problems from its customers. How to organize an expert team to efficiently resolve such queries determines, to a large extent, its competitive advantage. Decision-support systems in business intelligence directly answer human questions after analyzing massive enterprise data, press coverage, and social sentiment in blogs and customer reviews etc. A handful of companies including IBM and HG Data [2] are building cost-effective and scalable systems to achieve such goals. Such systems can greatly benefit various business dimensions including new customer discovery, customer behavior analysis, and monitoring potential competitors.

The big data age is known for the flood of diversified large-scale information sources, such as structured knowledge bases, unstructured texts like news articles and forum posts, semi-structured tables on the Web and in the enterprise domain, as well as human networks including social and expert networks. *How to utilize such big, disparate yet connected data resources to advance QA?* In this dissertation, as shown in Figure 1.1, we systematically investigate this problem from the perspectives of text mining, network analysis and human behavior understanding, based on two key observations:

- Although QA based on knowledge bases is popularly studied in recent years, existing knowledge bases are far from complete. Other information sources including comprehensive web texts and HTML tables should be exploited together with knowledge bases, in a complementary manner with each other.

- What current automated algorithms can achieve is still quite limited in many aspects, which necessitates the exploitation of human intelligence for knowledge discovery and problem solving.



Who was the first American on the Moon?
What is the currency of Singapore?
How to solve blue screen problem?
... ...

Apollo 11 was the spaceflight that landed the first humans on the Moon, Americans Neil Armstrong and Buzz Aldrin, on July 20, 1969, at 20:18 UTC.

| Country | Capital | Location | Main Language | Currency |
|---|---|---|---|---|
| Algeria | Algiers | Africa | Arabic, French | Dinar |
| France | Paris | Europe | French | Euro |
| Hungary | Budapest | Europe | Hungarian | Forint |
| Singapore | Singapore | Asia | Malay, Chinese, Tamil | Singapore Dollar |

Figure 1.1: Mining disparate sources for QA.

The thesis statement of this dissertation is as follows: (1) Question answering is well connected to many disciplines and endowed with various real-life applications in the big data age. (2) Both intelligent computing machines and large-scale human networks shall be resorted to answer questions and solve problems: On the one hand, human-aided question answering tends to induce a high cost in both time and money. Automated question answering is efficient and thereby highly desired. On the other hand, given the limited intelligence of current machines in understanding languages, human collective intelligence should be exploited to complement machine-aided problem solving. (3) In the big data era with large-scale knowledge bases, web texts, web tables, and human networks, methodologies in text mining, network science, and human behavior understanding can significantly advance both automated and human collaborative question answering.

Driven by this statement, our work has centered around mining disparate sources for question answering: (1) Text-based question answering [62]; (2) Table-based question answering [61]; (3) Expert-based question answering [63].

## 1.1 Text-based Question Answering

Various intelligent knowledge discovery and retrieval systems (e.g., Google Now, Apple Siri, Microsoft Cortana, and Amazon Echo) largely benefit from large-scale knowledge bases, or knowledge graphs, such as Freebase, Microsoft Satori, Google Knowledge Graph. Not only in industry, but also in academia numerous QA systems [6,7,24,25] are developed based on knowledge bases. However, despite their large size, existing knowledge bases are still far from complete and not updated in a timely fashion [21,45,68]. As a result, information required to answer a question may not always exist in KBs. In contrast, interesting or important facts and statements can often appear repeatedly in copious web texts including news articles, Wikipedia-like pages, community QA sites, blogs and forums.

Prior to the blossom of KBs, text-based QA systems were popularly studied and generally viewed as a variation of information retrieval systems. This view can be exemplified by the TREC QA tracks [67], where each participant system is required to extract a small piece of text from a large collection of documents, as the answer to a natural language query. Systems like Mulder [35] and AskMSR [11] leverages the *crowd knowledge* from the Web and avoids deep natural language analysis: such systems issue simple reformulations of the given questions as queries to a search engine, and rank the repeatedly occurring $N$-grams in the top snippets as answers, based on named entity recognition (NER) [43] and heuristic answer type checking. Despite the simplicity of this strategy, such systems are highly scalable and are among the top performing systems in TREC-10 [12]. One main weakness of these text-based QA systems is its insufficient knowledge about the generated answer candidates. For instance, different mentions of the same entity such as “*President Obama*” and “*Barack Obama*” are viewed as different answer candidates, and will not be grouped together in most cases. Answer type checking, which verifies whether the type of an answer candidate matches the question, relies on a generic named entity recognition component that provides a small set of crude type labels. As a result, such systems are typically limited to answering questions in only a handful of categories.

Given such situations about KB- and traditional text-based QA systems, we consider : *Can we jointly utilize texts and knowledge bases for question answering*? In Chapter 2, our QuASE (i.e., question answering via semantic enrichment) system mines answer candidates from large-scale web texts, and meanwhile employs KBs as a significant auxiliary to determine the true answer. Specifically, to the best of our knowledge, we make the first attempt to link answer candidates to entities in Freebase, during answer candidate generation. Several remarkable advantages follow: (1) Redundancy among answer candidates is automatically reduced; (2) The types of an answer candidate can be effortlessly determined by those of its corresponding entity in Freebase; (3) Capitalizing on the rich information about entities in Freebase, such as entity description texts and entity types, we can naturally develop semantic features for each answer candidate after linking them to Freebase. Particularly, we propose two novel probabilistic models to construct answer-type related features, which directly evaluate the appropriateness of an answer candidate's types under a given question. Overall, such semantic features turn out to play significant roles in determining the true answers from the large answer candidate pool. The experimental results show that across two testing datasets, our QA system achieves an 18% $\sim$ 54% improvement under $F_1$ metric, compared with various existing QA systems.

## 1.2 Table-based Question Answering

Apart from texts, we observe that informative tabular data are also pervasive on the Web: According to [40], based on a conservative estimation, over 25 million tables in 500 million web pages are expressing relational information, as opposed to implementing visual layout. Such tables naturally serve as valuable answer sources to satisfy user information needs. For example, Table 1.1 shows a list of countries and their attributes, which can answer question "*What languages do people in France speak*". Driven by this observation, in this dissertation, we investigate an important yet largely under-addressed problem: *Given millions of tables, how to precisely mine table cells to answer a user question?*

Question answering based on tables has been studied but in different formulations. In Pasupat et al. [51], the table that contains answers to an input question is known beforehand, and their task is to find answers in the given table. In our concerned problem, however, we need to explore a huge set of tables to answer a question. Using natural language interfaces to databases (NLIDBs) [4, 37, 39, 53], users can pose natural language queries instead of complex SQL queries to access databases. An NLIDB translates a natural language query into an SQL query based on the rigid schema of a given relational database. It is therefore hard to be applied on the unconstrained schemas of web tables in our task, where each table has a self-defined schema. Finally, while the authors in [15, 20] directly search

| **Country** | **Capital** | **Currency** | **Main Language** |
|---|---|---|---|
| Algeria | Algiers | Dinar | Arabic |
| Egypt | Cairo | Pound | Arabic |
| France | Paris | Euro | French |
| ... | ... | ... | ... |

Table 1.1: An example table on the Web.

relevant tables to satisfy user queries, we move one step further to precisely find table cells that contain correct answers (i.e., answer cells). There are two main challenges in identifying the correct answer cells: (1) *Among millions of tables, how to detect the relevant ones that may contain answers*? (2) *How to precisely locate answer cells in a relevant table*? This dissertation proposes a novel table cell search framework to tackle these challenges in Chapter 3.

We first formulate the concept of a relational chain which connects two cells in a table and represents the semantic relation between them. With the help of search engine snippets, our framework generates a set of relational chains pointing to potentially correct answer cells. We further employ deep neural networks to conduct more fine-grained inference on which relational chains best match the input question and finally extract the corresponding answer cells. Based on millions of tables crawled from the Web, we evaluate our framework in the open-domain question answering setting, using both the well-known WebQuestions dataset and user queries mined from Bing search engine logs. On WebQuestions, our framework is comparable to state-of-the-art QA systems based on knowledge bases, while on

Bing queries, it outperforms other systems by at least 56.7%. Moreover, when combined with results from our framework, KB-based QA performance can be significantly boosted by 28.1% to 66.7%, demonstrating that web tables supply rich knowledge that might not exist or is difficult to be identified in existing KBs.

## 1.3 Expert-based Question Answering

Despite the recent boom of automated QA systems, the intelligence possessed by current machines is still limited in many aspects such as understanding complicated questions and organizing different information pieces to form an answer. Human intelligence can be exploited to complement machine-aided problem solving. In fact, the surge of crowdsourcing platforms (e.g., Amazon MTurk, Crowdflower) and collaborative networks (e.g., Github) symbolizes the popularity of human-aided knowledge discovery and problem solving systems.

Collaborative networks are composed of experts who cooperate with each other to complete specific tasks, such as answering questions and resolving problems from customers (e.g., "*Reset password for user X*" and "*How to solve blue screen problem*"). We use "*task*", "*question*", and "*problem*" interchangeably in this context. A task is posted and subsequently routed in the network from an expert to another until being resolved. When an expert cannot solve a task, his routing decision (i.e., where to transfer a task) is critical since it can significantly affect the completion time of a task.

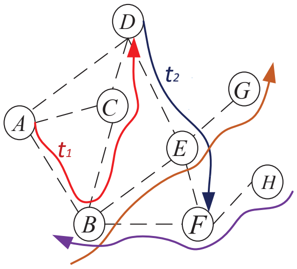


Figure 1.2: A sample collaborative network.

Figure 1.2 shows a sample collaborative network with task routing examples. Task $t_1$ starts at expert $A$ and is resolved by expert $D$ while task $t_2$ starts at expert $D$ and is resolved by expert $F$. The sequences $A \rightarrow B \rightarrow C \rightarrow D$ and $D \rightarrow E \rightarrow F$ are called *routing sequences* of task $t_1$ and $t_2$ respectively. The number of experts on a routing sequence measures the completion time of a task. The average completion time of tasks signifies the efficiency of a collaborative network in problem solving: the shorter, the more efficient. When the number of experts in a collaborative network becomes large, to whom an expert routes a task significantly affects the completion time of the task. For example, in Figure 1.2, task $t_1$ can be directly routed to resolver $D$ from $A$. In this case, the routing decision made by expert $A$ is critical. Therefore, understanding how an expert makes a certain routing decision and mining his routing behavioral patterns shall ultimately help improve the efficiency of a collaborative network.

Task resolution in collaborative networks has been studied before. Shao et al. [58] propose a sequence mining algorithm to automatically and efficiently route tasks to resolvers. Miao et al. [44] develop generative models and recommend better routing by considering both task routing sequence and task content. In [76], Zhang et al. study the resolution of prediction tasks, which is to obtain probability assessments for a question of interest. All of these studies aim at developing automated algorithms that can effectively speed up a task's resolution process. However, they largely ignore human factors in real task routing. Take Figure 1.2 as an example. Why does expert $A$ route task $t_1$ to $B$ instead of $D$? Is it because he does not understand $t_1$ well, thus randomly distributing it to $B$, or he believes $B$ has a better chance to solve it, or $B$ has a better chance to find the right expert to solve it? Does expert $A$ make more rational decisions than random decisions? While it is very hard to infer $A$'s decision logic based on an individual task, it is possible to infer it by analyzing many tasks transferred and solved by $A$, $B$ and $D$. In Chapter 4, we focus on analyzing *real* experts' decision logic, i.e., what kind of routing patterns an expert follows when deciding where to route a task. This understanding will help detect the inefficiency spots in a collaborative network and give guidance to management teams to provide targeted expert training.

## 1.4 Contributions

This dissertation comprehensively investigates the potential of disparate sources for QA, including unstructured texts linked with structured knowledge bases, semi-structured tables, and human networks. The concepts and techniques developed in this dissertation are to deal with key problems in both automated and human-aided QA, such as answer type matching with questions, representation of table cells, how to match table cells with a natural language question, and real expert behavior understanding for collaboration optimization. Our developed mining frameworks and analytical models aim at unleashing the power of big and diversified data to advance QA, which shall benefit various domains in the future including but not limited to healthcare, customer service, business intelligence, and software engineering. Our key contributions are summarized as follows:

### Exploiting multiple disparate yet connected sources.

Given the limitations of separately using them as answer sources, knowledge bases and texts are jointly exploited via the links in between: our QA framework mines answer candidates from comprehensive web corpora, links them to knowledge bases, and employs rich semantic information in knowledge bases to help determine the true answer. Apart from knowledge bases (structured) and texts (unstructured), we also explore the information covered by abundant semi-structured HTML tables on the Web for QA. Unlike unstructured texts, tables come with schemas, making it much easier to interpret each column and relations between columns. Moreover, relation representation in tables is quite different from, often more straightforward than, that in knowledge bases. These distinctions from knowledge bases and texts call for new mining frameworks in order to detect answers from tables. Last but not least, human networks are also a critical source to answer questions and solve problems, especially in complicated situations where automated QA fails. We focus on understanding human networks via mining individual behavioral patterns, in order to optimize collaboration and boost problem solving efficiency. Our research paves the way for an array of interesting topics, including how to unify/combine disparate data sources for QA, successful marriage of automated and human-aided QA, and harnessing machine intelligence and human intelligence in a mutually-boosting manner.

## Identifying critical challenges/limitations.

Chapter 2 identifies the incompleteness limitation of knowledge bases to QA, and aims at mining answers via jointly using web texts and KBs. After linking answer candidates detected from web texts to knowledge bases, how to utilize their rich information in KBs, such as fine-grained entity types, is anything but a trivial task. In Chapter 3, we recognize pervasive web tables as a valuable information source for QA. When mining tables, however, two main challenges need to be attacked: (1) Among millions of tables, how to detect the relevant ones that may contain answers? (2) How to precisely locate answer cells in a relevant table? For human-aided QA in Chapter 4, we are among the first to quantitatively analyze the task routing behaviors of *real* experts, i.e., how an expert decides where to transfer a task when she could not solve it. Such analysis shall spot the efficiency bottleneck and help further optimize human collaborations.

### Novel and effective methodologies, models and concepts.

In Chapter 2, we utilize web texts to detect answer candidates, and generate features based on the rich information in KBs to help determine the true answer. Particularly, we develop novel probabilistic models to directly evaluate the appropriateness of an answer candidate's types (available in KBs) given a question. The experimental results show that across two testing datasets, our framework jointly using KBs and web texts achieves an 18% $\sim$ 54% improvement under $F_1$ metric, compared with QA systems based on either KBs or web texts. Chapter 3 proposes an effective framework to find table cells that can answer a question. The core concept underlying the proposed framework is the relational chain representation of table cells. We employ deep neural networks to match a natural language question with a relational chain to detect correct answer cells thereafter.

Our table cell search framework is either comparable to or significantly outperforms state-of-the-art KB-based QA systems by at least 56.7%. Moreover, when combined with results from our framework, KB-based QA performance can be significantly boosted by 28.1% $\sim$ 66.7%. Chapter 4 presents a generative modeling approach to investigate the decision making and cognitive process of real experts, where a routing decision is a mixture of routing patterns potentially followed by an expert. We experimentally show that our analytical model can be employed to both accurately predict the completion time of a task before starting routing it in the network, and optimize collaborative networks via hypothesis testing.

# Chapter 2

# Text-based QA

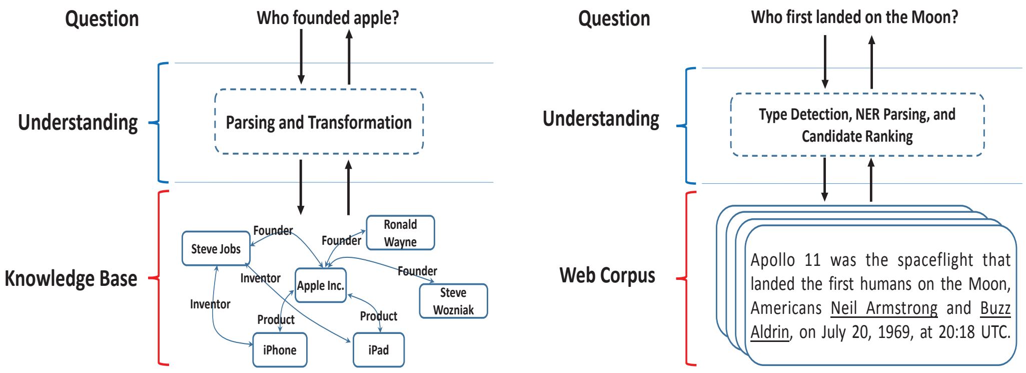
In this chapter, we introduce question answering based on either knowledge bases or texts, and discuss in great details our QA framework jointly exploiting knowledge bases and web texts. Since our framework mines answer candidates from web texts, we still refer it to text-based QA. However, the fundamental difference between our methodology and traditional text-based QA is that rich information in KBs is utilized to determine the correctness of answer candidates.

## 2.1 Preliminaries

Knowledge bases store a wealth of relation tuples (e.g., <*Obama*, *Place-Of-Birth*, *Honolulu*>), which provide answers to questions such as "*Where was Obama born*". Figure 2.1 (left) briefly illustrates the scheme of a KB-based QA system, where a question gets answered by being parsed and transformed to a specific form such as logic form, graph query and SPARQL, suitable to execute on KBs.

For example, Berant et al. [6, 7] develop semantic parsing techniques that map natural language utterances into logical forms. The Paralex system [24] extracts relation tuples from general web corpora via information extraction tools (e.g., ReVerb [23]) and stores them as extracted KBs; during QA, it maps open-domain questions to queries over the extracted KBs. QA systems developed in [25] resort to both curated KBs such as Freebase and extracted KBs from general corpora, to answer a question. Zou et al. [78] propose to represent a natural language question using a semantic graph query to be matched with a subgraph in KBs and reduce question answering to a subgraph matching problem. Unger et al. [66] rely on parsing a question to produce a SPARQL template, which mirrors the internal structure of the question. This template is then instantiated using statistical entity identification and predicate detection. Similarly, Yahya et al. [70] present a methodology for translating natural language questions into structured SPARQL queries based on an integer linear program.

To counter the incompleteness issue of existing knowledge bases [21, 45, 68], consider the fact that interesting or important stories and statements can often appear repeatedly in rich web corpora including news articles, Wikipedia-like pages, community QA sites, blogs and forums. Prior to the availability and popularity of knowledge bases, most of early QA systems such as [11, 17, 26, 34, 57, 67], mine answers from TREC [67] document collections or rich web corpora. In [34],



Apollo 11 was the spaceflight that landed the first humans on the Moon, Americans Neil Armstrong and Buzz Aldrin, on July 20, 1969, at 20:18 UTC.

Figure 2.1: KB-based (left) and traditional text-based (right) QA systems.

Ko et al. focus on a general model to estimate the correctness of answer candidates, instead of developing an entire QA system. Ferrucci et al. [26] give an overview of IBM Watson system framework including question analysis, search, hypothesis generation, and hypothesis scoring. Without deep natural language analysis, Mulder [35] and AskMSR [11] issue simple reformulations of questions to a search engine, and rank the repeatedly occurring $N$-grams in the top snippets as answers, based on named entity recognition [43] and heuristic answer type checking. A high-level view of such systems is illustrated in Figure 2.1 (right).

As discussed in Section 1.1, to address the disadvantages of KB-based and traditional text-based QA systems, we propose a new framework, named **QuASE**, (i.e., question answering via semantic enrichment), which jointly utilizes knowledge bases and web texts for QA. Our system extends traditional text-based QA system by linking answer candidates in the search texts to a knowledge base.

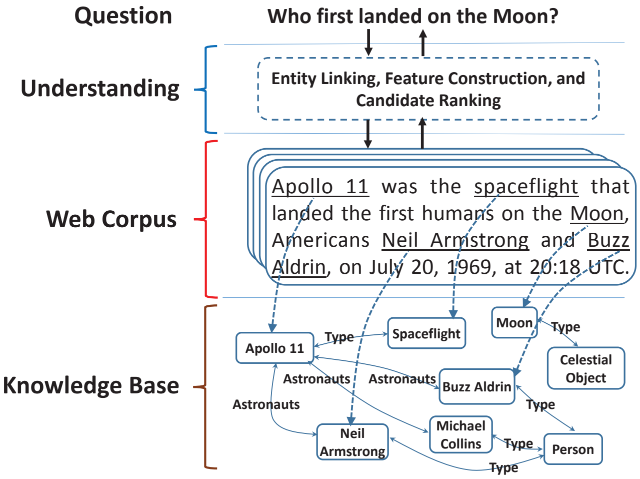

Question
Who first landed on the Moon?
Understanding
Entity Linking, Feature Construction, and Candidate Ranking
Web Corpus
Apollo 11 was the spaceflight that landed the first humans on the Moon, Americans Neil Armstrong and Buzz Aldrin, on July 20, 1969, at 20:18 UTC.
Knowledge Base
Apollo 11
Type
Spaceflight
Moon
Type
Celestial Object
Astronauts
Astronauts
Buzz Aldrin
Type
Astronauts
Neil Armstrong
Michael Collins
Type
Person
Type

Figure 2.2: Process diagram of our framework.

Figure 2.2 briefly illustrates how our system works, in contrast to systems in Figure 2.1. Specifically, given a question, QuASE first selects a set of most prominent sentences from web texts. Then from those sentences, we utilize entity linking tools [19] to detect answer candidates and link them to entities in Freebase. Once each answer candidate is mapped to the corresponding entity in Freebase, redundancy among answer candidates is automatically reduced. Additionally, abundant information, such as entity description texts and Freebase types, can be utilized for feature generation and modeling. A ranking algorithm is subsequently trained based on such features to rank correct answers as top choices. Section 1.1 summarizes the remarkable advantages brought by harnessing both KBs and web texts for QA in our framework.

## 2.2 Text Mining

Figure 2.3 shows an end-to-end pipeline of our QA framework, which contains the following components in order: (1) Web Sentence Selection via Search Engine; (2) Answer Candidate Generation via Entity Linking; (3) Feature Generation and Ranking. We elaborate the details of each component as follows:

(1) **Web Sentence Selection via Search Engine**. Given a question, in order to find high-quality answer candidates, we design the following mechanism to retrieve highly relevant sentences from the Web that can potentially answer the question. We first issue the question in a commercial search engine, and collect the top-50 returned snippets, as well as the top-50 documents. Since a query itself is generally short and contains only a few words, we compute the word count vector based on the returned snippets to represent the information for the query, denoted as $w_q$. For each sentence we parsed from the top-50 returned documents, we compute its word count vector $w_s$, and select those sentences with a high $\cos(w_s, w_q)$ into the high-quality sentence set. If $w_s$ deviates far from $w_q$, the corresponding sentence is regarded as irrelevant and thereby discarded.

(2) **Answer Candidate Generation via Entity Linking**. Once we obtain the sentence set, one of the state-of-the-art entity linking systems [19] is applied to identify answer candidates linked to Freebase. This system achieves the best scores at TAC-KBP 2013, by several novel designs such as postponing surface

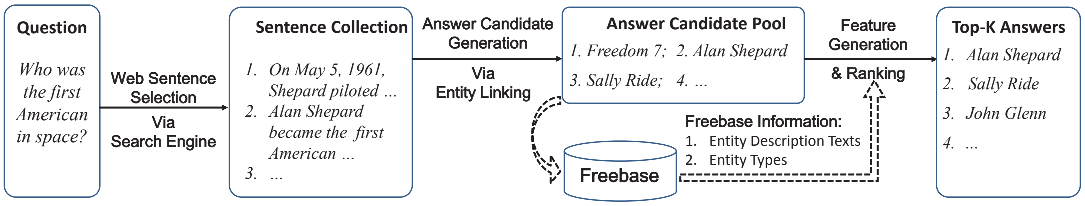


Figure 2.3: System framework of QuASE.

form boundary detections and discriminating concepts and entities in Wikipedia pages. Since the major target here is to verify that incorporating rich information from KBs will greatly boost the QA performance, we do not focus on constructing new entity linking tools in this chapter. Moreover, for questions whose answers are not entities in Freebase, such as questions starting with "*when*", our system can be reduced to traditional web texts based QA systems without the auxiliary of KBs. In this chapter, without loss of generality, we primarily focus on those questions targeted at certain entities in KBs.

(3) **Feature Generation and Ranking**. For each answer candidate, Freebase contains a wealth of information, such as their description texts and entity types. A set of semantic features shall be developed based on such rich information, and subsequently utilized in a ranking algorithm to evaluate the appropriateness of each candidate as the true answer.

Now we use an example to show how our system works. Given a question "*Who was the first American in space*", we submit it to a search engine to return a set of relevant sentences {1. *On May 5, 1961, Shepard piloted the Freedom 7 mission...* ; 2. *Alan Shepard became the first American in space when the Freedom 7...*; ... }. On this sentence set, we apply entity linking to extract entities , such as "*Freedom 7*", "*Alan Shepard*", and "*Sally Ride*", and link them to Freebase. Such linked entities are treated as answer candidates to the given question. For each answer candidate, semantic features are developed based on their rich information in Freebase, and subsequently integrated into a ranking algorithm, so that the true answer "*Alan Shepard*" will be ranked at the top of the candidate list.

Our QA system distinguishes itself from existing ones, in that it not only mines answers directly from large-scale web texts, but also employs Freebase as a significant auxiliary to boost QA. Freebase plays a significant role in both answer candidate generation and feature generation. By linking answer candidates to Freebase, our system is entitled with several unique advantages, such as reducing the redundancy among answer candidates and effortlessly granting answer candidates with Freebase entity types. Moreover, two kinds of rich information in KBs, entity description texts and entity types, will be naturally utilized to develop semantic features for downstream answer candidate ranking.

## 2.3 Feature Development

Upon the generation of an answer candidate pool, effective features shall be developed in order to rank true answers as top choices. In this section, we elaborate the features developed for answer candidate ranking. Given a question, totally three categories of features are computed for each answer candidate. The features include both (1) non-semantic features: frequency that an answer candidate occurs in the retrieved sentence set, and (2) semantic features: Since we have linked each answer candidate to Freebase via entity linking, we are able to utilize their rich information in KBs to develop semantic features.

### 2.3.1 Count

The sentence set, returned by the sentence selection component, is considered quite related to the given question. The more frequent an answer candidate occurs in the sentence set, the more related it is to the question. Therefore, the *count* or *frequency* of each answer candidate serves as a significant indicator of being correct or not. We compute the count of each answer candidate as one feature.

### 2.3.2 Textual Relevance

Given a question, the context where the true answer occurs and its descriptions in KBs should match the question. To evaluate the relevance of an answer candidate to a question, we first extract textual information from both the question side and the answer candidate side.

### Textual Information on Question Side

(1) The set of words in question $q$;

(2) The relevant sentence set returned for question $q$.

### Textual Information on Answer Candidate Side

(3) The context windows where an answer candidate $a$ appears in retrieved sentences. Due to the general short length of a sentence, the size of a context window is set at 2, i.e., 2 words before and after answer candidate $a$ compose the context where $a$ occurs in a sentence. We collect all the context windows for each answer candidate in all the sentences.

(4) The description texts of answer candidate $a$ in KBs. For example, Freebase describes entity "*Alan Shepard*", as "*Alan Bartlett 'Al' Shepard, Jr., was an American naval officer and aviator, test pilot, flag officer, one of the original NASA Mercury Seven astronauts ...*".

Based on the textual information for both questions and answer candidates, there can be many methods to measure the matching degree between a question and an answer candidate. We utilize a most intuitive method: For each piece

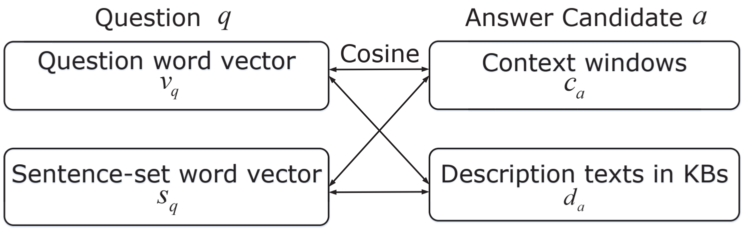

Question $q$
Answer Candidate $a$
Question word vector $v_q$
Cosine
Context windows $c_a$
Sentence-set word vector $s_q$
Description texts in KBs $d_a$

Figure 2.4: QA textual relevance features.

of information (1) to (4), we compute the word frequency vector denoted as $v_q$, $s_q$, $c_a$, and $d_a$ respectively. Then we apply cosine similarity measure between the textual information on the question side and that on the answer candidate side. As shown in Figure 2.4, we totally compute 4 features based on textual information for each <question, answer candidate> pair. Despite their simplicity, the features turn out to be very effective in improving QA performance. In fact, as future work, more sophisticated features can be developed and incorporated into our framework, such as deep semantic features learnt via deep learning [30].

### 2.3.3 Answer Type

Given a question "*the first American in space*", in the sentence set we selected, both the entity "*Alan Shepard*" (the astronaut) and the entity "*Freedom 7*" (the spaceflight) occur frequently. Both of them would be textually related to the question measured by features in Section 2.3.2. Therefore, the above count and textual relevance features turn out to be insufficient for such questions. However, by further checking the expected answer type of the question, it is obvious that the question is looking for a person, not a spaceflight. Therefore, to find correct answers, we need to build answer-type related features, which evaluate the appropriateness of an answer candidate's types under a question.

There has been much research studying the expected answer types of a question [5, 36, 38, 48, 49, 52]. Directly applying their methodology to our setting is not trivial. First, previous type-prediction methods adopt a supervised learning methodology where they classify questions into a small set of types such as person and location. Such methods can hardly scale to thousands of entity types in Freebase, especially when the expected answer of a question is associated with multiple types in Freebase, e.g., entity "*Barack Obama*" associated with multiple types such as "*government.president*", "*people.person*", and "*celebrities.celebrity*". On the other hand, it is quite challenging, if not impossible, to build a mapping between the small set of types and an answer candidate's Freebase types, since Freebase contains thousands of fine-grained types while the types studied in previous methods are quite general and limited. In this dissertation, we directly handle thousands of Freebase types, and propose probabilistic models to directly measure the matching degree between a question and an answer candidate's Freebase types. The intuition behind such models is that words in a question should correlate with its answer types. Given a question $q$, we try to model the probability

$P(t_a|q)$ or $P(q, t_a)$, where $t_a$ is the set of Freebase types associated with answer candidate $a$. Answer candidate $a$ with correct types should correspond to a higher $P(t_a|q)$ or $P(q, t_a)$. We next discuss two perspectives to model $P(t_a|q)$ and $P(q, t_a)$ respectively, and build our answer-type related features based on them.

## 2.4 Word to Answer Type

Now we first model the type predictive power of a single word, and then explore different models to integrate the predictive power of each word in a question. To emphasize the separate consideration of each word in the first step, the underlying methods are named *word to answer type* (WAT) models.

In WAT, we define $P(t|w)$ as the conditional probability of observing Freebase type $t$ as one answer type, if word $w$ occurs in the question. To learn $P(t|w)$, we rely on a training set of questions, each of which is paired with its expected answer types. We will give more details on such training sets in our experiments. Given such a training set, we model $P(t|w)$ as follows:

$$P(t|w) = \frac{\#(w,t)}{\sum_t \#(w,t)}, \tag{2.1}$$

where $\#(w, t)$ represents the co-occurrence frequency of word $w$ and type $t$ in the <question, answer types> pairs. The more frequently a type $t$ co-occurs with a word $w$, the more likely the answer contains type $t$, if word $w$ is in the question.

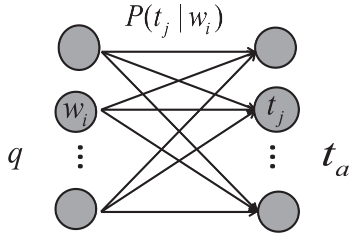


Figure 2.5: Word to Answer Type (WAT) model.

Given a question $q$ with a set of words $\{w_i\}$ and answer candidate $a$ with a set of Freebase types $t_a = \{t_j\}$, Figure 2.5 shows the point-wise word-to-type conditional probability, between the word set in $q$ and the type set of $a$. Based on this point-wise conditional probability, the probability $P(t_a|q)$, can be estimated using the following different models.

1. Best Word-to-Type: Choose the highest point-wise conditional probability.

$$P(t_a|q) = \max_{w_i \in q, t_j \in t_a} P(t_j|w_i). \tag{2.2}$$

2. Pivot Word: Choose the word in $q$ that generates the highest productive conditional probability for all the types in $t_a$.

$$P(t_a|q) = \max_{w_i \in q} \prod_{t_j \in t_a} P(t_j|w_i). \tag{2.3}$$

3. Pivot Word-to-Type: Choose the best set of point-wise conditional probabilities that give the highest productive conditional probability for all the types in $t_a$.

$$P(t_a|q) = \prod_{t_j \in t_a} \max_{w_i \in q} P(t_j|w_i). \tag{2.4}$$

Now we define the WAT feature for answer candidate $a$ based on the perplexity [10] of its type set $t_a$ as:

$$\text{Perplexity}(t_a) = \exp(-\frac{\log(P(t_a|q))}{|t_a|}), \tag{2.5}$$

where $|t_a|$ is the number of types in $t_a$. For Best Word-To-Type, since only one type is considered in the calculation, $|t_a| = 1$.

Perplexity has been applied for the evaluation of different topic models such as LDA [10], whereas in this chapter we use it as a matching measure between an answer candidate's Freebase types and words in a question. WAT model works based on the assumption that words in a question are predictive of the expected answer types. Based on golden pairs of questions and their expected answer types, we extract the distribution pattern of different types given a specific word in a question. For a new question and one answer candidate, if the answer candidate's types are expected, they should be better explained under the WAT models, i.e., associated with a higher $P(t_a|q)$, than otherwise. Correspondingly, WAT features for answer candidates with expected types should be lower than those with unmatched types. Overall three WAT features can be extracted, with $P(t_a|q)$ respectively instantiated by one of the three proposed models.

## 2.5 Type Association Modeling

Now we consider the question "*How likely can we observe a question and an entity with certain Freebase types, as a question-answer pair*". Different from WAT, we consider the predictive power of all the words simultaneously in a question. Given a question-answer pair, where answer $a$ is associated with multiple Freebase types $t_a$, we build a generative model of the joint likelihood $P(q, t_a)$, to measure the matching of $t_a$ with the question.

We assume the observation of a question $q$ and its associated answer types $t_a$, can be explained by latent association patterns. One of such latent associations might be, words such as "*city*", "*place*", and "*where*" will occur frequently in a question, if the expected answer type of the question is "*location*" in Freebase. The interplay of multiple latent associations can be captured by a generative model, named joint <question, answer type> association (JQA) model. Figure 2.6 shows the graphical representation of our generative model for JQA. We first clarify the notations in the figure as follows: (1) $\mathcal{D}$ is the set of <question, answer types> pairs while $|\mathcal{D}|$ denotes the set size. A plate means replicating a process for multiple times. (2) $\theta_i$ is the $K \times 1$ mixture weights of $K$ latent association patterns, for the $i$-th <question, answer types> pair. (3) $\alpha$, a $K \times 1$ vector, is parameters in a Dirichlet prior, and serves as a constraint of the mixture weights $\theta_i$'s. A Dirichlet prior for the mixture weights tends to alleviate over-fitting problems [10]. (4) $Z_i$

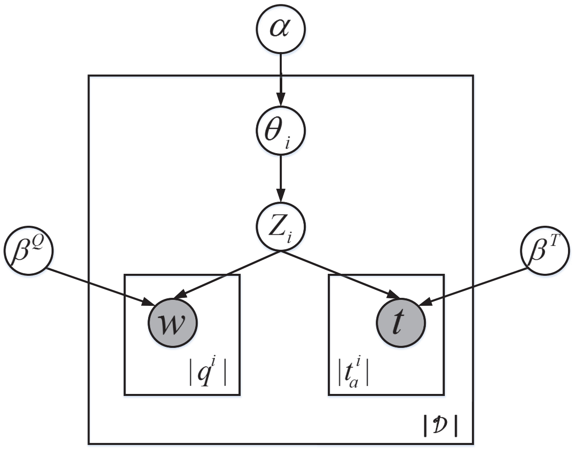


Figure 2.6: JQA generative model.

is the hidden pattern label that can explain the joint observation of the current question and its answer types. Here we assume all the words in $q$ and types in $t_a$ are generated from the same latent association, since the number of words in a question, together with its associated types, is usually very small. (5) $q^i$ and $t_a^i$ respectively refer to the $i$-th question and its answer types. (6) $\beta^Q$, a $K \times |V^Q|$ matrix, defines the probability distribution over the word vocabulary $V^Q$, under $K$ hidden patterns. $\beta^T$, a $K \times |V^T|$ matrix, defines the probability distribution over the Freebase type vocabulary $V^T$, under $K$ hidden patterns. (7) The shaded variable $w$ indicates one word observed in $q^i$ while $t$ represents one type in $t_a^i$.

Figure 2.6 conveys the generative process of a question and its answer types under multiple latent associations. Now we formally describe it as follows:

For the $i$-th <question, answer types> pair in $\mathcal{D}$,

- Draw the mixture weights of $K$ hidden association patterns:

  $\theta_i \sim \mathbf{Dir}(\alpha)$.

- Draw a pattern label: $Z_i \sim \mathbf{Mult}(\theta_i)$.

  - Draw a word for the question $q^i$ from $V^Q$:

    $w \sim \beta^Q_{Z_i,:}$ .

  - Draw a type for the answer $a^i$ from $V^T$:

    $t \sim \beta^T_{Z_i,:}$ .

We formulate the likelihood of observing all the question and its associated answer types as follows:

$$\mathcal{L} = \prod_{i \in \mathcal{D}} P(q^i, t^i_a | \alpha, \beta^Q, \beta^T) = \prod_{i \in \mathcal{D}} \int_{\theta_i} P(\theta_i|\alpha) P(q^i, t^i_a | \theta_i, \beta^Q, \beta^T) \, d\theta_i, \tag{2.6}$$

where,

$$P(q^i, t^i_a | \theta_i, \beta^Q, \beta^T) = \sum_{Z_i} P(Z_i|\theta_i) \prod_{w \in q^i} P(w|Z_i, \beta^Q) \prod_{t \in t^i_a} P(t|Z_i, \beta^T). \tag{2.7}$$

Finally, we resort to the maximum likelihood estimation approach to optimize the parameters in the model:

$$\arg \max_{\alpha, \beta^Q, \beta^T} \log \mathcal{L}. \tag{2.8}$$

### 2.5.1 Model Solution

#### Variational Inference

Due to the interdependence of the latent variables, their true posterior distributions are computationally intractable. We introduce a variational distribution $Q$ [8] in which the latent variables are independent of each other to approximate their true posterior distribution, *i.e.*, $Q(\theta, Z) = Q(\theta)Q(Z)$, where $\theta = \{\theta_i, \forall i \in \mathcal{D}\}$ and $Z = \{Z_i, \forall i \in \mathcal{D}\}$. According to the variational distribution, $Q(\theta_i) \sim \mathbf{Dir}(\gamma_i)$, $Q(Z_i) \sim \mathbf{Mult}(\phi_i)$, where $\gamma_i$ and $\phi_i$ are $K \times 1$ variational parameters.

Instead of directly maximizing $\mathcal{L}$, we can maximize the lower bound of the log likelihood under the variational distribution and Jensen's inequality:

$$\log \mathcal{L} \geq E_Q \mathrm{log} P(\mathcal{D}, \theta, Z | \alpha, \beta^T, \beta^Q) + H(Q) = \lfloor \log \mathcal{L} \rfloor, \tag{2.9}$$

where $\mathcal{D}$ denotes all the questions and their associated answer types. We expand the lower bound of the log likelihood as follows:

$$\begin{aligned} \lfloor \log \mathcal{L} \rfloor = & \sum_{i \in \mathcal{D}} E_Q \log P(\theta_i|\alpha) + \sum_{i \in \mathcal{D}} E_Q \log P(Z_i|\theta_i) \\ & + \sum_{i \in \mathcal{D}} E_Q \log P(w \in q^i, t \in t_a^i | Z_i, \beta^Q, \beta^T) \\ & + H(Q(\theta, Z)). \end{aligned} \tag{2.10}$$

Each term on the right-hand side of the above equation, is a function over the model parameters as shown in Eqn. 2.11 to Eqn. 2.14.

$$\sum_{i \in \mathcal{D}} E_Q \log P(\theta_i|\alpha) = -|\mathcal{D}| \cdot B(\alpha) + \sum_{i \in \mathcal{D}} \sum_{k} (\alpha_k - 1)[\psi(\gamma_{i,k}) - \psi(\sum_{k} \gamma_{i,k})], \tag{2.11}$$

where $B(\alpha) = \frac{\prod_k \Gamma(\alpha_k)}{\Gamma(\sum_k \alpha_k)}$ is the normalization constant of the distribution $\mathbf{Dir}(\alpha)$.

$$\sum_{\mathcal{D}} E_Q \log P(Z_i|\theta_i) = \sum_{i \in \mathcal{D}} \sum_k \phi_{i,k}[\psi(\gamma_{i,k}) - \psi(\sum_k \gamma_{i,k})]. \tag{2.12}$$

The third term

$$\begin{aligned} &\sum_{i \in \mathcal{D}} E_Q \log P(w \in q^i, t \in t_a^i | Z_i, \beta^Q, \beta^T) \\ &= \sum_{i \in \mathcal{D}} \sum_k \phi_{ik} (\sum_{w \in q^i} N_w^{q^i} \log \beta_{k,w}^Q + \sum_{t \in t_a^i} \log \beta_{k,t}^T), \end{aligned} \tag{2.13}$$

where $N_w^{q^i}$ is the frequency of word $w$ in question $q^i$. The entropy term

$$\begin{aligned} H(Q(\theta, Z)) &= -\sum_{i \in \mathcal{D}} E_Q \log Q(\theta_i|\gamma_i) + \sum_{i \in \mathcal{D}} E_Q \log Q(Z_i|\phi_i) \\ &= \sum_{i \in \mathcal{D}} [\log B(\gamma_i) - \sum_k (\gamma_{i,k} - 1)(\psi(\gamma_{i,k}) - \psi(\sum_k \gamma_{i,k}))] \\ &- \sum_{i \in \mathcal{D}} \sum_k \phi_{ik} \log \phi_{ik}. \end{aligned} \tag{2.14}$$

**Parameter Estimation**

The model parameters are estimated by using the variational expectation-maximization (EM) algorithm. In E-step, we update the variational parameters $\{\gamma\text{'s}, \phi\text{'s}\}$ while in M-step, we update the model parameters $\alpha$, $\beta^Q$, and $\beta^T$ so that $\lfloor \log \mathcal{L} \rfloor$ is maximized.

Specifically, E-step updates the variational parameters according to Eqn. 2.15 and Eqn. 2.16:

$$\phi_{i,k} \sim \exp(\sum_{w \in q^i} N_w^{q^i} \log \beta_{k,w}^Q + \sum_{t \in t_a^i} \log \beta_{k,t}^T. + \psi(\gamma_{i,k}) - \psi(\sum_k \gamma_{i,k}) - 1), \tag{2.15}$$

$$\gamma_{i,k} = \alpha_k + \phi_{i,k}. \tag{2.16}$$

During M-step, we maximize the lower bound over the parameter $\alpha$, $\beta^Q$, and $\beta^T$, by utilizing the classic L-BFGS optimization algorithm [41]. The derivatives over the parameter $\alpha$ are calculated in Eqn. 2.17.

$$\frac{\partial \lfloor \log \mathcal{L} \rfloor}{\partial \alpha_k} = |\mathcal{D}|[-\psi(\alpha_k) + \psi(\sum_k \alpha_k)] + \sum_{i \in \mathcal{D}} [\psi(\gamma_{i,k}) - \psi(\sum_k \gamma_{i,k})]. \tag{2.17}$$

We solve $\beta^Q$ and $\beta^T$ by $\beta^Q_{k,w} \propto \sum_{i \in \mathcal{D}} \phi_{i,k} N^{q^i}_w$ and $\beta^T_{k,t} \propto \sum_{i \in \mathcal{D}} \phi_{i,k}$.

We conduct E-step and M-step iteratively until the algorithm converges, indicating the current model parameters fit the observed training data.

### 2.5.2 JQA Feature Extraction

Now we discuss how to apply the learnt JQA model to evaluate the appropriateness of an answer candidate's types *w.r.t* a given question. Given a new question $q_{\text{new}}$ and an answer candidate $a_{\text{new}}$ with types $t_{a_{\text{new}}}$, we evaluate $P(q_{\text{new}}, t_{a_{\text{new}}} | \alpha, \beta^Q, \beta^T)$ using Eqn. 2.7. Similar to WAT, perplexity of observing question $q_{\text{new}}$ and answer types $t_{a_{\text{new}}}$ is defined as:

$$\text{Perplexity}(q_{\text{new}}, t_{a_{\text{new}}}) = \exp(-\frac{\log(P(q_{\text{new}}, t_{a_{\text{new}}} | \alpha, \beta^Q, \beta^T))}{|q_{\text{new}}| + |t_{a_{\text{new}}}|}).$$

This perplexity is named JQA feature for answer candidate ranking. The rationale behind JQA is similar to WAT. JQA assumes words in a question are associated with its expected answer types. We try to capture such associations by training JQA on golden pairs of questions and their expected answer types.

## 2.6 Experiments

We are interested in evaluating QuASE in terms of the following aspects: (1) How do different feature combinations affect QuASE's performance? (2) How does QuASE compare to the state-of-the-art question answering systems? (3) What are the advantages of incorporating rich semantics in KBs into text-based QA systems? (4) What are the advantages of our answer-type related features JQA and WAT? We further provide a detailed error analysis of different QA systems on questions they fail to answer.

### 2.6.1 Experimental Setup

#### QA Evaluation Datasets

We evaluate different question answering systems on two datasets: TREC questions and Bing queries. Table 2.1 shows statistics and example questions from each set.

**TREC**. The Text REtrieval Conference (TREC) had a QA track [67] since 1999. In the competition, editors first prepared some questions, and each partici-

| Datasets | Example Questions |
|---|---|
| **TREC**<br>1700 training<br>202 testing | what are pennies made of<br>what is the tallest building in Japan<br>who sang "Tennessee Waltz" |
| **Bing query**<br>4725 training<br>1164 testing | the highest flying bird<br>indiana jones named after<br>designer of the golden gate bridge |

Table 2.1: Two question sets in our experiments.

pant system then finds answers from a big collection of news articles. TREC data have been publicly available, and become popular benchmarks for evaluating QA systems. We use factoid questions from TREC 8-12 as the TREC dataset in this section. Example questions are listed in Table 2.1. For questions to which answers are not entities in KBs, such as those starting with "*when*", QuASE can be reduced to traditional web texts based QA systems without incorporating KBs. Without loss of generality, we thus eliminate those questions from the original dataset. Among the remaining 1902 questions, 202 questions from TREC 12 are used for testing and 1700 from TREC 8-11 for training. Although answers to these questions are provided by TREC, they are incomplete or sometimes incorrect for two reasons. First, the provided answers were detected from the given corpus. It is possible that some correct answers do not occur in the corpus, and therefore not included. Second, the correct answers to some questions may have changed, such as "*Who is the prime minister of France*". In order to have a fair evaluation, we revise the answers using Amazon MTurk (see [65] for detail).

**Bing query**. Although roughly 10% of the queries submitted to a search engine are with specific informational intent, only one fifth of them are formulated as well-formed questions (e.g., lack of Wh-words) [69]. Bing query dataset in Table 2.1 shows several such examples. We create Bing query dataset by selecting queries from Bing users: queries are not well-formed questions, but targeted at certain entities in Freebase. Questions here are from real search engine users and reflect more realistic information need than many existing QA datasets. We crowdsource each question to at least three experienced human labelers for collecting correct entity answers in Freebase. Once all the labelers reach an agreement on the correct answers, we save the question paired with the correct answers. In the end, we have gathered approximately 6000 question-answer pairs in total, where we randomly select around 20% for testing and 80% for training.

### Training Dataset for JQA and WAT

To train JQA and WAT proposed, a sufficiently large training dataset with golden <question, answer types> pairs is indispensable. In reality, we do not have purified data available for training. Instead, we adopt a novel alternative way to obtain labels by joining users' implicit behavioral data in query click logs and the Freebase data. Specifically, we can obtain the <query, clicked url> pairs from the query click logs. Moreover, each entity in Freebase is also linked to some urls that are related to this entity (mostly Wikipedia pages or official web sites of this entity). Hence once a user issued a query and clicked on an entity related url, we can form a <question, answer types> pair: The question is the query given by the user, while we use the Freebase types of the entity corresponding to the clicked url as the answer types. Although such collected dataset is noisy in the sense that the clicked url might not be what the user truly look for, we will show that useful answer-type related features can still be learnt from the large amount of data to benefit the ultimate QA performance. Overall we collect a dataset of around **1.3 million** <question, answer types> pairs based on Bing query logs. We have also tried directly using the training portion of each dataset in Table 2.1 to train the models. However, the training portions contain too limited questions, and features learnt from them can not perform as well as those learnt from the big query log dataset.

### Answer Candidate Ranking

For each input question, our question answering pipeline produces an answer candidate pool. To rank these candidates, we first extract all the features discussed previously, and map an answer candidate to a feature vector representation *w.r.t.*

the question. Our ranker then assigns a score to each feature vector and orders the answer candidates accordingly. We use an in-house fast implementation of the MART gradient boosting decision tree algorithm [13,27] to learn our ranker using the training set of our data. This algorithm learns an ensemble of regression trees and has shown great performance in various search ranking tasks [14].

### Evaluation Measures

We compare different QA systems on each dataset using the following metrics:

(1) Precision, Recall & $\mathrm{F}_1$: As in [25], we treat the top ranked answer candidate as the answer returned to a question. Notice that because the answer candidate pool might be empty, it is possible that no answer is returned by a QA system. As usual, precision and recall are defined as $\frac{\#(correct\ answers)}{\#(questions\ with\ answers\ returned)}$ and $\frac{\#(correct\ answers)}{\#questions}$. We also compute the $\mathrm{F}_1$ score, which is the harmonic mean of precision and recall.

(2) Mean Reciprocal Rank (MRR). Given a question, Reciprocal Rank (RR) is the reciprocal of the highest ranking position of a correct answer [55]. MRR is the average of the reciprocal ranks over questions:

$$\mathrm{MRR} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{r_i}, \tag{2.18}$$

where $N$ is the number of questions and $r_i$ is the highest ranking position of an answer to question $i$. If the true answer to a question is not detected, the RR for the question is 0.

### Alternative QA systems

We compare QuASE with existing web texts based and KB-based QA systems. Due to unavailability of most existing QA systems, we select one representative from each category to compare.

(1) Web texts based QA system: AskMSR+ [65]. Early web texts based systems like Mulder [35] and AskMSR [11] have demonstrated that by leveraging the Web redundancy, a simple system can compete with those conducting sophisticated linguistic analysis of either questions or answer candidates[1]. We compare QuASE with AskMSR+ [65], an advanced version of AskMSR with two main changes. First, instead of reformulating a question according to some statement-like patterns as query terms to a search engine, AskMSR+ issues the question directly, as [65] found out that question reformulation no longer helps retrieve high-quality snippets. This may be due to the better performance of modern search engines, as well as the increased coverage of various community QA sites. Second, instead of using only N-grams extracted from snippets as answer candi-

[1] For instance, AskMSR is one of the top systems in TREC-10 [12].

dates, AskMSR+ requires candidates to be specific types of named entities that can match the question. For instance, only a location entity can be a candidate to the "*where*" questions. With these design changes and other enhancements detailed in [65], AskMSR+ increases MRR by roughly 25% on the TREC dataset, compared to the original AskMSR system, and is thus a solid baseline.

(2) KB-based QA system: Sempre [6, 7]. Sempre [6] and ParaSempre [7] develop semantic parsers to parse natural language questions to logical forms, which are subsequently executed against knowledge bases. They have shown great performance for questions that are directly coined based on relation tuples in KBs. Here we test them on questions that are not necessarily answerable in KBs. The implementation of their systems is publicly available[2], as well as the well-trained systems. We have applied the well-trained systems to our evaluation datasets, and also re-trained the systems on our training datasets. We finally show the best performance we could obtain by these two systems.

### 2.6.2 Extensive Feature Comparison

Results of different feature groups in QuASE are summarized in Table 2.2. We have made the following observations:

[2] https://github.com/percyliang/sempre .

| | Bing query | | | | TREC | | | |
|---|---|---|---|---|---|---|---|---|
| Features in QuASE | Precision | Recall | $F_1$ | MRR | Precision | Recall | $F_1$ | MRR |
| Count | 0.5513 | 0.5262 | 0.5384 | 0.6111 | 0.5446 | 0.5446 | 0.5446 | 0.6224 |
| TR | 0.5243 | 0.5004 | 0.5121 | 0.5880 | 0.4554 | 0.4554 | 0.4554 | 0.5740 |
| JQA | 0.1358 | 0.1296 | 0.1326 | 0.2737 | 0.1980 | 0.1980 | 0.1980 | 0.3579 |
| WAT | 0.1709 | 0.1631 | 0.1669 | 0.3100 | 0.2624 | 0.2624 | 0.2624 | 0.4049 |
| Count+TR | 0.5674 | 0.5416 | 0.5512 | 0.6239 | 0.5644 | 0.5644 | 0.5644 | 0.6425 |
| Count+TR+WAT | 0.5926 | 0.5657 | 0.5788 | 0.6370 | 0.5693 | 0.5693 | 0.5693 | 0.6513 |
| Count+TR+JQA | 0.5764 | 0.5502 | 0.5630 | 0.6296 | 0.5743 | 0.5743 | 0.5743 | 0.6476 |
| **ALL** | **0.5962** | **0.5691** | **0.5823** | **0.6402** | **0.5792** | **0.5792** | **0.5792** | **0.6532** |
| Count+TR$^{-\text{KBs}}$ | 0.5638 | 0.5382 | 0.5507 | 0.6187 | 0.5495 | 0.5495 | 0.5495 | 0.6281 |

Table 2.2: Comparison among different feature combinations.

(1) As we discussed previously, Count is a significant indicator of the true answer to a question. Although the simplest feature, it performs the best compared with other separated features. Such observation is consistent with the conclusion in [11] that based on data redundancy: Simple techniques without complicated linguistic analyses can work well in QA systems.

(2) Combined with the Textual Relevance (TR) features, Count+TR can further boost the performance by around 3% under $F_1$ and MRR, on both datasets. TR can be effective when some wrong answer candidates appear very frequently but their context information does not match the question. For example, given a question "*Who first landed on the moon*", entity "*American*", the nationality of the astronaut, occurs quite frequently in the retrieved texts. However, the textual information related to "*American*" in Freebase can distinguish itself from the true answer, as its description is about American people, rather than people first landing on the Moon.

(3) Comparing ALL (Count+TR+WAT+JQA) with Count+TR, our answer-type related features WAT and JQA turn out to be effective: They further improve the performance by around 2% on TREC and by 2% $\sim$ 5% on Bing query across all the metrics.

(4) The advantages of incorporating rich semantics in KBs into QuASE can be observed by comparing Count+TR+WAT+JQA with Count+TR$^{-\mathrm{KBs}}$, where TR$^{-\mathrm{KBs}}$ denotes the two textual relevance features without involving entity description texts from KBs, i.e., $\cos(v_q, c_a) + \cos(s_q, c_a)$ in Figure 2.4. With around 5% improvements on both datasets, utilizing rich semantics in KBs can obviously benefit QA performance.

### 2.6.3 Comparison against Either KB or Text Based QA

Table 2.3 shows the performance of different QA systems. Compared with AskMSR+ system, which also utilized the count of an answer candidate for ranking, the single feature Count in QuASE, as shown in Table 2.2, performs better by at least 10% in terms of $F_1$. The potential reason is that through linking answer candidates to KBs, same answer candidates with different surface forms can be automatically merged, and therefore, redundancy and noise among answer candidates can be significantly reduced. On Bing query, QuASE with all the features can obtain around **54%** improvement on $F_1$ and **20%** improvement under

| | Bing query | | | | TREC | | | |
|---|---|---|---|---|---|---|---|---|
| QA Systems | Precision | Recall | $F_1$ | MRR | Precision | Recall | $F_1$ | MRR |
| **QuASE** | **0.5962** | **0.5691** | **0.5823** | **0.6402** | **0.5792** | **0.5792** | **0.5792** | **0.6532** |
| AskMSR+ [65] | 0.3782 | 0.3760 | 0.3771 | 0.5337 | 0.4925 | 0.4901 | 0.4913 | 0.6223 |
| Sempre [6, 7] | 0.2646 | 01940 | 0.2239 | 0.2372 | 0.1567 | 0.1040 | 0.1250 | 0.1437 |

Table 2.3: Comparison among different QA systems.

MRR, while on TREC, it can achieve about **18**% improvement under $F_1$ and **5**% improvement under MRR. The great improvement further verifies the advantages of employing KBs as an auxiliary into web texts based QA systems. The improvement on $F_1$ measure is generally higher. MRR takes into account the entire candidate list while $F_1$ focuses on the first ranking position, implying AskMSR+ has a poor ability to rank the true answer at the top position. We will give more detailed analysis on QuASE and AskMSR+ in the next section.

Sempre systems formulate a question into logical forms to be executed against KBs. They are among the state-of-the-art KB-based QA systems on questions that are guaranteed answerable in KBs. However as shown in Table 2.3, the performance of Sempre is generally much lower than our system under all the measures. Similar performance of Sempre on TREC has also been reported in [25]. There are potentially two reasons why Sempre systems cannot perform well on Bing query and TREC dataset. First, the knowledge required to answer questions might not exist in KBs. For example, given one example question in TREC "*What color is indigo*", there is no relationship between the true answer

| The distribution of QuASE's rank on failed questions | | | | | |
|---|---|---|---|---|---|
| Dataset | $r = \infty$ | r=2 | r=3 | r=4 | $r \geq 5$ |
| Bing query | **63.76%** | 12.86% | 7.62% | 2.86% | 11.90% |
| TREC | **53.57%** | 14.29% | 12.50% | 3.57% | 16.07% |

| The distribution of AskMSR+'s rank on failed questions | | | | | |
|---|---|---|---|---|---|
| Dataset | $r = \infty$ | r=2 | r=3 | $r = 4$ | $r \geq 5$ |
| Bing query | 13.82% | **27.88%** | 21.66% | 11.75% | **24.88%** |
| TREC | 19.64% | **35.71%** | 14.29% | 10.71% | **19.64%** |

Table 2.4: Error analysis of QuASE and AskMSR+.

"*blue*" and "*indigo*" corresponding to "*alias*" or "*synonym*" although both entities "*blue*" and "*indigo*" exist in Freebase. Second, many questions, especially those not well-formed ones in Bing query, are generally hard to be transformed to logical forms and thereby unsuccessful to obtain answers by executing logical forms against KBs. Since SEMPRE turns out to be much less effective on both datasets, we will not include it in the following experiments.

### 2.6.4 System Comparison on Failed Questions

Now we provide detailed analysis of QuASE and AskMSR+ on their respective failed questions. We first define the failed questions for each system: AskMSR+'s failed questions are those where AskMSR+ ranks the true answer lower than QuASE, whereas QuASE's failed questions as those where QuASE ranks the true answer lower than AskMSR+. If there are multiple answers to a question, we consider the highest-ranked answer in defining failed questions.

| | Bing AQS | | |
|---|---|---|---|
| Systems | MRR | $F_1$ | \|AQS\| |
| QuASE | **0.8583** | **0.7629** | 869 |
| AskMSR+ [65] | 0.6230 | 0.4388 | 998 |
| | TREC AQS | | |
| QuASE | **0.8195** | **0.7267** | 161 |
| AskMSR+ [65] | 0.7023 | 0.5530 | 179 |

Table 2.5: Results on answerable question sets.

Table 2.4 shows the rank distribution of the true answer given by each system on their failed questions. $r = \infty$ means the true answer is not in the candidate list. For those questions where QuASE performs worse than AskMSR+, in most cases, we do not have the answer in the candidate list, i.e., $r = \infty$. However, for those AskMSR+ performs worse, in most cases, AskMSR+ did find the true answer, but ranks it lower than QuASE. This result verifies that compared with AskMSR+, features in QuASE for ranking are effective as long as true answers are in the candidate list.

Due to the imperfectness of any answer candidate generation technique, including entity linking or named entity detection, true answers to many questions might not be detected and included in the candidate list. Therefore, no matter how effective a ranking feature is, the performance on questions without true answers in the candidate list, cannot be boosted. For each method, we define the answerable question set (AQS) as those questions with true answers included in their candidate list. Table 2.5 shows the performance of QuASE and AskMSR+

on their respective answerable question sets. Although the answerable questions by QuASE are slightly fewer than those by AskMSR+ system, the performance of QuASE is significantly better than AskMSR+ with a **17%** $\sim$ **38%** improvement on MRR and a **31%** $\sim$ **74%** improvement on $F_1$. For QuASE, how to improve entity linking performance, in order to include true answers in the candidate list, can be important to further improve QA performance. However the study of that problem is beyond the scope of this dissertation and we leave it to future work.

# Chapter 3

# Table-based QA

Tables on the Web contain a huge body of semi-structured information about wide-ranging topics, and naturally serve as a rich knowledge pool to answer open-domain questions. Employing tables as the answer source is entitled with the following advantages:

1. Web tables have schema. Different from unstructured texts, each table comes with its own schema; therefore it is much easier to interpret the entities and relations contained in a table. For example, in Table 1.1, entity "*France*" can be easily interpreted as a country by its column name, and the column name pair <`Country`, `MainLanguage`> [1] is likely to indicate main languages spoken in a country. Such schema and structure provide valuable clues for answering questions.

[1] We will use this representation to distinguish column names from other texts.

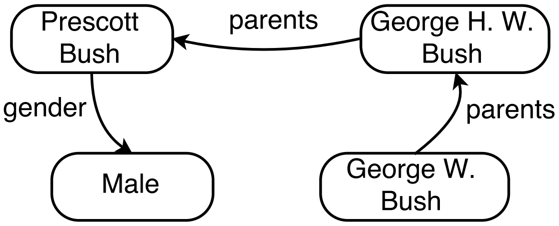


Figure 3.1: Freebase representation of relation "Grandfather" is complex.

2. Web tables complement knowledge bases. On the one hand, knowledge bases are incomplete, and web tables might contain information not covered by knowledge bases. On the other hand, the semantic structure of a KB is often more complex than that of a table: For example, Freebase stores the simple fact "*Prescott Bush is George W. Bush's grandfather*" in a complex structure, as shown in Figure 3.1. On the contrary, tables often represent relations in a more straightforward way, with each column for a relation such as `Father`, `Mother`, `Grandfather` etc. Such straightforwardness makes answering questions like "*Who is X's grandfather*" much easier.

In this chapter, we propose an end-to-end framework to identify table cells that can answer a natural language question. The core problem is to match a natural language question with the information in tables. We propose a unified chain representation for both the input question and table cells. The *question chain* starts from an identified topic entity in the question (e.g., "*France*"), goes through an edge labeled with the question pattern, and points to the to-be-determined answer. The question pattern is just the input question excluding the topic entity, and expresses the relation between the topic entity and the answer. On the other hand, we also represent the semantic relation between any two cells in the same row of a table as a *relational chain.* For example, the semantic relation between "*France*" and "*French*" in Table 1.1 is represented as "France$\xrightarrow{\texttt{Country}}\bigcirc\xrightarrow{\texttt{MainLanguage}}$French", where $\bigcirc$ is a pseudo-node referring to the particular row (to be discussed later in Section 3.1). Question answering is then reduced to finding the relational chains that best match with the question chain.

There are two main challenges in identifying the correct answer cells: (1) *Among millions of tables, how to retrieve the relevant ones that may contain answers*? (2) *How to precisely locate answer cells in a relevant table*? With the chain representations, we tackle the first challenge as follows: (a) Detect topic entities in the given question, retrieve tables that contain the topic entities, and generate an initial set of candidate (relational) chains pointing to possible answer cells; (b) Issue the input question to a search engine, and use the returned snippets to select relevant candidate chains from the initial set.

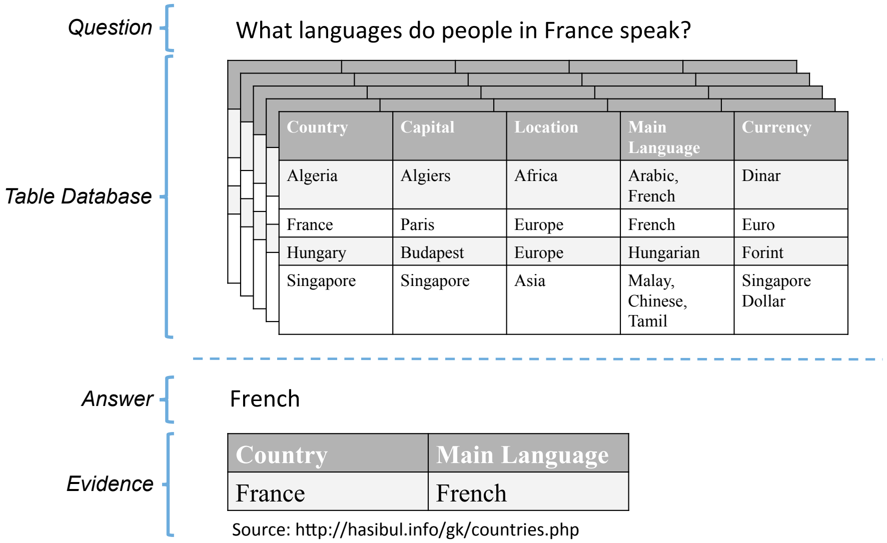
To deal with the second challenge, we develop techniques for more fine-grained matching between candidate chains and the question chain, i.e., to find which candidate chain represents the relation expressed in the question. Simple bag-of-words based matching is insufficient because few words are shared by a candidate chain and the input question. We therefore employ deep neural networks to map both the question and the information about a candidate chain into a common semantic space. We adopt information about a candidate chain from three perspectives: answer type, pseudo-predicate, and entity pairs, which we shall detail in Section 3.3. We conduct extensive experiments and show that table cells containing correct answers can be effectively identified using the proposed framework. Moreover, combining our table cell search framework with state-of-the-art KB-based QA systems can achieve even better performance, showing that the two different kinds of systems complement each other.

## 3.1 Preliminaries

Now we give a more formal task description, followed by the high-level idea of our approach.

### Task Definition

Given a natural language question, we aim at answering it by retrieving table cells that contain correct answers, from a large collection of tables. Figure 3.2 shows a concrete example of this task. Given a question, such as "*What languages do people in France speak*", we assume that the answer lies in a particular cell



| Country | Capital | Location | Main Language | Currency |
|---|---|---|---|---|
| Algeria | Algiers | Africa | Arabic, French | Dinar |
| France | Paris | Europe | French | Euro |
| Hungary | Budapest | Europe | Hungarian | Forint |
| Singapore | Singapore | Asia | Malay, Chinese, Tamil | Singapore Dollar |

| Country | Main Language |
|---|---|
| France | French |

Source: http://hasibul.info/gk/countries.php

Figure 3.2: An example of mining table cells for QA.

in one of the collected web tables. As the column `MainLanguage` can be interpreted as main languages spoken in a country, we would like to identify the table cell "*French*" lying under the `MainLanguage` column and in the same row as "*France*". The identified answer as well as a small sub-table composed of the related cells and column names can be presented, together with the URL of the source table for further exploration.

As implied in this example, we make two assumptions when attacking this task. First, we take an entity-centric view: The targeted questions in this task are those that contain at least one entity, named topic entity. This view has been commonly adopted in recent question answering studies [6, 7, 75]. Second, we assume that the relationship between the topic entity in the question and the answer can be represented by the information in a single table. A cell matched with a topic entity and that matched with an answer should occur in the same row. These two assumptions are made because of the unique setting of our task. Using a large collection of web tables as the sole information source for answering questions poses both advantages and challenges, especially when compared to relying on a well-curated knowledge base. For instance, instead of having a rigid schema that define all possible relations between entities using a fixed set of predicates, as usually seen in a knowledge base, more diversified types of relations are described by a large number of column names existing in millions of tables. Due to the Web redundancy and coverage, it's more likely that a pair of column names can match with a given question, in contrast to a single predicate in a knowledge base[2]. On the other hand, it is challenging to find relevant tables that might contain answers, from a large collection of independent tables. Diversified forms of relations presented by the column names also increase the difficulty of determining whether they are equivalent to the natural-language description of a question.

[2] More sophisticated analysis of multiple tables and their column names will be needed for complicated, highly compositional questions, which we leave for future work.

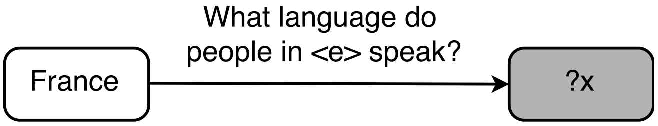


Figure 3.3: Graph representation of a question chain.

## Approach

Our strategy is to formulate the task as a joint entity and relation matching problem. Here we present the basic idea with intuitive graphical views, leaving more details to be introduced in Section 3.2.

For each input question, we apply entity linking [72] to identify possible entities in the question. Each identified entity defines the *topic entity* and *question pattern*, where the former is just the canonical name of the entity and the latter is the rest of the question after removing the entity mention. For instance, assuming "*France*" is the topic entity identified in "*What languages do people in France speak*", the question pattern is simply "*What languages do people in <e> speak*", where <e> indicates the slot for the topic entity. A question can then be naturally represented as a two-node graph as shown in Figure 3.3, where the topic entity node points to the answer node, and the edge is labeled with the question pattern. We call this graph a *question chain*. Notice that a question may produce multiple question chains because it can contain more than one topic entity.

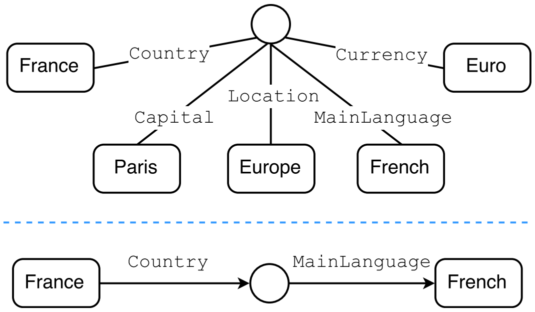


Figure 3.4: A table row graph (top) and a relational chain (bottom).

Each table essentially defines a "*mini knowledge base*" – each row describes multi-relations among the cells it contains. Following [51], a row in a table can be represented by an undirected row graph that connects each cell to a pseudo-node, where the edge is labeled with the corresponding column name (i.e., relation). The pseudo-node[3] simply indicates the row where the cells come from. Figure 3.4 (top) shows an example of this graph. The circle is a pseudo-node for this particular row, connecting all the cells with their column names as edges.

A row graph can be decomposed into several *relational chains*. Each relational chain connects two cell nodes by starting from one node, going through the pseudo-node and then pointing to the other node. Similarly, the edges are labeled with the corresponding column names. Figure 3.4 (bottom) shows one example of this construction. Hereafter, we denote the starting cell of a relational chain as the *topic cell*, and the ending cell as the *answer cell*.

[3] The pseudo-node can be analogous to the compound-value-type design in Freebase, which is a standard way to encode multi-relations in RDF triples.

Reminiscent of our second assumption mentioned previously, we shall map the question to a pair of cells in the same row of a table. Having represented the question as a question chain $q$ and a pair of cells as a relational chain $r$, finding a table cell to answer the question is reduced to a chain matching problem. Comparing Figure 3.3 and 3.4 (bottom), for $q$ and $r$ to be matched, the topic entity in $q$ has to be matched with the topic cell of $r$, and the question pattern in $q$ needs to be implied by both inward and outward relations of $r$. After that, the answer cell of $r$ can thus be extracted. In cases where multiple topic entities are identified, all the corresponding relational chains will be jointly considered.

## 3.2 Table Cell Search

Our end-to-end table cell search framework consists of three main steps. We first generate a set of candidate (relational) chains. To do that, we detect topic entities in an input question, match the topic entities with tables to find topic cells, and generate candidate chains from each topic cell. The first step results in a large set of candidate chains, many of which are irrelevant to the question. Therefore, in the second step, search engine snippets providing more information about the input question are utilized to help filter out irrelevant chains. We further employ deep neural networks to match the question and a candidate chain in a common semantic space. Overall candidate chains are ranked based on a set of carefully derived features. An answer cell can subsequently be extracted from each top ranked candidate chain.

## Candidate Chain Generation

Given a question, one can first apply named entity recognition [50] to identify topic entities, and then retrieve all the table cells that contain any of the topic entities via substring matching. Since an entity often has many aliases (e.g., "*Barack Obama*" and "*President Obama*"), it is beneficial to match table cells with not only the entity mention in the question, but also other entity aliases. Fortunately, open-domain knowledge bases like Freebase store common aliases of an entity; therefore, in our framework, we link each topic entity in the question to Freebase and fetch its alias list. We employ a state-of-the-art entity linking system [72], which is designed particularly for short and noisy texts and has been shown especially suitable for topic entity linking in natural language questions [75]. Table cells containing any alias of a topic entity are retrieved as topic cells. In the rest of this chapter, any statement like a table cell *contains* an entity, means the cell contains the entity mention or any of its aliases (if available).

As discussed in Section 3.1, given a topic cell, we assume the answer cell lies in the same row. But it is not known which one is the answer cell at this stage. We therefore first blindly generate a candidate chain for each possible answer cell, and leave candidate chain ranking for later. Consider Table 1.1, suppose we have identified the "*France*" cell under the `Country` column as a topic cell, we will generate 3 candidate chains, one for each cell in the same row ("*Paris*", "*Euro*", and "*French*"). Each candidate chain starts from the topic cell, goes through the corresponding column names in the table, and points to the candidate answer cell. We repeat the same procedure for every topic cell, and end up with a large (can be hundreds of thousands) set of candidate chains.

## Coarse-Grained Pruning

In the first step, all relational chains related to any identified topic entity are generated as candidates. One consequence is that many of them are not truly relevant to the input question, e.g., "France$\xrightarrow{\texttt{From}}\bigcirc\xrightarrow{\texttt{Actress}}$Sophie Marceau" in Figure 2.3, which is generated from a table about French actresses. We now prune the candidate chain set for the subsequent ranking model as well as for efficiency consideration. To do that, we need to evaluate the *relevance* of a candidate chain to the input question. However, both the question and a candidate chain usually only contain a few words, and have even fewer words in common. If we directly compare them word-by-word, many relevant chains will be deemed as irrelevant. Therefore, we employ search engine snippets to enrich the question, a common technique used in information retrieval related tasks [62].

We issue the input question $q$ as a query to a commercial search engine, then compute the word frequency vector based on the top-50 returned snippets, denoted as $w_q$. For each candidate chain $c$, we also compute its word frequency vector, denoted as $w_c$, based on the table caption, topic/answer cells, and column names on it. Two vector similarities are adopted: $\text{cosine}(w_c, w_q)$ and $\text{InterScore}(w_c, w_q)$ where $\text{InterScore}(w_c, w_q) = \|w_c \odot w_q\|_0$ computes the number of unique words in common. Here $\odot$ is the element-wise product and $\|\cdot\|_0$ is the $l_0$ norm of a vector. Candidate chains with both high $\text{cosine}(w_c, w_q)$ and $\text{InterScore}(w_c, w_q)$ are kept. These two measures inspect vector similarity from different aspects and make a more restrictive selection of relevant candidate chains. If $w_c$ deviates far from $w_q$, the corresponding candidate chain is regarded as irrelevant and thereby discarded.

## Deep Chain Inference

After relevant candidate chains to the input question are collected, we perform deeper inference on whether a candidate chain can represent the natural language statement of the given question. On the candidate chain side, we explore its information from three perspectives: answer type, pseudo-predicate, and entity pairs. We use the question pattern defined in Section 3.1 to represent what factual information is being asked in the question, regardless of the specific topic entity. In order to capture the syntactically different but semantically equivalent ways of stating the same question, as well as to handle the mismatch between natural language sentences and table schemas, we construct deep neural networks to evaluate the matching degree between a question pattern and each perspective of a candidate chain in a common semantic space. Finally, for each candidate chain, we develop a set of features for downstream ranking so that candidate chains pointing to correct answer cells can be ranked as high as possible. Next we introduce our methodology for deep chain inference in greater details.

## 3.3 Chain Inference via Deep Neural Networks

Coarse-grained pruning gives a set of candidate chains that are likely relevant to the input question. We now need to conduct deeper inference on which candidate chain can actually answer the question. Each candidate chain is inspected from the following perspectives, which give clues about whether the candidate chain matches the question.

- Answer type. Answer type is defined as the column name corresponding to the answer cell of a candidate chain. Obviously, the answer type `MainLanguage` matches the question "*What languages do people in France speak*" better than the answer types in other candidate chains such as `Currency` and `Capital`.

- Pseudo-predicate. While answer type gives information about the answer cell, the relation between the topic cell and the answer cell is also critical for identifying the answer. Predicate is a term representing the relation between two entities in a knowledge base. For example, *President-Of* is a predicate between Barack Obama and the United States. Analogically, we use the column name pair, e.g., `Country-MainLanguage`, on each candidate chain to form a *pseudo-predicate*. A pseudo-predicate connects a topic cell to an answer cell and represents a certain relation between them. Intuitively, the pseudo-predicate `Country-MainLanguage` matches questions asking about languages spoken in a country better than other pseudo-predicates such as `Country-Population` and `Country-Currency`.

- Entity pairs. Entity pairs from two columns in a table shall have the same relation. For example, all the entity pairs {<Algeria, Arabic>, <Egypt, Arabic>, <France, French>, <Germany, German>, ...} are about some country and its main language. The entity pairs from the same columns as the topic and answer cells in a candidate chain therefore provide significant information about the implicit relation expressed in the chain, complementing the pseudo-predicate.

Because the question pattern represents what information is being asked irrespective of the topic entity, intuitively a correct candidate chain should match

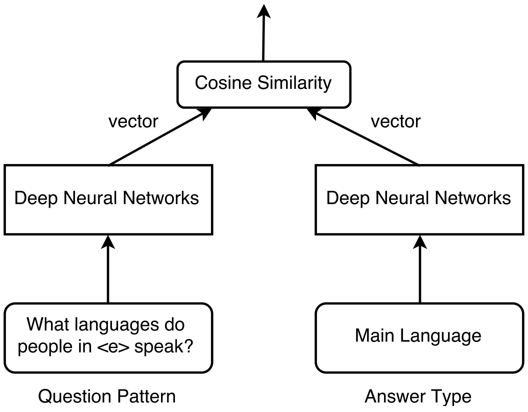


Figure 3.5: Semantic similarity between question pattern and answer type.

the question pattern from the above three perspectives. Since a question pattern usually share few common words with each perspective, we can hardly build effective matching models based on word-level information. For example, the entity pair <Spain, Spanish> shares no common word with the question pattern "*What languages do people in <e> speak*", yet they are about the same relation, i.e., the spoken language of a country. Therefore, we first map them into a common semantic space, where semantically similar texts will be represented as similar fixed-length vectors. Text embedding via neural networks (more broadly termed "deep learning for natural language processing") has been extensively studied recently and demonstrated to excel at capturing the syntactically different ways of stating the same meaning [28, 31, 33, 64, 74, 75]. Hence we employ deep neural networks to embed question patterns and various perspectives about candidate chains and measure their similarity in the semantic space.

Take answer type as example. Figure 3.5 shows the architecture to match the question pattern with the answer type of a candidate chain. Two deep neural networks are constructed respectively to embed both the question pattern and the answer type. We then compute the cosine similarity of the embedded semantic vectors as the matching degree between the given question and the answer type. The same model architecture is applied to match other perspectives of candidate chains with a question pattern, but model parameters are separately learned for each perspective using the corresponding inputs.

There could be different designs for the deep neural network in Figure 3.5. We select the Convolutional Deep Structured Semantic Model [4] (C-DSSM) developed in [60] because of its great potential that has been demonstrated in many information retrieval related tasks [28,59,75]. Figure 3.6 illustrates the C-DSSM. It takes a word sequence such as "*What languages do people in <e> speak*" as input. The word hashing layer decomposes a word into a vector of letter-trigrams. For example, word "*speak*" is converted to a bag of letter-trigrams {#-s-p, s-p-e, p-e-a, e-a-k, k-e-#} where "#" is the word boundary symbol. All the unique letter-trigrams in the dataset form the letter-trigram vocabulary of size $N$ and each word will be converted to an $N \times 1$ vector (e.g., $f_t$) with each component being the frequency of a letter-trigram in the word. Following this, a convolutional layer concatenates

[4] Publicly available at: `http://research.microsoft.com/en-us/projects/dssm/`.

the letter-trigram frequency vectors in a context window of size 3 and projects it to a local contextual feature vector, e.g., $h_t = \tanh(W_c[f_{t-1}, f_t, f_{t+1}]), \forall t = 1, ..., T$. Then a max pooling layer is deployed to extract the most salient local features and forms a fixed-length global feature vector. This global feature vector is subsequently fed to a non-linear feed-forward neural network layer, which outputs the final semantic representation of the original input word sequence (a question pattern or word sequence representing answer type, pseudo-predicate, or entity pairs), i.e., $y = \tanh(W_s v)$.

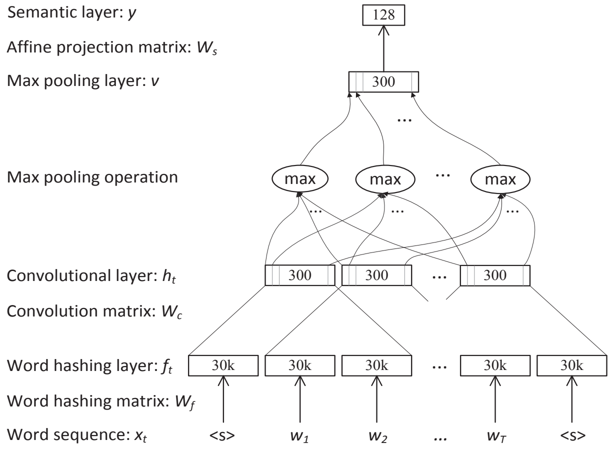


Figure 3.6: Architecture of C-DSSM [60].

We instantiate the deep neural networks in Figure 3.5 with C-DSSM. Three matching models shall be learned for question pattern respectively paired with answer type, pseudo-predicate, and entity pairs. To train these semantic matching models, we need to collect three training sets, formed by pairs of question patterns and their true answer type/pseudo-predicate/entity pairs. Unfortunately, no such training sets are readily available. Instead, question-answer pairs are available in existing question answering datasets. Based on the question-answer pairs, our mechanism to construct training sets is as follows: For each question in a question-answer set, we first match both the topic entities and the answer entities with table cells. Then we extract the relational chains connecting a topic cell to an answer cell. In order to effectively train the model, we shall obtain a cleaner training set; therefore we conservatively keep only the relational chains with both top-20 $\cos(w_c, w_q)$ and $\text{InterScore}(w_c, w_q)$ scores. We choose top-20 because it induces high-quality training sets via manual checking. For each selected chain, we extract its answer type, pseudo-predicate, and entity pairs to respectively pair with the corresponding question pattern, and finally form the training sets.

In each case, we randomly sample 5% pairs as the held-out set, and the rest as the training set. Hyper-parameters of the C-DSSM, such as the number of neurons in each layer, are selected using the held-out set. Consistent with other studies employing deep neural networks [32, 75], we observe that the C-DSSM is insensitive to the hyper-parameters in a reasonable range (e.g., 300 $\sim$ 500 nodes in the semantic layer, and learning rate 0.05 $\sim$ 0.005). We leave more details about the C-DSSM related model learning to [32, 59, 60].

## 3.4 Answer Cell Ranking

We summarize our developed features to rank candidate chains as below.

### 3.4.1 Shallow Features

Shallow features consider the matching degree between a question and a candidate chain at the word level.

As introduced in Section 3.2, for each question, we prepare the word frequency vector $w_q$ based on the top-50 snippets returned by a search engine. On the candidate chain side, we construct the word frequency vector $w_c$ based on its table caption, column names and table cells. Two similarity measures are applied:

- $\text{cosine}(w_q, w_c) = \frac{w_q \cdot w_c}{\|w_q\|_2 \|w_c\|_2}$,
- $\text{InterScore}(w_q, w_c) = \|w_q \odot w_c\|_0$,

where InterScore stands for the intersection score and calculates the number of overlapped words in $w_q$ and $w_c$.

### 3.4.2 Deep Features

Three perspectives are investigated: answer type, pseudo-predicate, and entity pairs. For a candidate chain $c$, the word sequence of the three types of information is denoted as $c_a$, $c_p$, and $c_e$, respectively. With the trained C-DSSM model, we capture the high-level features of $c_a$, $c_p$, and $c_e$ respectively as $y(c_a)$, $y(c_p)$, and $y(c_e)$. On the question side, we input the word sequence representing the question pattern ($q_p$) and extract its high-level features $y(q_p)$. Additionally, we concatenate the topic cell with the pseudo-predicate $c_p$, noted as $c_{p*}$, and compare it with the original question sentence $q$. This feature specifically takes into account the effect of topic entities on semantic matching, which was also adopted in [75].

The semantic similarities between information on the question side and that on the candidate chain side are calculated as below and incorporated as features in our framework:

- DeepType: $\text{cosine}\big(y(q_p), y(c_a)\big)$,
- DeepPredicate: $\text{cosine}\big(y(q_p), y(c_p)\big)$,
- DeepEntityPairs: $\text{cosine}\big(y(q_p), y(c_e)\big)$,
- DeepSentence: $\text{cosine}\big(y(q), y(c_{p*})\big)$.

Here DeepSentence can be regarded as a variation of DeepPredicate. For the word sequence on entity pairs $c_e$, we include two variations: (1) $c_e$ is the word sequence generated by concatenating all entity pairs under the two columns of a candidate chain, in the order of their row indices; (2) $c_e$ is the word sequence corresponding to a single entity pair, and we average $\text{cosine}\big(y(q_p), y(c_e)\big)$ over all entity pairs corresponding to a candidate chain as DeepEntityPairs.

Overall, for each candidate chain, features investigated in our framework include shallow features {cosine, InterScore} and deep features {DeepType, DeepPredicate, DeepEntityPairs, DeepSentence }. Nevertheless, this framework is readily extensible by incorporating other features.

### 3.4.3 Ranking

Based on the above features, we map a candidate chain to a feature vector *w.r.t.* the question. A ranking algorithm is then deployed to order candidate chains. For each question, in the coarse-grained pruning stage, we select candidate chains with both top-3K cosine similarity and top-3K InterScore, in order to reduce noise and speed up the ranking process. For training, we label each candidate chain as correct if its answer cell contains at least one gold-standard answer and incorrect if otherwise. We adopt an in-house fast implementation of the MART gradient boosting decision tree algorithm [13, 27], which learns an ensemble of regression trees and has shown great performance in various tasks [14].

## 3.5 Experiments

We evaluate our table cell search framework in the open-domain question answering setting from the following aspects: (1) How do different feature combinations affect the search performance? (2) What are the advantages of the features learned by deep neural networks? Which deep feature is the most discriminative? (3) How does our table cell search framework compare with state-of-the-art QA systems? Can the two kinds of systems complement each other?

### 3.5.1 Experimental Setup

#### Table Sets

We test our framework using two sets of tables as answer sources: one is extracted from Wikipedia pages whereas the other is from the broader Web, denoted respectively as WikiTables and AllTables. We employ the table extractor used in [71], which extracts HTML tables from the web crawl and deploys a classifier to distinguish relational tables from other types of tables, such as layout or formatting tables. This approach is also similar to the one used in [16]. We do not discuss the details here since it is not the main focus of our table search framework. WikiTables contains around 5 million tables whereas AllTables contains roughly 99 million tables, much larger but also noisier than WikiTables.

#### Question Answering Evaluation Sets

To test our table cell search framework, we pick the popularly studied open-domain QA setting: Open-domain QA datasets and systems are available, which

| WebQ Splits: | WebQ Examples: |
|---|---|
| 2,032 testing | who did the voice for lola bunny |
| 3,778 training | in what coutries do people speak danish |

| BingQ Splits: | BingQ Examples: |
|---|---|
| 1,164 testing | cherieff callie voice |
| 4,725 training | boeing charleston sc plant location |

Table 3.1: Statistics of question sets.

makes the evaluation and comparison with different systems very straightforward. Nevertheless, our framework can be extended to closed-domain question answering as long as the table sources are domain-specific. In our experiments, two question-answer sets are employed, where the gold-standard answer set of each question contains one or more entities in Freebase. We show the statistics and example questions in Table 3.1.

**WebQ**. WebQuestions (WebQ) is developed by [6] and consists of 5,810 question-answer pairs. The questions are collected using the Google Suggest API and answers are obtained from Freebase with the help of Amazon MTurk. The dataset is split into training and testing sets, respectively containing 3,778 questions (65%) and 2,032 questions (35%). Since its release, WebQ has been widely used in testing an array of open-domain QA systems [6, 7, 25, 74, 75].

**BingQ**. BingQ is constructed in [62]. Questions in this dataset are mined from search engine logs, and therefore not necessarily well-formed yet reflect realistic information needs of the general public. Each question is crowdsourced to obtain

| Datasets | WikiTables | AllTables |
|---|---|---|
| WebQ | Training: 2,551 (68%)<br>Testing: 1,362 (67%) | Training: 2,818 (75%)<br>Testing: 1,507 (74%) |
| BingQ | Training: 2,794 (59%)<br>Testing: 679 (58%) | Training: 3,235 (68%)<br>Testing: 793 (68%) |

Table 3.2: Table coverage of question sets.

correct entity answers. One distinction between BingQ and WebQ is that the knowledge required to answer a question in BingQ (i.e., the relation between its topic entity and answer entity) might not exist in knowledge bases. There are in total 5,889 question-answer pairs, in which 4,725 (80%) are randomly selected for testing and 1,164 (20%) for training in [62].

**Table-answerable question sets**. The construction of the above evaluation sets does not refer to our table collections, therefore the knowledge required to answer a question might not exist in WikiTables or AllTables. Such questions are unanswerable using the table collections, no matter what algorithms are developed to search table cells. In order to evaluate the effectiveness of our framework, we need to remove those unanswerable questions. It is difficult, if not impossible, to automatically verify whether a question is answerable by a huge set of tables. We adopt the following mechanism to make an approximation: We will keep a question if and only if at least one of its topic entities and at least one of its answer entities simultaneously exist in the same row of some table. The percentage of questions left in each evaluation set is defined as its coverage by a particular table set, which are shown in Table 3.2. As one can see, a large proportion of questions in both sets, around 58% to 75%, are covered by tables, and AllTables covers roughly 10% more questions than WikiTables. We next evaluate our framework only on questions covered by each table set.

### Evaluation Measures

For each question, we extract the answer cell from each of the top-$K$ ranked candidate chains, and thereby top-$K$ answer cells are retrieved. To evaluate the results, we adopt Precision, Recall, and $\mathrm{F}_1$ measures as similarly defined in traditional information retrieval [42], since table cell search is analogical to retrieving relevant documents to a query. We regard an answer cell as relevant to the input question if it contains at least one entity in the gold-standard answer set. Therefore, for question $q$, Precision ($P$) and Recall ($R$) are defined as $P = \frac{n_q}{K}$ and $R = \frac{a_q}{N_q}$. Here $n_q$ is the number of *relevant* answer cells retrieved, $a_q$ is the number of unique gold-standard answer entities contained in the top-$K$ answer cells, and $N_q$ is the total number of gold-standard answer entities. $F_1$ is the harmonic mean of $P$ and $R$. We present the average Precision, Recall, and $\mathrm{F}_1$ over all test questions as final results.

### Alternative Systems for Comparison

KB-based QA systems via semantic parsing can be regarded as a special kind of table cell search if we take a tabular view of knowledge bases: Each predicate corresponds to a two-column table with subject entities in one column and object entities in the other. Two state-of-the-art such systems are taken into account. Sempre and ParaSempre developed in [6] and [7] have shown excellent performance to answer questions in WebQ. We directly use their predicted results on WebQ. On BingQ, we re-train the systems on the training set and evaluate using the testing set. The implementation of the systems is publicly available in `https://github.com/percyliang/sempre`. These systems return a set of entities from Freebase as answer. For fair comparison, the returned entity set is treated as their top-1 retrieved answer cell, and evaluated according to our previously defined measures. Unless otherwise stated, we compare the top-1 answer cell returned by each system in the following experiments.

## 3.5.2 Performance of Different Feature Groups

We first discuss the performance of different feature combinations in our framework, with Wikitables as the answer source. For each feature combination, we merely use features from that combination for training and testing. As shown in Table 3.3, we have made the following discoveries:

| | WebQ | | | BingQ | | |
|---|---|---|---|---|---|---|
| Features | Precision | Recall | $F_1$ | Precision | Recall* | $F_1$ |
| Shallow Features | 0.4214 | 0.3373 | 0.3561 | 0.4035 | 0.4035 | 0.4035 |
| Deep Features | 0.5352 | 0.4210 | 0.4462 | 0.4757 | 0.4747 | 0.4750 |
| Shallow + Deep Features | **0.5712** | **0.4540** | **0.4804** | **0.5817** | **0.5817** | **0.5817** |
| Shallow + DeepType | 0.5433 | 0.4323 | 0.4566 | 0.5493 | 0.5493 | 0.5493 |
| Shallow + DeepPredicate | 0.5492 | 0.4315 | 0.4572 | 0.5493 | 0.5493 | 0.5493 |
| Shallow + DeepSentence | 0.4728 | 0.3768 | 0.3954 | 0.4227 | 0.4227 | 0.4227 |
| Shallow + DeepEntityPairs | 0.4662 | 0.3703 | 0.3907 | 0.5538 | 0.5528 | 0.5531 |
| All Features − DeepType | 0.5551 | 0.4362 | 0.4623 | 0.5538 | 0.5538 | 0.5538 |
| All Features − DeepPredicate | 0.5609 | 0.4474 | 0.4732 | 0.5714 | 0.5714 | 0.5714 |
| All Features − DeepSentence | 0.5639 | 0.4467 | 0.4729 | 0.5803 | 0.5803 | 0.5803 |
| All Features − DeepEntityPairs | 0.5698 | 0.4523 | 0.4786 | 0.5596 | 0.5596 | 0.5596 |

* Recall is close to precision, as almost every question in BingQ has one answer.

Table 3.3: Performance of different feature combinations.

(1) Shallow features, which only take advantage of word-level information, actually achieve surprisingly good performance on both evaluation sets. This may be explained by the high redundancy of web tables: A table using similar wording as the input question likely exists, which makes direct word-level matching effective on some questions. It shows a unique advantage of using web tables as answer sources, as opposed to using a rigidly defined knowledge base. Nevertheless, deep features still get much better performance than shallow features, showing that deeper inference is also necessary and is more advanced.

(2) Shallow features and deep features complement each other. Search engine snippets used in computing shallow features can gather question-related information on the Web to help match a question with the correct candidate chain. Therefore, although shallow features are less effective than deep features, com-

bining them achieves the best performance, with a 34.9% $\sim$ 44.2% relative improvement over only shallow features and 7.7% $\sim$ 22.5% over only deep features on both evaluation sets.

(3) Each deep feature defined in Section 3.3 is combined with shallow features to compare their relative advantage. The two deep features based on answer type and pseudo-predicate, i.e., DeepType and DeepPredicate, are most important in both evaluation sets. Both DeepType and DeepPredicate contain the answer type information, i.e., the column name corresponding to the answer cell. This gives us an important implication that correctly inferring the answer type of a question is critical to finding correct answers. The performance of DeepEntityPairs is quite different on these two evaluation sets. On BingQ, DeepEntityPairs is slightly better than DeepPredicate and DeepType. This is possibly because BingQ questions are not well-formed (see examples in Table 3.1), and using a significant number of entity pairs to match them can be more effective than using regular word sequences such as answer type and pseudo-predicate. Although overall DeepSentence and DeepEntityPairs are not as effective as DeepType or DeepPredicate, removing either of them from our framework can hurt the performance, as seen from the last two rows in Table 3.3.

| | WebQ | | | BingQ | | |
|---|---|---|---|---|---|---|
| System | Precision | Recall | $F_1$ | Precision | Recall | $F_1$ |
| TabCell | 0.5712 | 0.4540 | 0.4804 | 0.5817 | 0.5817 | 0.5817 |
| Sempre | 0.5419 | 0.4738 | 0.4895 | 0.3328 | 0.3328 | 0.3328 |
| ParaSempre | 0.6013 | 0.5294 | 0.5463 | 0.3711 | 0.3711 | 0.3711 |
| **TabCell +ParaSempre** | **0.7702** | **0.6765** | **0.6998** | **0.6186** | **0.6186** | **0.6186** |

Table 3.4: Comparison of different systems.

### 3.5.3 Comparison with KB-based QA Systems

We now compare our table cell search framework with two state-of-the-art KB-based QA systems Sempre and ParaSempre, which extract answers from Freebase, a large and widely used knowledge base. This comparison can help gain insights about the two answer sources (web tables *vs.* knowledge bases). Apart from separately evaluating each system, we also combine the predicted answer cell from our framework with that from ParaSempre. If our framework and ParaSempre complement each other in answering different questions, the combined results are expected to induce a large performance gain. Results are shown in Table 3.4, with TabCell referring to our framework.

Our observations lie in two aspects:

(1) System performance varies on different evaluation sets. On WebQ, ParaSempre obtains the best performance, yet TabCell is still comparable to Sempre. While on BingQ, TabCell outperforms Sempre and ParaSempre respectively by **74.8%** (from 0.3328 to 0.5817) and **56.7%** (from 0.3711 to 0.5817). These

significant differences partly result from the evaluation set construction process. Questions in WebQ were coined on Freebase and are guaranteed answerable by Freebase. However, in BingQ, questions were collected from search engine logs and the knowledge required to answer them does not necessarily exist in Freebase. Nevertheless, we can safely draw the conclusion that our framework is at least as effective as state-of-the-art KB-based QA on these evaluation sets.

(2) We concatenate the answer cells returned by TabCell and ParaSempre for each question, which brings around **28.1%** (from 0.5463 to 0.6998) and **66.7%** (from 0.3711 to 0.6186) improvements over ParaSempre on WebQ and BingQ, respectively. The simple combination approach asserts non-decreasing performance; however, such a large performance gain convincingly indicates that table cell search can complement KB-based QA. It verifies our hypothesis that tables contain rich information that might be missing or difficult to be identified in KBs, and our framework presents an effective way to precisely locate such information to satisfy user needs.

### 3.5.4 Deep Neural Networks versus Shallow Models

In Section 3.3, we employ deep neural networks to embed a question pattern and one perspective about a candidate chain such as pseudo-predicate or answer type, and measure their similarity in the semantic space. In ParaSempre [7], the

| | WebQ | | | BingQ | | |
|---|---|---|---|---|---|---|
| Features | Precision | Recall | $F_1$ | Precision | Recall | $F_1$ |
| Shallow + DeepPredicate | 0.5492 | 0.4315 | 0.4572 | 0.5493 | 0.5493 | 0.5493 |
| Shallow+ Paraphrase | 0.4339 | 0.3452 | 0.3644 | 0.4197 | 0.4197 | 0.4197 |
| Shallow Features | 0.4214 | 0.3373 | 0.3561 | 0.4035 | 0.4035 | 0.4035 |

Table 3.5: Advantages of deep neural networks.

paraphrase model was used to measure the semantic closeness between a question $x$ and a canonical utterance $c$ generated based on a candidate logic form. The paraphrase model is composed of two parts: an association model and a vector space model. The association model produces a score based on whether $x$ and $c$ contain phrases that are likely to be paraphrases. The vector space model assigns a vector representation for $x$ and $c$, and learns a scoring function to evaluate their semantic similarity. Both models are based on shallow linear combinations. Now we employ these two models in our task to score the semantic matching between a question pattern $q_p$ and the pseudo-predicate $c_p$ on a candidate chain, as an alternative to deep neural networks. The scores are treated as features (named Paraphrase here) and combined with the shallow features in our framework. As seen from Table 3.5, adding Paraphrase features only slightly improves the performance than otherwise. In contrast, with DeepPredicate extracted by deep neural networks, we can obtain a much better performance, showing the advantage of deep neural networks for semantic matching.

| Table Sources | Precision | Recall | $F_1$ |
|---|---|---|---|
| WebQ | | | |
| WikiTables | 0.5162 | 0.4103 | 0.4342 |
| AllTables | 0.4738 | 0.3708 | 0.3923 |
| BingQ | | | |
| WikiTables | 0.4981 | 0.4981 | 0.4981 |
| AllTables | 0.5233 | 0.5226 | 0.5228 |

Table 3.6: Comparison of different table sources.

### 3.5.5 Mining All Tables on the Web for QA

We not only test our framework with WikiTables as the answer source, but also with AllTables which is around 20 times larger and covers 10% more questions than the former. On the other hand, AllTables is also noisier since general web users compose tables with less attention to table schema or column naming than Wikipedia contributors. Table parsers [71] can be more error-prone when extracting tables from the general webpages. We now compare these two table sets as answer sources using the larger testing set in Table 3.2, i.e., 1507 questions in WebQ and 793 in BingQ. These testing sets are preferred over the smaller ones, because it allows the opportunity to show advantages of the larger coverage by AllTables. Table 3.6 shows the results of our framework using all the features. On WebQ, WikiTables can provide better results than AllTables, partly because of the strong connection between Freebase and Wikipedia, i.e., the knowledge stored in Freebase is heavily derived from Wikipedia [1]. Detecting answers to WebQ questions from the smaller and cleaner WikiTables shall be easier than from AllTables. On the other hand, for BingQ questions which are not necessarily answerable by Freebase or Wikipedia pages, AllTables performs better owing to its higher coverage.

# Chapter 4

# Expert-based QA

Apart from automated algorithms and systems based on various kinds of information sources including knowledge bases, texts, and tables, human collaborative networks are also abundant in real life for question answering and problem solving. Now in this chapter, we analyze *real* expert behaviors in human collaborative networks, which largely affects the network problem solving efficiency. Recall the task resolution process introduced in Section 1.3. After investigating thousands of tasks from an IBM IT service department, we recognize that in many cases, an expert might not route a task to the *best* candidates (in terms of the possibility to solve the task), especially when the task is far beyond his expertise. Instead, the task is transferred to an expert whose speciality is between the current expert and the best candidates. This routing pattern is clearly indicated by Figure 4.1. Figure 4.1 plots the histogram of expertise difference $\frac{\|E_A - E_B\|}{\|E_A\|}$, which is calculated when expert $A$ transfers a task to $B$. $E_A$ represents the expertise of $A$ and is automatically learnt based on $A$'s task resolution records. It is observed that an expert tends to transfer a task to some expert whose expertise is *neither* too similar to *nor* too different from his own. This phenomenon can be explained as follows: An expert is less likely to transfer a task to another expert whose expertise is very similar, given that the current expert already fails to resolve the task. On the other hand, if the expertise of two experts are very different, they might actually specialize in quite different domains; therefore, an expert might not be clear about the other's speciality and few tasks would be transferred between them.

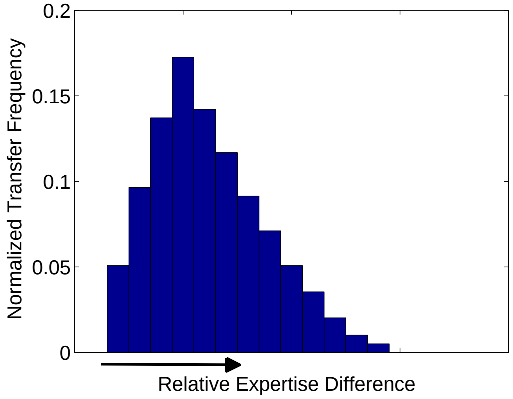


Figure 4.1: Task transfer frequency vs. expertise difference.

Inspired by the above observation, we introduce a routing pattern describing the general trend of expert $A$ transferring a task to $B$, based on the *expertise difference* between $A$ and $B$. Apart from this routing pattern, another two are also formalized. Specifically, when an expert finds there are five candidates to dispatch a task to – all of them can solve the task, who is he going to contact? A straightforward approach is to randomly pick one. An alternative is to look at the capacity of these candidates and route more tasks to an expert who can process more tasks. An expert could follow a certain pattern when deciding where to transfer a task. Different experts might adopt each routing pattern to a different degree and demonstrate different routing behavioral characteristics. Our study is going to infer the routing patterns as well as experts' preferences over them, from the historical routing data, and finally give insightful analysis of expert performance in a collaborative network to improve the problem solving efficiency.

## 4.1 Preliminaries

We first clarify the notations used in this chapter. $\mathcal{E} = \{e_1, e_2, ..., e_i, ..., e_G\}$ is a set of experts in a collaborative network. $\mathcal{N}_i$ denotes the 1-hop neighborhood of expert $e_i$ in the network. $\mathcal{W} = \{w_1, w_2, ..., w_n, ..., w_N\}$ is a set of words used to describe the tasks. $\mathcal{T} = \{t_1, t_2, .., t_m, ..., t_M\}$ is a set of tasks resolved by the collaborative network, where each $t_m$ is an $N \times 1$ word vector with each dimension recording the word frequency in the task description. Apart from the textual description, each task is also associated with a routing sequence starting from an initial expert to the resolver of the task.

Table 4.1 shows one example problem ticket in an IT service department. The ticket with ID 599 is a problem related to *operating system*, specifically, *the low*

| ID | Entry | Time | Expert |
|---|---|---|---|
| 599 | New ticket: the available space on the /var file system is low | 9/14/06 5:57:16 | IN039 |
| 599 | ...(operations by IN039)... | ... | IN039 |
| 599 | Ticket 599 transferred to SAV59 | ... | IN039 |
| 599 | ...(operations by SAV59)... | ... | SAV59 |
| 599 | Ticket 599 transferred to SAV4F | ... | SAV59 |
| 599 | ...(operations by SAV4F)... | ... | SAV4F |
| 599 | Problem resolved: free up disk space in the file system | 9/14/06 9:57:31 | SAV4F |

Table 4.1: The lifetime of an example task.

*percentage of the available file system space*. It was assigned to or initiated by expert IN039, then routed through SAV59, and finally resolved by expert SAV4F.

In this chapter, we study the following problems: *How does an expert in a collaborative network make a routing decision? Is there any pattern an expert generally follows when routing a task?* Different from previous studies [44, 58, 76], we do not propose algorithms to perform more efficient routing. Instead, we hope to understand the routing decisions *actually* made by the real experts.

Our intuition is that an expert makes a routing decision by adopting a certain routing strategy either consciously or unconsciously. For example, an expert might decide where to route a task by evaluating the next expert's possibility to solve the task. We advocate a general and extensible methodology to analyze the routing decision making process, which consists of two routing strategies: *Task-Neutral Routing* (TNR) and *Task-Specific Routing* (TSR). Under the Task-

Neural Routing, an expert does not take into account the specificity of a task when making a routing decision, and treat different tasks equivalently. Under the Task-Specific Routing, however, an expert makes a routing decision by analyzing the specific task being transferred. We attempt to deduce the cognitive process of an expert during task routing, and model the decision making of an expert as a generative process where a routing decision is generated based on the two routing strategies. Each routing strategy is respectively combined with three basic routing patterns, which overall produces six particular routing patterns. Our generative model, with these six routing patterns as mixture components, is then proposed to describe the process of an expert's decision making.

## 4.2 Modeling Expert Routing Behaviors

### 4.2.1 Routing Patterns

Given a task, we assume an expert $e_i$ first establishes a pool of candidates, $C$, to dispatch the task to. The difference between the Task-Neutral Routing (TNR) and the Task-Specific Routing (TSR) lies in the composition of $C$. In terms of TNR, $C$ contains all of his neighbors $\mathcal{N}_i$. In terms of TSR, $C$ is limited to experts who are able to solve the current task in $\mathcal{N}_i$. TNR accommodates the situations when an expert does not understand a task very well, or an expert has a careless work attitude, and thereby making a routing decision irrespective of the specific task and its possible resolvers; whereas TSR mimics the situation where an expert assesses others' ability to solve a task before dispatching it.

Once $C$ is established, $e_i$ selects one expert from $C$ to route the task to, based on a certain routing pattern. We identified three basic routing patterns:

*Uniform Random* (UR). This routing pattern implies that an expert makes a decision by randomly selecting one of the candidates in $C$ with an equal probability.

$$P(e_i \xrightarrow{t} e_j \mid \mathrm{UR}) = \mathbb{1}(e_j \in C)\frac{1}{|C|}, \tag{4.1}$$

where $e_i \xrightarrow{t} e_j$ denotes the event that expert $e_i$ transfers task $t$ to $e_j$. $\mathbb{1}$ is an indicator function: $\mathbb{1}(e_j \in C)$ picks value 1 if $e_j \in C$ holds otherwise 0.

*Volume-biased Random* (VR). Under this routing pattern, an expert also randomly dispatches tasks, but with a rate proportional to the volume of tasks previously dispatched. VR mimics the situation where the task processing capacity could vary for different experts. VR can be regarded as an expert's reaction when he finds tasks are processed slowly by some of his collaborators. Let volume $v_{ij}$ denote the number of transferred tasks from expert $e_i$ to $e_j$.

$$P(e_i \xrightarrow{t} e_j \mid \mathrm{VR}) = \mathbb{1}(e_j \in C)\frac{v_{ij}}{\sum_{e_k \in C} v_{ik}}. \tag{4.2}$$

*Expertise Difference* (EX). In addition to the above two routing patterns, this one is derived from the observation shown in Figure 4.1: An expert is more likely to send a task to another expert whose expertise is neither too close nor too far from his own. Let $f_{ij}$ denote the general trend of expert $e_i$ sending a task to $e_j$, given their expertise.

$$P(e_i \xrightarrow{t} e_j \mid \mathrm{EX}) = \mathbb{1}(e_j \in C)\frac{f_{ij}}{\sum_{e_k \in C} f_{ik}}. \tag{4.3}$$

To estimate $f_{ij}$, we build a model based on the observation in Figure 4.1. The relative expertise difference between expert $e_i$ and $e_j$, is calculated as $\Delta(e_i, e_j) = \frac{\|e_i - e_j\|}{\|e_i\|}$, where for simplicity $e_i$ is reused to represent the expertise vector of expert $e_i$. The expertise estimation problem will be discussed later in this section. Based on Figure 4.1, we assume for those tasks transferred, the relative expertise difference between a task sender and a task receiver follows a log-normal distribution with parameters $\mu$ and $\sigma^2$. We made this assumption due to the non-negative nature of $\Delta(e_i, e_j)$ and the asymmetric shape of the distribution. However, in our experiments, we also test a model with a normal distribution to estimate $f_{ij}$ and show that the log-normal distribution works better. Under the log-normal distribution, expert $e_i$ is more likely to transfer tasks to $e_j$ once $\Delta(e_i, e_j)$ obtains a higher probability density; therefore, $f_{ij}$ is estimated by:

$$f_{ij} \propto \frac{1}{\Delta(e_i, e_j)} e^{-\frac{[\ln \Delta(e_i, e_j) - \mu]^2}{2\sigma^2}}. \tag{4.4}$$

Note that the histogram in Figure 4.1 is the accumulation of all possible routing patterns, not only the *expertise difference* pattern. Later, in the experiments, we will show that the EX pattern with $\mu$ and $\sigma^2$ directly estimated from Figure 4.1

does not perform the best. Instead, parameters $\mu$ and $\sigma^2$ shall be estimated with more emphasis on tasks transferred following the EX pattern, which are not known a priori. We will learn them simultaneously through a mixture model with all the routing patterns considered.

To summarize, the three basic patterns {UR, VR, EX} combined with the Task-Neutral Routing and Task-Specific Routing strategy, generate six particular routing patterns: TNR$^{ur}$, TNR$^{vr}$, TNR$^{ex}$, TSR$^{ur}$, TSR$^{vr}$, and TSR$^{ex}$.

### 4.2.2 Task-Specific Routing

TNR does not take into account the distinctiveness of a specific task. In contrast, TSR means that an expert makes a routing decision based on the specific task and matches it with potential resolvers. Given expert $e_i$ to transfer task $t$, his routing decision under TSR is influenced by two factors: (1) whether an expert can solve the task or not, and (2) how familiar $e_i$ is with that expert. The first factor, only related to the task itself, can be conducted by a classification process without involving $e_i$. The classification identifies a subset of $e_i$'s neighbors who can solve the task, as the candidate pool $C$. We build the classifier based on the task resolution records of experts. The second factor is a human factor which can be modeled by the same routing patterns we previously introduced such as UR, VR, and EX. Overall, TSR will check the neighborhood of $e_i$, run the classifier, and establish a set of candidates $C$ who are capable of solving $t$. It then selects one particular candidate from $C$ based on one of {UR, VR, EX}. In a special case where $C = \emptyset$, TSR is reduced to TNR where we simply use the entire neighborhood $\mathcal{N}_i$ as the candidate pool.

The success of our work is related to the recognition of human factors in task routing. A straightforward routing pattern considering the specificity of a task is to transfer the task to an expert with the highest probability to solve it. Our experiments show that such a routing pattern cannot capture the real characteristics of human decision making, such as randomness, uncertainty, and sub-optimality. We observe that similar tasks are often routed to very different experts. Moreover, one usually does not search for people who are most likely to solve a problem due to, e.g., unfamiliarity with those people. Instead, he might select a close collaborator who should be able to solve the problem, but not necessarily with the highest probability.

### 4.2.3 Expertise Estimation

In TSR strategy and the EX routing pattern, we need to estimate an expert's expertise and capability to solve a task. Intuitively, the capability depends on both the expert's expertise and the task description. We resort to a classic logistic regression model [8] that takes an expert's expertise vector and a task's word vector as input, and outputs the expert's capability to solve the task. In the logistic model, the probability for expert $e_i$ to solve task $t$, denoted as $P(e_i, t)$, is defined as follows:

$$P(e_i, t) = \frac{1}{1 + \exp(-(W_1 t + W_2 e_i + b))}. \tag{4.5}$$

For simplicity, the expertise vector for expert $e_i$ is of the same length as task $t$, i.e., an $N \times 1$ vector. $W_1$ and $W_2$ are the $1 \times N$ weights respectively associated with the word vector of a task and the expertise vector of an expert. Each component of $W_1$ and $W_2$ denotes the contribution of the corresponding dimension in $t$ or $e_i$ to the capability prediction. $b$ is a bias scalar in the logistic model. The expertise vectors $e_i$'s are not known a priori and to be estimated with the model parameters $\{W_1, W_2, b\}$. We use $W = \{W_1, W_2, b, e_i's\}$ to denote all the parameters.

Given a task $t$ and its routing sequence, e.g., $e_i \rightarrow ... \rightarrow e_k$, we observe the groundtruth regarding the resolution capability: expert $e_k$ solves $t$ and any other expert on the sequence does not solve it. Therefore, we can formulate a training dataset composed of <expert, task> pairs as instances and $\{0, 1\}$ as the observed probability of an expert to solve a task, e.g., 0 for <$e_i$, $t$> while 1 for <$e_k$, $t$>.

Parameters in the model are obtained by minimizing the cross-entropy error function [8], based on the <expert, task> pairs in the training dataset:

$$\arg\min_{W} \sum_{<e_i,t>} [-P^*(e_i, t) \log P(e_i, t) - (1 - P^*(e_i, t)) \log(1 - P(e_i, t))], \tag{4.6}$$

where $P^*(e_i, t)$ is the observed probability for expert $e_i$ to solve task $t$. After the parameters are learnt, given a task and an expert, one can predict the probability for the expert to solve the task using Eqn. 4.5. Under Task-Specific Routing, when transferring task $t$, expert $e_i$ identifies a subset of his neighbors $\mathcal{N}_i$ as the candidate pool $C = \{e_j \in \mathcal{N}_i : P(e_j, t) \geq \delta\}$, where $\delta$ is set at 0.5 in our implementation. Experts in $C$ have a probability to solve $t$ larger than a threshold, and are estimated capable to solve the task.

## Discussion

(1) In our scenario, the expertise vector of an expert is not known a priori and needs to be learnt. One might consider using the average word vector of the tasks *resolved* by an expert to represent the expert's knowledge. However, in practice, this method can be problematic, because we observe that there might be many experts in a network that serve as "intermediate transferrers" and did not resolve any tasks. Therefore, we also utilize those tasks *unresolved* by an expert to estimate his expertise as shown in Eqn. 4.6.

(2) In general, human expertise can be characterized in two aspects: *specialization* (the topics/areas an expert is good at) and *proficiency level* (how proficient an expert is in his area). For specialized collaborative networks, i.e., those designated to resolve tasks in a certain domain such as Java programming contests, the

specialization of all the experts can be regarded as the same (i.e., the designated domain), but their task solving abilities, or proficiency levels, differ. Our model described in this section only captures the proficiency level of experts, in that it focuses on predicting whether the proficiency level of an expert is high enough to solve a given task. The ranking of experts based on proficiency level is fixed and does not vary across different tasks. For collaborative networks dealing with more broad-range topics, however, more advanced expertise modeling that is also able to distinguish the specialization of experts is needed (e.g., one expert may excel at civil laws while another expert is adept at criminal laws). We refer interested readers to our recent work [29], where specialization and proficiency level are jointly modeled. Expertise modeling, or more broadly representation learning, in human networks is a new yet important problem that calls for further study.

Here the expert proficiency/capability estimation under a given task is casted as a traditional classification problem. Different classification models could be employed potentially. Our adopted model can be perceived as the most basic neural network model, i.e., perceptron [9]. We also built multi-layer neural networks, where non-linear higher-level features for a given expert $e_i$ and task $t$ can be extracted and further used for classification. For the specific datasets tested in this chapter, the multi-layer neural networks only outperform the perceptron model by around 3%, by promoting the classification accuracy from roughly 80%

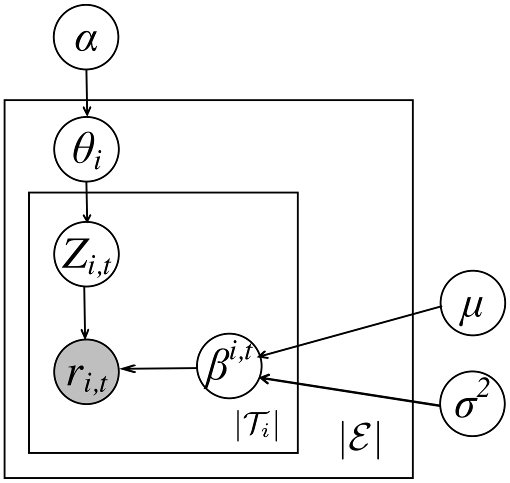


Figure 4.2: Graphical representation of our model.

to 83%. Therefore, for simplicity, we adopt the perceptron model, as it is much more efficient when incorporated with our task routing patterns.

## 4.3 Generative Model

In this section, we present a generative model to put the previously discussed routing patterns together and describe an integrated decision making process.

Figure 4.2 shows the graphical representation of our generative model. We first clarify the notations in the figure as follows: (1) $|\mathcal{E}|$ denotes the number of experts while $|\mathcal{T}_i|$ is the number of tasks expert $e_i \in \mathcal{E}$ has ever transferred. A plate means replicating a process for multiple times. (2) $\theta_i$ is the $K \times 1$ mixture weights of different routing patterns for expert $e_i$, where $K$ is the number of routing patterns. In our current setting, we have $K = 6$ routing patterns, from TNR$^{ur}$ to TSR$^{ex}$. The $k$-th component of $\theta_i$ reveals the probability that the $k$-th routing pattern is adopted by $e_i$ to transfer a task. (3) $\alpha$, a $K \times 1$ vector, is parameters in a Dirichlet prior, and serves as a constraint of all the mixture weights $\theta_i$'s. A Dirichlet prior for the mixture weights tends to alleviate over-fitting problems [10]. Besides, with the Dirichlet prior, the mixture weight $\theta_{new}$ for a new expert can be naturally assigned. (4) $Z_{i,t}$ is the label of the routing pattern employed by expert $e_i$ when transferring task $t$. (5) $\beta^{i,t}$, a $K \times |\mathcal{N}_i|$ matrix, defines the probability distribution of expert $e_i$ transferring task $t$ to an expert in his neighborhood $\mathcal{N}_i$, under $K$ routing patterns. Particularly, each row of $\beta^{i,t}$ is filled by a probability distribution under one of the six routing patterns, as defined in Section 4.2.1. For experts in $\mathcal{N}_i$ but not in the candidate pool $C$, the corresponding elements in $\beta^{i,t}$ are naturally filled with 0. For patterns irrelevant to EX, we pre-compute their probability distributions and fill corresponding rows of $\beta^{i,t}$, while TNR$^{ex}$ and TSR$^{ex}$ are parameterized with $\mu$ and $\sigma^2$. Note that $\beta^{i,t}$ is in the inner plate of the graphical model because $\beta^{i,t}$ is associated with expert $e_i$ and task $t$. (6)The shaded variable $r_{i,t}$ indicates the observed receiver of task $t$ transferred from expert $e_i$.

Figure 4.2 conveys that expert $e_i$ decides where to route task $t$ based on multiple routing patterns $\beta^{i,t}$ and his preference $\theta_i$ towards adopting different routing patterns. Now we formally describe the generative process as follows:

For each expert $e_i$ to transfer tasks,

- Draw the mixture weights of $K$ routing patterns: $\theta_i \sim \mathbf{Dir}(\alpha)$.
- For each task $t$ to be transferred by expert $e_i$,
  - Draw a pattern label: $Z_{i,t} \sim \mathbf{Mult}(\theta_i)$.
  - Draw an expert from $\mathcal{N}_i$ to receive $t$:
    $r_{i,t} \sim P(e_i \xrightarrow{t} e_j | Z_{i,t}, \beta^{i,t})$, $\forall e_j \in \mathcal{N}_i$.

For each task $t \in \mathcal{T}_i$, the transfer relationship for $t$ is represented by $e_i \xrightarrow{t} r_{i,t}$. We formulate the likelihood of observing all the task transfer relationships as follows:

$$\mathcal{L} = P(e_i \xrightarrow{t} r_{i,t}, \forall t \in \mathcal{T}_i, \forall e_i \in \mathcal{E} | \alpha, \mu, \sigma^2). \tag{4.7}$$

Since a routing decision of an expert is independent from that of another expert while the routing decisions of the same expert for different tasks are not independent from each other, we can rewrite $\mathcal{L}$ in the following way:

$$\begin{aligned}\mathcal{L} &= \prod_{e_i \in \mathcal{E}} P(e_i \xrightarrow{t} r_{i,t}, \forall t \in \mathcal{T}_i | \alpha, \mu, \sigma^2) \\ &= \prod_{e_i \in \mathcal{E}} \int_{\theta_i} P(\theta_i|\alpha) P(e_i \xrightarrow{t} r_{i,t}\ , \ \forall t \in \mathcal{T}_i | \theta_i, \mu, \sigma^2)\, d\theta_i,\end{aligned} \tag{4.8}$$

where,

$$P(e_i \xrightarrow{t} r_{i,t} \ , \ \forall t \in \mathcal{T}_i \mid \theta_i, \mu, \sigma^2) = \prod_{t \in \mathcal{T}_i} \{ \sum_{Z_{i,t}} P(Z_{i,t}|\theta_i) P(e_i \xrightarrow{t} r_{i,t} \mid Z_{i,t}, \beta^{i,t}) \}, \tag{4.9}$$

where $P(Z_{i,t}|\theta_i) = \theta_{i,k}$ and $P(e_i \xrightarrow{t} r_{i,t} \mid Z_{i,t}, \beta^{i,t}) = \beta^{i,t}_{k,r}$, if $Z_{i,t} = k$, i.e., the $k$-th routing pattern is adopted. $\beta^{i,t}_{k,r}$ is the probability for $e_i$ routing task $t$ to expert $r_{i,t}$, under the $k$-th pattern. $\beta^{i,t}_{k,r}$ shall contain parameters $\mu$ and $\sigma^2$ if the $k$-th pattern is TNR$^{ex}$ or TSR$^{ex}$.

Finally, we resort to the maximum likelihood estimation approach to optimize the parameters in the model:

$$\arg\max_{\alpha,\mu,\sigma^2} \log \mathcal{L}. \tag{4.10}$$

Now we discuss how to estimate the model parameters in detail. The latent variables $\{\theta_i$'s, $Z_{i,t}$'s$\}$ are not independent of each other, which makes their true posterior distributions computationally intractable. In this section, we employ a variational approach [8] to solve our model.

### 4.3.1 Variational Inference

We introduce a variational distribution $Q$ in which the latent variables are independent of each other to approximate their true posterior distribution, i.e., $Q(\theta, Z) = Q(\theta)Q(Z)$, where $\theta = \{\theta_i, \forall e_i \in \mathcal{E}\}$ and $Z = \{Z_{i,t}, \forall e_i \in \mathcal{E}, t \in \mathcal{T}_i\}$. According to the variational distribution, $Q(\theta_i) \sim \mathbf{Dir}(\gamma_i)$, $Q(Z_{i,t}) \sim \mathbf{Mult}(\phi^{i,t})$, , where $\gamma_i$ and $\phi^{i,t}$ are $K \times 1$ variational parameters. $\gamma_i$ and $\phi^{i,t}$ have significant meanings where $\gamma_i$ represents the variational prior for $\theta_i$ and reflects which routing pattern $e_i$ tends to adopt, while $\phi^{i,t}$ is the variational posterior mixture weights of different routing patterns adopted by $e_i$ when transferring task $t$. Given the observed data, both $\gamma_i$ and $\phi^{i,t}$ will be derived automatically.

Under the variational distribution and Jensen's inequality, we can maximize the lower bound of the log likelihood, instead of directly maximizing $\mathcal{L}$.

$$\log \mathcal{L} \geq E_Q \text{log} P(\mathcal{D}, \theta, Z | \alpha, \mu, \sigma^2) + H(Q) = \lfloor \mathcal{L} \rfloor, \tag{4.11}$$

where $\mathcal{D}$ denotes all the observed task transfer relationships. We expand the lower bound of the log likelihood as follows:

$$\begin{aligned} \lfloor \mathcal{L} \rfloor = & \sum_{e_i \in \mathcal{E}} E_Q \log P(\theta_i | \alpha) + \sum_{e_i \in \mathcal{E}} \sum_{t \in \mathcal{T}_i} E_Q \log P(Z_{i,t} | \theta_i) \\ & + \sum_{e_i \in \mathcal{E}} \sum_{t \in \mathcal{T}_i} E_Q \log P(e_i \xrightarrow{t} r_{i,t} | Z_{i,t}, \mu, \sigma^2) \\ & + H(Q(\theta, Z)). \end{aligned} \tag{4.12}$$

Each term on the right-hand side of the above equation, is a function over the model parameters as shown in Eqn. 4.13 to Eqn. 4.16.

$$\sum_{e_i \in \mathcal{E}} E_Q \log P(\theta_i | \alpha) = -|\mathcal{E}| \log B(\alpha) + \sum_{e_i \in \mathcal{E}} \sum_k (\alpha_k - 1)[\psi(\gamma_{i,k}) - \psi(\sum_k \gamma_{i,k})], \tag{4.13}$$

where $B(\alpha) = \frac{\prod_k \Gamma(\alpha_k)}{\Gamma(\sum_k \alpha_k)}$ is the normalization constant of the Dirichlet distribution $\mathbf{Dir}(\alpha)$.

$$\sum_{e_i \in \mathcal{E}} \sum_{t \in \mathcal{T}_i} E_Q \log P(Z_{i,t} | \theta_i) = \sum_{e_i \in \mathcal{E}} \sum_{t \in \mathcal{T}_i} \sum_k \phi_k^{i,t} [\psi(\gamma_{i,k}) - \psi(\sum_k \gamma_{i,k})]. \tag{4.14}$$

The third term

$$\sum_{e_i \in \mathcal{E}} \sum_{t \in \mathcal{T}_i} E_Q \log P(e_i \xrightarrow{t} r_{i,t} | Z_{i,t}, \mu, \sigma^2) = \sum_{e_i \in \mathcal{E}} \sum_{t \in \mathcal{T}_i} \sum_k \phi_k^{i,t} \log \beta_{k,r}^{i,t}. \tag{4.15}$$

As discussed in Eqn. 4.9, $\beta_{k,r}^{i,t}$ contains parameters $\mu$ and $\sigma^2$ if the $k$-th pattern is TNR$^{ex}$ or TSR$^{ex}$.

The entropy term

$$\begin{aligned} H(Q(\theta, Z)) &= -\sum_{e_i \in \mathcal{E}} [E_Q \log Q(\theta_i|\gamma_i) + \sum_{t \in \mathcal{T}_i} E_Q \log Q(Z^{i,t}|\phi^{i,t})] \\ &= \sum_{e_i \in \mathcal{E}} [\log B(\gamma_i) - \sum_k (\gamma_{i,k} - 1)(\psi(\gamma_{i,k}) - \psi(\sum_k \gamma_{i,k}))] \\ &- \sum_{e_i \in \mathcal{E}} \sum_{t \in \mathcal{T}_i} \sum_k \phi_k^{i,t} \log \phi_k^{i,t}. \end{aligned} \tag{4.16}$$

### 4.3.2 Parameter Estimation

The model parameters are estimated by using the variational expectation-maximization (EM) algorithm. In the E-step, we update the variational parameters $\{\gamma\text{'s}, \phi\text{'s}\}$ while in the M-step, we update the model parameters $\alpha$, $\mu$, and $\sigma^2$ so that $\lfloor \mathcal{L} \rfloor$ is maximized.

Specifically, the E-step updates the variational parameters according to Eqn. 4.17 and 4.18.

$$\phi_k^{i,t} \sim \beta_{k,r}^{i,t} \exp(\psi(\gamma_{i,k}) - \psi(\sum_k \gamma_{i,k}) - 1), \tag{4.17}$$

$$\gamma_{i,k} = \alpha_k + \sum_{t \in \mathcal{T}_i} \phi_k^{i,t}. \tag{4.18}$$

During the M-step, we maximize the lower bound over the parameter $\alpha$, $\mu$, and $\sigma^2$, by utilizing the classic L-BFGS optimization algorithm [41]. The derivatives over the parameter $\alpha$ are calculated in Eqn. 4.19.

$$\frac{\partial \mathcal{L}}{\partial \alpha_k} = |\mathcal{E}|[-\psi(\alpha_k) + \psi(\sum_k \alpha_k)] + \sum_{e_i \in \mathcal{E}} [\psi(\gamma_{i,k}) - \psi(\sum_k \gamma_{i,k})]. \tag{4.19}$$

Derivatives over $\mu$ and $\sigma^2$ depend on the routing pattern TNR$^{ex}$ and TSR$^{ex}$, as well as the mixture weights corresponding to the two patterns.

$$\frac{\partial \mathcal{L}}{\partial \mu} = \sum_{e_i \in \mathcal{E}} \sum_{t \in \mathcal{T}_i} \sum_{k=1}^{2} \mathbb{1}(r_{i,t} \in C_k) \times \frac{\phi_k^{i,t}}{\beta_{k,r}^{i,t}} \times \frac{f_{ir} \sum_{e_j \in C_k} X_{ij}}{\sigma^2 (\sum_{e_j \in C_k} f_{ij})^2}, \tag{4.20}$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = \sum_{e_i \in \mathcal{E}} \sum_{t \in \mathcal{T}_i} \sum_{k=1}^{2} \mathbb{1}(r_{i,t} \in C_k) \times \frac{\phi_k^{i,t}}{\beta_{k,r}^{i,t}} \times \frac{f_{ir} \sum_{e_j \in C_k} Y_{ij}}{2(\sigma^2 \sum_{e_j \in C_k} f_{ij})^2}, \tag{4.21}$$

where we assume TNR$^{ex}$ and TSR$^{ex}$ are the 1st and 2nd mixture component respectively. $C_k$ is the candidate pool established under TNR or TSR by $e_i$ when routing task $t$. $f_{ir}$ is the general trend of $e_i$ sending a task to $r_{i,t}$, based on $\Delta(e_i, r_{i,t})$. $X_{ij} \doteq f_{ij}(\ln \Delta(e_i, r_{i,t}) - \ln \Delta(e_i, e_j))$ and $Y_{ij} \doteq f_{ij}[(\ln \Delta(e_i, r_{i,t}) - \mu)^2 - (\ln \Delta(e_i, e_j) - \mu)^2]$.

The E-step and M-step are performed iteratively until the algorithm converges, which indicates that the current model parameters fit the observed training data.

## 4.4 Experiments

Now we validate the expertise difference routing pattern and evaluate the accuracy of our method in modeling expert behaviors on various real-life datasets. We will further demonstrate that with the help of our model, better recommendations on expert training could be automatically obtained and provided to managers for improving the performance of collaborative networks.

### 4.4.1 Datasets and Evaluation Measures

**Datasets**

We use real-world problem ticket data collected from a problem ticketing system in an IBM IT service department throughout 2006. Three datasets in different problem categories are explored: DB2, WebSphere, and AIX. DB2 contains problem tickets on database usage and management; WebSphere is a set of problem tickets on the enterprise software IBM WebSphere [3]; and AIX is the category of problem tickets on operating systems.

The details of the three datasets, i.e., the number of tasks, experts, and the distribution of completion time (CT), are shown in Table 4.2. The three datasets involve approximately 50 to 400 experts. Understanding how an expert makes a certain routing decision among many candidates is a meaningful yet potentially challenging problem. As evident in these datasets, the completion time for different tasks possesses a large diversity, which drives us to analyze expert routing behaviors that possibly lead to such diversity. For each dataset, we randomly partition it into two disjoint subsets: 75% of tasks for training and 25% for testing.

| Datasets | # of tasks | # of experts | % of tasks with CT | | | |
|---|---|---|---|---|---|---|
| | | | $= 2$ | $= 3$ | $= 4$ | $\geq 5$ |
| DB2 | 26,740 | 55 | 44.2 | 34.3 | 16.5 | 5.0 |
| WebSphere | 65,786 | 234 | 39.0 | 36.2 | 20.0 | 4.8 |
| AIX | 120,780 | 404 | 40.0 | 39.4 | 14.2 | 6.4 |

Table 4.2: Three datasets on ticket resolution.

### Evaluation Measures

*Routing Sequence Likelihood.* We compute the log likelihood (LL) of the routing relationships in the held-out testing dataset, according to Eqn 4.8. The higher the log likelihood, the better a model explains the routing decisions of experts.

*Predicted Task Completion Time.* In a real collaborative network, a task is routed and completed as long as it reaches an expert who can solve it. The completion time (CT) of a task is defined as the number of experts in its routing sequence. Estimation of the completion time *before actually routing a task* is critically useful, as it can raise attention for those troublesome tasks and ask the network allocate more resources to handle such tasks. The estimated completion time can also be used to evaluate routing models. A good routing model shall reflect the real decision making process and give the estimation as accurate as possible.

Experts that can resolve a task are not unique and are not known before the task is actually routed. For a new task, one cannot estimate its completion time by targeting a unique "*resolver*". Instead, we need to consider multiple potential resolvers and multiple routing sequences. Given a task and its initial expert, our generative model can generate a routing sequence of experts to process the task. Specifically, given a task $t$ and its current holder, $e_i$, the receiver $r_{i,t}$ can be sampled according to our generative process described in Section 4.3. Once $r_{i,t}$ is obtained, it is treated as the current holder of $t$; the same procedure is repeated to produce the next receiver, until we have $L$ experts to process $t$ in sequence. Although the initial expert to deal with a task is important, in our work, we do not particularly deal with the assignment of an initial expert to a certain task. We assume that the initial expert to a task is given beforehand: it is either decided by the task requestor (e.g., a customer) or by the system.

Task $t$ will stop routing once an expert can solve it. Since each expert in the routing sequence has a probability to solve the task, the completion time can be estimated ($\widehat{\mathrm{CT}}_t$) as the expected number of experts having accessed the task when it is solved, named *predicted completion time.*

$$\widehat{\mathrm{CT}}_t = \sum_{m=1}^{L} m \prod_{n=1}^{m-1} [1 - P(r_n, t)] P(r_m, t), \tag{4.22}$$

where $r_m$ ($r_n$) is the $m$-th ($n$-th) expert in a routing sequence. $\prod_{n=1}^{m-1}[1-P(r_n,t)]P(r_m,t)$ gives the probability for the $m$-th expert in the sequence to solve the task while the previous $m-1$ experts fail to, where $P(r_m,t)$'s are estimated using Eqn. 4.5. Since the probability diminishes quite quickly, we set $L = 10$ in practice. Indeed, $\widehat{\mathrm{CT}}_t$ does not vary much when $L$ is beyond 10.

The routing decision of an expert significantly affects the completion time of a task. Our model is considered valid if a task routed according to our generative process can achieve a similar completion time as it does in real situations. Two measures are employed to calculate the difference between the predicted, $\widehat{\mathrm{CT}}$ and the real completion time, CT.

a. Mean Absolute Error (MAE).

$$\mathrm{MAE} = \frac{1}{|\text{Test Set}|} \sum_{t \in \text{Test Set}} |\widehat{\mathrm{CT}}_t - \mathrm{CT}_t|. \tag{4.23}$$

b. Step Loss Measure (SL). Instead of directly computing the difference between $\widehat{\mathrm{CT}}_t$ and $\mathrm{CT}_t$ as errors, step loss measure [47] incorporates some tolerance of the difference. If the difference is larger than the tolerance, one estimation mistake is made. We set the tolerance in our case as 1. That is, if the difference between $\widehat{\mathrm{CT}}_t$ and $\mathrm{CT}_t$ is within 1, the estimation is regarded as correct; otherwise, it is regarded as wrong. We calculate the percentage of the wrong estimations in the testing data set. The lower, the better.

$$\mathrm{SL} = \frac{1}{|\text{Test Set}|} \sum_{t \in \text{Test Set}} \mathbb{1}(|\widehat{\mathrm{CT}}_t - \mathrm{CT}_t| > 1), \tag{4.24}$$

where $\mathbb{1}$ is an indicator function: it picks value 1 if the condition $|\widehat{\mathrm{CT}}_t - \mathrm{CT}_t| > 1$ holds otherwise 0.

We compare our model with the following algorithms:

(1) Regression: For each task, to estimate its completion time, one can resort to a regression algorithm to make the prediction. We use two classic methods: Support Vector Regression (SVR) [18] and Bayesian regression method [46]. Given a task, two types of features are input to each method: (i) word frequency vector in the description of a task; (ii) the initial expert assigned to the task. 10-fold cross validation is conducted for both methods. We evaluate SVR with different kernels including a linear kernel, a polynomial kernel, an RBF kernel and a wavelet kernel. For Bayesian regression, we consider Bayesian linear regression and Bayesian logistic regression. Classification using SVM [18] or naive Bayes classifier [46] are also tested, which turns out to be worse than the regression methods. Among all the variants of SVR or Bayesian regression, we always show their best results obtained. The classification/regression approaches are employed as straightforward methods for completion time estimation. They do not attempt to understand the decision making process of experts, and their results on the sequence likelihood measure are not available.

(2) Generative models. Miao et al. [44] estimate the probability of an expert to solve a task and the probability of transferring a task from an expert to another. Given a task, [44] recommends a sequence of experts to route the task. Their goal is to shorten the routing as much as possible, while our goal is to characterize human routing patterns in the real network.

### 4.4.2 Model Accuracy

Table 4.3 summarizes the performance of all the methods on step loss measure, MAE, and log likelihood, where step Loss is shown in percentage and LL is $\times 10^4$. From the results of SVR and Bayesian regression, we can see that the completion time of a task cannot be accurately predicted based on the straightforward regression methods. In fact, we observe that in the real datasets, similar tasks, even if assigned initially to the same expert, are often routed to different experts and resolved with a different completion time. This implies the resolution of a task is a complicated process and involves human factors. The estimated completion time in [44] is usually shorter than the real one as its goal is to shorten routing sequences. It is not surprising that it incurs a large step loss and MAE.

Now we test multiple variants of our generative model. For each variant, we select combinations of different routing patterns to train a generative model, and test the learned model under the three measures.

We first examine the performance of TNR-related and TSR-related routing patterns separately. Then they are combined together as TNR+TSR. Table 4.3 clearly shows both TNR and TSR play a critical role to reduce SL and MAE, indicating both types of strategies are adopted by experts in real cases. Our model does capture the decision making process of experts in a collaborative network. Our method significantly outperforms the content-only classification methods by

| | DB2 | | | WebSphere | | | AIX | | |
|---|---|---|---|---|---|---|---|---|---|
| Models | SL | MAE | LL | SL | MAE | LL | SL | MAE | LL |
| TNR | 4.11 | 0.30 | -0.28 | 4.77 | 0.40 | -0.88 | 4.46 | 0.37 | -0.41 |
| TSR | 4.56 | 0.29 | -0.25 | 4.56 | 0.37 | -0.80 | 4.15 | 0.30 | -0.35 |
| **TNR+TSR** | **1.77** | **0.08** | **-0.07** | **1.44** | **0.07** | **-0.19** | **1.99** | **0.15** | **-0.17** |
| TNR+TSR−EX | 3.05 | 0.14 | -0.10 | 3.86 | 0.25 | -0.25 | 3.86 | 0.25 | -0.25 |
| Miao *et al.* [44] | 9.89 | 0.68 | -0.61 | 11.10 | 0.81 | -1.21 | 11.10 | 0.81 | -1.21 |
| SVR | 14.78 | 0.80 | N/A | 18.20 | 0.71 | N/A | 15.08 | 0.77 | N/A |
| Bayesian regression | 13.77 | 0.84 | N/A | 17.02 | 0.80 | N/A | 12.56 | 0.85 | N/A |

Table 4.3: Effectiveness of routing models.

**75%**. Moreover, the MAE between our estimated completion time and the real one is between **0.07** and **0.15**, which shows that our method can be used to accurately predict the task completion time.

We then experiment if the expertise difference (EX) routing pattern makes sense. Specifically, we test the model that combines all the routing patterns except $\text{TNR}^{ex}$ and $\text{TSR}^{ex}$, denoted as TNR+TSR−EX. The results indicates that with the EX routing pattern considered, TNR+TSR will better capture the real decision making process. This result can be attributed to our observation: an expert is more likely to transfer a task to some expert whose expertise is neither too similar nor too different.

### 4.4.3 Resolution Efficiency

One natural hypothesis is that under the task-specific routing, a task will be resolved quickly, since TSR directly takes into account the next expert's ability to

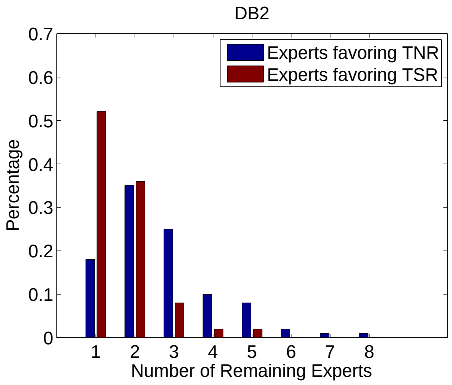


Figure 4.3: Efficiency of TNR vs. TSR.

solve the task. We now verify this hypothesis. Recall that the variational parameter $\gamma_i$ in our model TNR+TSR reflects the mixture weights used by expert $e_i$ when transferring a task. If in $\gamma_i$, the sum of the components corresponding to TSR-related routing patterns is larger than that corresponding to TNR-related patterns, expert $e_i$ is regarded as using TSR more to transfer a task; otherwise the expert is using TNR more. Therefore, we can roughly divide experts into two groups: TNR-kind and TSR-kind. After an expert transfers a task, we count the number of remaining experts needed to resolve the task. We respectively summarize the distribution of the remaining expert number when a TSR-kind expert transfers a task, and that when a TNR-kind expert transfers a task. Figure 4.3 shows the results on the DB2 tickets. It clearly verifies the hypothesis. On DB2 tickets, a ticket will likely get solved with one more step when an expert favoring TSR routes it. However, if routed by a TNR-kind expert, a ticket might still need 2 or 3 more experts to get resolved. We obtain an additional implication from Table 4.3 and Figure 4.3, that is, TNR+TSR better captures the expert real routing behaviors in a collaborative network while routing based on TSR can lead to more efficient task resolution. Due to space constraints, we omit the results for AIX and WebSphere, which are very similar to that of DB2.

### 4.4.4 Expertise Difference Routing Pattern

In our EX routing pattern, given the expertise of $e_i$ and $e_j$, we estimate $f_{ij}$, *i.e.*, the general trend of $e_i$ sending a task to $e_j$, based on a log-normal distribution of $\Delta(e_i, e_j)$. The selection of log-normal is due to the non-negative nature of $\Delta(e_i, e_j)$ and the asymmetric shape of the distribution shown in Figure 4.1. However, one might consider estimating $f_{ij}$ based on a normal distribution, since a normal distribution also seems to be quite similar to Figure 4.1:

$$f_{ij} \propto e^{-\frac{[\Delta(e_i,e_j)-\mu]^2}{2\sigma^2}}. \tag{4.25}$$

Table 4.4 empirically justifies our selection of a log-normal distribution (Step Loss is shown in percentage and LL is ×$10^4$). (TNR+TSR)$^{\#}$ is the result corresponding to using a normal distribution to estimate $f_{ij}$ based on Eqn. 4.25, which is much worse compared with TNR+TSR.

Instead of optimizing $\mu$ and $\sigma^2$ during model solution, we could pre-estimate $\mu$ and $\sigma^2$ based on the distribution of the relative expertise difference in the training dataset, as shown in Figure 4.1, and keep them fixed during model training. We

| | DB2 | | | WebSphere | | | AIX | | |
|---|---|---|---|---|---|---|---|---|---|
| Models | SL | MAE | LL | SL | MAE | LL | SL | MAE | LL |
| TNR+TSR | **1.77** | **0.08** | **-0.07** | **1.44** | **0.07** | **-0.19** | **1.99** | **0.15** | **-0.17** |
| (TNR+TSR)# | 3.54 | 0.22 | -0.18 | 3.67 | 0.24 | -0.55 | 4.01 | 0.34 | -0.38 |
| (TNR+TSR)* | 1.90 | 0.10 | -0.08 | 1.59 | 0.08 | -0.20 | 2.12 | 0.17 | -0.19 |

Table 4.4: Variants of EX routing pattern.

denote this setting as (TNR+TSR)*. As discussed in Section 4.2.1, the histogram in Figure 4.1 is due to the integrated effects of all the routing patterns, whereas TNR+TSR optimizes $\mu$ and $\sigma^2$ with more emphasis on tasks transferred following the EX pattern, and can further improve the accuracy. Nevertheless, given that the performance does not differ too much between TNR+TSR and (TNR+TSR)*, in practice, one might consider saving the trouble of deriving complicated derivatives over $\mu$ and $\sigma^2$ during model solution.

### 4.4.5 Optimizing Collaborations

In the management of real collaborative networks, system administrators need to optimize the current network, e.g., in terms of expert training, to improve the efficiency of task execution. However, it is very expensive, if not impossible, to alter the real collaborative network just for hypothesis testing. Currently such decisions are manually made by experienced managers or consultants, without much quantitative analysis on how the resulting network will perform. Since our model accurately captures the routing behaviors of experts, it can naturally serve as a trustable simulation means for real task routing in the collaborative network. Hypotheses on whether a certain change to the network can improve the efficiency or not, can be much more easily examined with the help of our model.

Here we study optimization of the collaborative network, in terms of training experts to have more efficient routing patterns. Particularly, we examine two questions: *What kind of routing patterns might bring better resolution efficiency? Which expert(s) should be selected for more training, given a limited budget?* Section 4.4 implies if an expert is more likely to route a task based on TSR, the task will be resolved more quickly. We now formally verify this hypothesis. For each expert, we treat TSR and TNR as two groups of routing patterns, and set the group mixture weights respectively as $(x, 1-x)$. When an expert transfers a task, we first randomly select TSR or TNR based on the group mixture weights, and then select a routing pattern inside the selected group according to their mixture weights previously learnt in our model TNR+TSR. Based on Section 4.4, we estimate the completion time of a task, and evaluate the task resolution efficiency by the average CT of all the tasks. We vary $x$ to test the change of the task resolution efficiency. Only the results in the DB2 tickets are shown in Figure 4.4 since similar results on WebSphere and AIX are observed. We can see that as the mixture weight for TSR gets closer to 1, the average completion time tends to become shorter, indicating task resolution becomes more efficient. Therefore, experts in the network should be trained to route a task based on TSR.

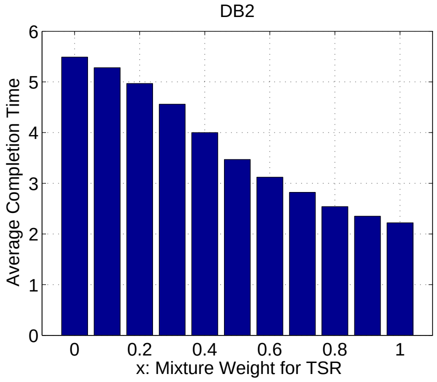


Figure 4.4: Effect of TSR weights.

When the training budget is limited, which experts should be trained first to maximize the performance of the entire network, is an interesting problem. For simplicity, we consider the problem of selecting the best candidate. One can extend to top-k candidate selection by adopting a greedy method. Possible methods to recommend an expert include (1) randomly select one expert from the network, denoted as *Random*; (2) select the expert that transfers the most tasks, denoted as *Frequent Transferrer*; (3) select the expert that is the least efficient: after the expert transfers a task, the average number of remaining steps to solve a task is the highest, denoted as *Least Efficient*; (4) use our model to conduct task routing after an expert's routing pattern is changed and select the expert that can lead to the most improvement of efficiency. Efficiency improvement is evaluated by the decrease of the average CT of tasks. Methods (2)-(4) are executed on a training

| Methods | Efficiency Improvement (%) |
|---|---|
| Random | 0.27 |
| Frequent Transferrer | 0.91 |
| Least Efficient | 1.21 |
| Recommendation with Our Model | **2.75** |

Table 4.5: Training recommendation.

task set and evaluated by calculating the efficiency improvement on a testing task set. Table 4.5 clearly demonstrates that compared with other methods, training the expert recommended with the help of our model, can result in a much more efficiency improvement. This result is expected because through routing a training set of tasks with our model, we are able to know which expert's routing pattern plays a critical role in decreasing the average CT of tasks. This study demonstrates that our model could help conduct hypothesis testing easily and can provide valuable recommendations to decision makers during the optimization of a collaborative network.

# Chapter 5

# Conclusion

Question answering (QA), which is concerned with directly returning precise answers to natural language questions, has been advocated as the key problem for advancing web search [22]. In fact, not only general web search, QA techniques are transforming the way to acquire knowledge in many domains such as querying relational databases via natural language interfaces, healthcare consulting, customer service, robotics etc. In this dissertation, we mine disparate information sources granted by big data age, including texts linked with knowledge bases, tables, and human networks, for question answering.

In Chapter 2, we develop a novel web texts based question answering framework with KBs as a significant auxiliary. Driven by the incompleteness problem of KBs, our system directly mines answers from the web texts and shows great advantages in questions not necessarily answerable by KBs. Unlike existing text-based QA systems, our system links answer candidates to KBs during answer candidate generation, after which, rich semantics of entities, such as their description texts and entity types in KBs, are utilized to develop effective features for downstream answer candidate ranking. Compared with various QA systems, our system framework has achieved an $18\% \sim 54\%$ improvement under $F_1$.

Chapter 3 proposes an end-to-end framework to precisely locate table cells in millions of web tables for question answering. Our table cell search framework is compared with state-of-the-art KB-based QA systems. Through extensive experiments, we show that our framework could outperform other systems by a large margin on real-world questions mined from search engine logs. Our results also support the hypothesis that web tables are a good complement to knowledge bases, providing rich knowledge missing from existing knowledge bases.

We investigate employing human networks for question answering and particularly study mining expert behaviors in Chapter 4. We model the decision making and cognitive process of a human expert during task/question routing in collaborative networks. A routing decision of an expert is formulated as a result of a generative process based on multiple routing patterns. We formalize each routing pattern in a probabilistic framework, and modele experts' routing decision making through a generative model. Our analytical model has been verified that it not only explains the real routing sequence of a task very well, but also accurately predicts a task's completion time in the current collaborative network. In comparison with all the alternatives, our method improves the performance by more than 75% under three different measures. We also demonstrate that our model can provide guidance on optimizing the performance of collaborative networks.

## Research Frontiers

This dissertation is towards our long-term goal to advance knowledge discovery and problem solving systems that directly take natural language questions from users as inputs. To approach this goal, we shall continue to study both machine-aided (e.g., Chapter 2 and 3) and human-aided (e.g., Chapter 4) question answering and knowledge discovery systems separately, advance their respective effectiveness and efficiency. Meanwhile, we will design frameworks/mechanisms to marry up their advantages and develop both types of systems in a mutually reinforcing manner. Therefore, a wealth of research problems are awaiting our efforts to investigate:

(1) Deep Understanding of Human Behaviors. Existing human networks are far more complicated than the collaborative networks we have explored in this thesis. For example, Bugzilla[1] is a bug tracking system where software developers jointly fix the reported bugs in projects. Various companies, such as Amazon and oDesk, maintain a pool of freelancers to provide crowdsourcing services for users.

[1] Bugzilla: http://www.bugzilla.org/

Community question answering websites, such as Quora and Zhihu in China, attract a large pool of users to ask and answer questions they are interested in. Different systems have different mechanisms, and human behaviors in different systems can demonstrate different characteristics. Exploring all these types of human-aided systems will provide us a deep and comprehensive understanding on human behaviors for problem solving, which will ultimately help boost system efficiency. Other interesting future work include designing mechanisms and incentives for human-aided systems, in order to organize and motivate experts to solve problems in a more efficient manner.

(2) Full Exploration and Exploitation of Knowledge Bases. Although a wealth of valuable information is provided in knowledge Bases, they tend to be quite noisy and disorganized, which poses severe challenges in utilizing the information in an effective way. One manifestation of the information disorganization in Freebase lies in the cluttered types associated with each entity. In Freebase, an entity is usually associated with dozens of types. For example, entity *Barack Obama* is associated with types *person, president, politician, artist, celebrity*, etc. However, each type is by no means of the same significance under a given context. Given a relation tuple in knowledge bases <*Obama, President-Of, U.S.A*>, the type "*president*" and "*politician*" should be more important than others. It is important to rank the types of an entity based on their relevance w.r.t. a context, which will be

important in various scenarios including question answering, entity disambiguation and so on. Such information disorganization studies can also be applied to canonicalizing the relation information and building the hierarchy of entity types, where the common goal is to arrange information in knowledge bases in a more precise and organized manner, in order for systems relying on knowledge bases to make the most of them.

In terms of exploitation of various knowledge graphs, most of existing works focus on directly querying them to retrieve information. In [62], we explored the great potential of knowledge graphs as a significant auxiliary in question answering, where answer candidates extracted from the Web were linked to Freebase and semantic features were developed based on their types and description texts in Freebase. However, the information provided by knowledge bases is far beyond entity types and their description texts (e.g., the relations or paths between two entities). How to utilize such information to help question answering is worth studying in the future. Apart from question answering, massive amount of knowledge manually encoded in knowledge bases, such as relations among entities, naturally makes tremendous labeled information, which could be employed in NLP tasks such as sentence parsing and information extraction.

(3) Mutual Promotion and Effective Union of Two Types of Systems. Interesting studies to conduct are on the union of the two types of systems. For example,

given a problem, which type of systems should be employed to solve it? One might not find satisfying answers by searching Google or by reading lengthy technical manuals, whereas one might wait for a long time, yet still get low-quality, untrustworthy answers after consulting friends or an expert forum. What is a good measure that takes into account both the cost (waiting time) and the gain (answer quality)? These questions inspire future studies on leveraging machine-aided and human-aided systems in an effective manner for better task accomplishments.

(4) Advanced Text Mining via Incorporating Semantics. Text is a significant type of human knowledge carriers, and effective mining of useful information from diversified texts will significantly benefit intelligent knowledge discovery systems. Traditional text mining techniques mostly rely on bag-of-words models, which only take into account the co-occurrence statistics of words for modeling. As a known fact, word semantics and linguistic structures, referred to as *semantics* here, play important roles in understanding the meaning of texts: (i) Word semantics. Language models and deep neural networks have been proposed to embed each word into an N-dimensional space, where words sharing similar semantic meanings are close to each other in the embedded space; (ii) Structural semantics, such as those obtained by sentence parsers. Recurrent structural patterns in large-scale texts, together with word semantic representations and word co-occurrence statistics, shall all be combined in a systematic way to capture deep meanings underlying

surface texts. Future work in this line could be to systematically investigate the potential of incorporating word and structural semantics into classic topic modeling approaches. Combining such semantics with statistical information in big data is promising in significantly boosting traditional text mining techniques, especially for short texts where word co-occurrence patterns are inconspicuous.

(5) Domain-specific Knowledge Discovery. Apart from the previously discussed knowledge graph, texts, and tables, other domain-specific data such as health records, are also of huge significance in our daily life. By effectively mining such data, one could build more advanced domain-specific knowledge discovery systems, such as doctor assistant systems to recommend effective and affordable treatments. Specifically, given vast amounts of medical information, such as in scientific reports or health records, tremendous knowledge mining efforts could be made, including (i) extracting entities in the medicine domain and linking them to knowledge bases, and detecting relations between two entities. Deducing the causes of certain symptoms and their potential treatments can be conducted later based on the extracted relations; (ii) capturing correlation patterns between treatment and outcome, based on historical treatment-outcome pairs; (iii) among various information sources including different scientific journals and medical specialists, identifying those trustable or accurate ones so that information from them

should be put more emphasis on. Such mined knowledge will undoubtedly play significant roles in assisting doctors to make wiser decisions in different situations.

In summary, this dissertation aims at unleashing the power of disparate information sources for question answering and knowledge discovery. We deal with fundamental limitations of current automated QA, and critical factors that affect the efficiency and effectiveness of human collaborative QA. Our work helps us build close collaborations with researchers at Microsoft Research, Army Research Lab, IBM Research, and Institute for Collaborative Biotechnologies. Our recent collaboration with Baidu Research Big Data Lab initiates a series of exciting research projects on deep question answering in healthcare forums to assist disease diagnosis and generate treatment recommendations. Not only in healthcare, techniques for understanding natural language questions and for mining answers from disparate sources could be applied in any other domain where people desire to access knowledge more effortlessly. For example, in software engineering, how to retrieve code snippets available on the Web for re-use, or more excitingly generate code snippets based on a natural language task description, could largely promote project development efficiency. Such topics are of interdisciplinary nature and request collaborative efforts from researchers in natural language processing, crowdsourcing, human computer interaction, software engineering, etc. My upcoming visit to the Computer Science Department at the University of Washington is dedicated to establishing such collaborations with experts in these areas. Hence, our research studies unfold a bright future for data mining to boost natural language question answering in various domains, and for jointly utilizing human intelligence and machine intelligence for problem solving and decision making.

# Bibliography

[1] Freebase wiki. `http://wiki.freebase.com/wiki/Wikipedia`.

[2] Hg data. *hgdata.com*.

[3] http://en.wikipedia.org/wiki/ibmwebsphere.

[4] Ion Androutsopoulos, Graeme D Ritchie, and Peter Thanisch. Natural language interfaces to databases–an introduction. *Natural language engineering*, 1(01):29–81, 1995.

[5] Krisztian Balog and Robert Neumayer. Hierarchical target type identification for entity-oriented queries. In *CIKM*, pages 2391–2394. ACM, 2012.

[6] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on Freebase from question-answer pairs. In *EMNLP*, pages 1533–1544, 2013.

[7] Jonathan Berant and Percy Liang. Semantic parsing via paraphrasing. In *ACL*, 2014.

[8] Christopher Bishop et al. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

[9] C.M. Bishop. *Pattern recognition and machine learning*. 2006.

[10] David Blei, Andrew Ng, and Michael Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[11] Eric Brill, Susan Dumais, and Michele Banko. An analysis of the AskMSR question-answering system. In *EMNLP*, pages 257–264, 2002.

[12] Eric Brill, Jimmy J Lin, Michele Banko, Susan T Dumais, and Andrew Y Ng. Data-intensive question answering. In *TREC*, 2001.

[13] ChristopherJC Burges. From RankNet to LambdaRank to LambdaMART: An overview. *Learning*, 11:23–581, 2010.

[14] ChristopherJC Burges, Krysta Marie Svore, Paul N Bennett, Andrzej Pastusiak, and Qiang Wu. Learning to rank using an ensemble of lambda-gradient models. In *Yahoo! Learning to Rank Challenge*, pages 25–35, 2011.

[15] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *VLDB*, 1(1):538–549, 2008.

[16] Michael J Cafarella, Alon Y Halevy, Yang Zhang, Daisy Zhe Wang, and Eugene Wu. Uncovering the relational web. In *WebDB*. Citeseer, 2008.

[17] Jennifer Chu-Carroll, John Prager, Christopher Welty, Krzysztof Czuba, and David Ferrucci. A multi-strategy and multi-source approach to question answering. Technical report, DTIC Document, 2006.

[18] Nello Cristianini and John Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[19] Silviu Cucerzan and Avirup Sil. The msr systems for entity linking and temporal slot filling at TAC 2013. In *Text Analysis Conference*, 2013.

[20] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. Finding related tables. In *SIGMOD*, pages 817–828. ACM, 2012.

[21] Xin Dong, K Murphy, E Gabrilovich, G Heitz, W Horn, N Lao, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A Web-scale

approach to probabilistic knowledge fusion. In *SIGKDD*, pages 601–610, 2014.

[22] Oren Etzioni. Search needs a shake-up. *Nature*, 476(7358):25–26, 2011.

[23] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *EMNLP*, pages 1535–1545, 2011.

[24] Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. Paraphrase-driven learning for open question answering. In *ACL*, pages 1608–1618, 2013.

[25] Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. Open question answering over curated and extracted knowledge bases. In *SIGKDD*. ACM, 2014.

[26] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the DeepQA project. *AI magazine*, 31(3):59–79, 2010.

[27] Jerome Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

[28] Jianfeng Gao, Patrick Pantel, Michael Gamon, Xiaodong He, Li Deng, and Yelong Shen. Modeling interestingness with deep neural networks. In *EMNLP*, 2014.

[29] Fangqiu Han, Shulong Tan, Huan Sun, Xifeng Yan, Mudhakar Srivatsa, and Deng Cai. Distributed representations of expertise. SDM, 2016.

[30] Geoffrey Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[31] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In *NIPS*, pages 2042–2050, 2014.

[32] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*, pages 2333–2338. ACM, 2013.

[33] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *CVPR*, 2015.

[34] Jeongwoo Ko, Eric Nyberg, and Luo Si. A probabilistic graphical model for joint answer ranking in question answering. In *SIGIR on Rearch and Development in IR*, pages 343–350. ACM, 2007.

[35] Cody Kwok, Oren Etzioni, and Daniel Weld. Scaling question answering to the Web. *TOIS*, 19(3):242–262, 2001.

[36] Adam Lally, John Prager, Michael McCord, BK Boguraev, Siddharth Patwardhan, James Fan, Paul Fodor, and Jennifer Chu-Carroll. Question analysis: How watson reads a clue. *IBM Journal of Research and Development*, 56(3.4):2–1, 2012.

[37] Fei Li and HV Jagadish. Constructing an interactive natural language interface for relational databases. *VLDB*, 8(1):73–84, 2014.

[38] Xin Li and Dan Roth. Learning question classifiers. In *ICCL*, pages 1–7, 2002.

[39] Yunyao Li, Huahai Yang, and HV Jagadish. Nalix: an interactive natural language interface for querying xml. In *SIGMOD*, pages 900–902. ACM, 2005.

[40] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *VLDB*, 3(1-2).

[41] Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[42] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[43] Elaine Marsh and Dennis Perzanowski. MUC-7 evaluation of ie technology: Overview of results. In *MUC-7*, volume 20, 1998.

[44] Gengxin Miao, Louise E Moser, Xifeng Yan, Shu Tao, Yi Chen, and Nikos Anerousis. Generative models for ticket resolution in expert networks. In *SIGKDD*, pages 733–742. ACM, 2010.

[45] Bonan Min, Ralph Grishman, Li Wan, Chang Wang, and David Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *HLT-NAACL*, pages 777–782, 2013.

[46] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45, 1997.

[47] Herbert Moskowitz and Kwei Tang. Bayesian variables acceptance-sampling plans: quadratic loss function and step-loss function. *Technometrics*, 34(3):340–347, 1992.

[48] J William Murdock, Aditya Kalyanpur, Chris Welty, James Fan, David A Ferrucci, DC Gondek, Lei Zhang, and Hiroshi Kanayama. Typing candidate answers using type coercion. *IBM Journal of Research and Development*, 56(3.4):7–1, 2012.

[49] SeungHoon Na, InSu Kang, SangYool Lee, and JongHyeok Lee. Question answering approach using a WordNet-based answer type taxonomy. In *TREC*, 2002.

[50] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[51] Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. *ACL*, 2015.

[52] Christopher Pinchak and Dekang Lin. A probabilistic answer type model. In *EACL*, 2006.

[53] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157. ACM, 2003.

[54] Mengye Ren, Ryan Kiros, and Richard Zemel. Exploring models and data for image question answering. In *Advances in Neural Information Processing Systems*, pages 2935–2943, 2015.

[55] Stephen Robertson and Hugo Zaragoza. On rank-based effectiveness measures and optimization. *Information Retrieval*, 10(3):321–339, 2007.

[56] Ashutosh Saxena, Ashesh Jain, Ozan Sener, Aditya Jami, Dipendra K Misra, and Hema S Koppula. Robobrain: Large-scale knowledge engine for robots. *preprint arXiv:1412.0691*, 2014.

[57] Nico Schlaefer, Petra Gieselmann, Thomas Schaaf, and Alex Waibel. A pattern learning approach to question answering within the ephyra framework. In *Text, speech and dialogue*, pages 687–694. Springer, 2006.

[58] Qihong Shao, Yi Chen, Shu Tao, Xifeng Yan, and Nikos Anerousis. Efficient ticket routing by resolution sequence mining. In *SIGKDD*, pages 605–613. ACM, 2008.

[59] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Gregoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *CIKM*, pages 101–110. ACM, 2014.

[60] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *WWW companion*, pages 373–374, 2014.

[61] Huan Sun, Hao Ma, Xiaodong He, Wen-Tau Scott Yih, Yu Su, and Xifeng Yan. Table cell search for question answering. *Under Review*, 2016.

[62] Huan Sun, Hao Ma, Wen-tau Yih, Chen-Tse Tsai, Jingjing Liu, and Ming-Wei Chang. Open domain question answering via semantic enrichment. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1045–1055. International World Wide Web Conferences Steering Committee, 2015.

[63] Huan Sun, Mudhakar Srivatsa, Shulong Tan, Yang Li, Lance M Kaplan, Shu Tao, and Xifeng Yan. Analyzing expert behaviors in collaborative networks. In *SIGKDD*, pages 1486–1495. ACM, 2014.

[64] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[65] Chen Tsai, Wentau Yih, and ChrisJ.C. Burges. Web-based question answering: Revisiting AskMSR. Technical Report MSR-TR-2015-20, Microsoft Research, 2015.

[66] Christina Unger, Lorenz Bühmann, Jens Lehmann, AxelCyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. Template-based question answering over RDF data. In *WWW*, pages 639–648. ACM, 2012.

[67] Ellen M Voorhees and Dawn M Tice. Building a question answering test collection. In *SIGIR on Rearch and Development in IR*, pages 200–207. ACM, 2000.

[68] Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge base completion via search-based question answering. In *WWW*, pages 515–526, 2014.

[69] Ryen W. White, Matthew Richardson, and Wentau Yih. Questions vs. queries in informational search tasks. Technical Report MSR-TR-2014-96, Microsoft Research, 2014.

[70] Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, Maya Ramanath, Volker Tresp, and Gerhard Weikum. Natural language questions for the Web of data. In *EMNLP-CoNLL*, pages 379–390, 2012.

[71] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD*, pages 97–108. ACM, 2012.

[72] Yi Yang and Ming-Wei Chang. S-mart: Novel tree-based structured learning algorithms applied to tweet entity linking. *ACL*, 2015.

[73] Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked attention networks for image question answering. *arXiv preprint arXiv:1511.02274*, 2015.

[74] Xuchen Yao and Benjamin Van Durme. Information extraction over structured data: Question answering with Freebase. In *ACL*, 2014.

[75] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. *ACL*, 2015.

[76] Haoqi Zhang, Eric Horvitz, Yiling Chen, and David C Parkes. Task routing for prediction tasks. In *AAMS-Volume 2*, pages 889–896. IFAAMS, 2012.

[77] Linchao Zhu, Zhongwen Xu, Yi Yang, and Alexander G Hauptmann. Uncovering temporal context for video question and answering. *arXiv preprint arXiv:1511.04670*, 2015.

[78] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffer Xu Yu, Wenqiang He, and Dongyan Zhao. Natural language question answering over RDF: a graph data driven approach. In *SIGMOD*, pages 313–324. ACM, 2014.