Question Paraphrase Generation for Question Answering System

by

Haocheng Qin

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2015

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The queries to a practical Question Answering (QA) system range from keywords, phrases, badly written questions, and occasionally grammatically perfect questions. Among different kinds of question analysis approaches, the pattern matching works well in analyzing such queries. It is costly to build this pattern matching module because tremendous manual labor is needed to expand its coverage to so many variations in natural language questions. This thesis proposes that the costly manual labor should be saved by the technique of paraphrase generation which can automatically generate semantically similar paraphrases of a natural language question. Previous approaches of paraphrase generation either require large scale of corpus and the dependency parser, or only deal with the relation-entity type of simple question queries. By introducing a method of inferring transformation operations between paraphrases, and a description of sentence structure, this thesis develops a paraphrase generation method and its implementation in Chinese with very limited amount of corpus. The evaluation results of this implementation show its ability to aid humans to efficiently create a pattern matching module for QA systems as it greatly outperforms the human editors in the coverage of natural language questions, with an acceptable precision in generated paraphrases.

Acknowledgements

I would like to express the deepest appreciation to my supervisor, Professor Ming Li, for his enthusiasm, his encouragement, and his inspirational and patient guidance. Without his guidance and persistent help this thesis would not have been possible.

I would like to thank the readers of my thesis, Professor Chrysanne DiMarco and Professor Grant Weddell, for being willing to spend time and efforts on reviewing my work. Their advice is very important to me.

A thank you to Kun Xiong, Anqi Cui and Guangyu Feng, who give me a lot of help and advice in my work.

Also, thank my family and all my friend for always supporting me.

Dedication

This is dedicated to my parents who give me so much support in my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

If the bottleneck of a search engine is the amount of information, this is no longer true for a question answering (QA) system where exactly one answer is expected. A modern QA system fails to answer a question usually not because it did not have the answer but because it did not understand the question. There are currently two ways to understand a question:

- Keyword-based systems. These approaches usually are too aggressive with many false positives. If you ask Siri [1]: What does a cat eat? What does a dog eat? The answer is a list of human restaurants [iPhone 6+, June 6, 2015].
- Template-based systems. These systems, such as Evi [2], IBM's Watson [17], or Wolfram Alpha [3], strongly depend on the exhaustive enumeration of templates or patterns. A slight ungrammatical variation: "Who is the Toronto's mayor" spoils an otherwise answerable question [evi.com, July 13, 2015].

We believe a general-purpose open domain QA system will need a combination of these two approaches. This work focuses on the second approach using templates and patterns. Our research aims at the following three application scenarios:

1. Natural language query to database query. Assuming we already know how to map "What is the population of Canada?" to an SQL query to a knowledge database such

[1] Apple's voice assistant, http://www.apple.com/ios/siri/
[2] Amazon's voice assistant, http://www.evi.com
[3] A computational knowledge engine developed by Wolfram Research, https://www.wolframalpha.com

as FreeBase [8], can we also map "How many people live in Canada?" to the same SQL query?

2. Search community QA databases. A community QA database contains hundreds of millions of QA pairs created by humans. Such a database may contain an answer to "What is the population of Canada", but not for "What population Canada".

3. Generalizing existing FAQs. While this is in principle similar to item 2, we list this separately as the application scenario is different. Websites of millions of organizations have FAQs attempting to answer some of the user questions. While this might be acceptable for human readers, it is certainly too restrictive for automatic customer service.

In all three applications, it is important to generate a list of paraphrases, grammatical or not, with "similar meanings". Before we proceed we answer two questions:

1. Do we need to generate paraphrases offline or measure "paraphrase distances" online? There are several reasons in practical systems that such paraphrases need to be generated offline, a priori. A main pragmatic reason is actually the need for human programmers to write regular expression or context free grammar (CFG) [36] type of templates for vertical QA domains. In this case, our system is designed to assist them to generate all possible templates and patterns from one question. In the ideal world, if we could generate all variations, then this automatically solves item 3 above.

2. Paraphrases are defined to be "phrases with similar meanings". What do we mean by "similar meanings"? The original Greek "$\pi\alpha\rho\alpha\varphi\rho\acute{\alpha}\sigma\varepsilon\iota\varsigma$" means "additional manner of expression" that has elegantly avoided the trouble of defining "similar meaning". In Feng et al. paper [16], it is proved that Information Distance [6] can be used to optimally approximate the undefined concept of "similar meaning" or "semantic distance". Thus, when we talk about "similar meaning", it can be interpreted as relatively short encoding (length) between two respective paraphrases.

In most of QA systems, there is a "question analysis" module that converts user's natural language input to a logical query that is passed to the following modules for retrieving answers. Usually, the goal of question analysis is to cover as many variations as possible of user's natural language questions.

There are several common methods to do question analysis. One is question classification and reformulation [1]. In this approach, the question is first classified based on

its type and topic. According to the type and topic, some key words are extracted as query for further answer retrieval. Another popular method is semantic parsing [24, 2]. This method is mostly used for the vertical QA systems with structural knowledge base such as a geographical system. It uses combination grammars [3] to parse the question and map the parsing result to the query in a logical form such as lambda calculus [4] or Dependency-Based Compositional Semantics (DCS) trees [24]. In addition, we have developed a method that is based on pattern match. In the question analysis of pattern-based match, the input question will be matched to one or several patterns, which are usually handwritten in CFG, and the patterns are paired with corresponding logical queries.

In a template or pattern-based approach, for each question, there are almost an infinite number of paraphrases that express the similar meanings. In most cases, for human editors, it is very inefficient to enumerate these paraphrases. Thus, practical QA systems based on the handwritten templates or patterns usually have low coverage of the question variations. In order to improve both the efficiency and the coverage of using pattern-based match to build up question analysis module, it is vital to improve the efficiency of template or pattern writing. More precisely, our system aims to reduce the manual work in writing patterns by automatically generating the templates or patterns.

For keyword queries, question expansion is mostly about morphology variants and synonyms. The paraphrases of a natural language question not only contain the variants of words and phrases, but also different sentence structures. Though the question expansion can make better coverage for the question analysis, it can barely generate sentential paraphrases that with different sentence structures.

To generate paraphrases of sentences, many studies have been done using different approaches [15, 40, 25, 5, 38, 14, 39]. These approaches have provided different methods to generate sentential paraphrases for different inputs in different applications. However, they either focus only on factual questions that can extract relation and entities [15], or on phrasal query extension [40], or need dependency tree parsing [25], or need very large scale of parallel corpus [15, 25, 5]. When building the QA systems in non-English languages that aim to handle more than factual questions, it is very difficult to obtain large scale of parallel corpus. Furthermore, with the complexity of some languages such as Chinese, there is no parser doing dependency tree parsing with high accuracy. So we need a method for sentential paraphrase question generation which can generate as many variations as possible, against limited corpus and limited dependency parsing.

In this work, we propose an inference method of transformation operations from a sentence to its paraphrases, and a description of sentence structure. In addition, we introduce a method to generate paraphrases, from given input question in natural language. This

method generates paraphrases by applying proper transformation operations on the input sentence, based on its sentence structure. With the supervised parallel corpus which are consist of paraphrase questions that either handwritten or extracted from the community-based QA databases, this method can generate a ranked list of paraphrase patterns of the input question, which can assist editors in pattern writing for pattern-based question analysis module. The quality of paraphrase generation depends on the scale of corpus, which can be improved by adding correct paraphrases from generation results. The evaluation shows that with limited scale of initial corpus in Chinese language, our method can generate far great number of correct paraphrases than human editors, with acceptable precision.

We partially solve the problem of generating sentential paraphrases for both English and non-English questions with limited corpus. With this work, in pattern-based matching system, the human editors no longer need to generalize and handwrite all the patterns. Instead, they only need to select the generated paraphrase patterns from the input questions in natural language, with a little or no modification. Therefore, both the coverage of patterns and the efficiency of pattern writing can be greatly improved, which will finally make it easier to apply the pattern-based matching in question analysis to open domain QA systems.

Chapter 2

Related Work

2.1 Question Analysis Techniques

Serving as an important part in QA systems, question analysis module greatly influences the performance of the whole QA system. Many different approaches have been applied in different QA systems.

Many early systems and some web-based systems [18] use the traditional way of question analysis, which has two steps, question classification and question reformulation [1].

In order to provide constraints for the type of the answer and target to the relevant data, knowing the type and topic of a question usefully helps in the question reformulation and guides other modules of the QA system to retrieve and process the answer. With the input question classified by its topic and type, the question analysis module can extract the keywords from the question to form the query or use certain rules to generate the query. Then the query will be passed to the answer retrieval part.

To classify the type and topic of the question, there are two ways in general. The rule-based method is mostly used for question type classification. It depends on the rules which define the features of different types of questions. E.g., Stoyanchev et al. [33] have done a QA system that uses rule-based question classification. The other one is the supervised learning method, which uses large scale of labelled corpus and supervised learning classifiers, such as SVM [35], Naïve Bayes [28], Decision Tree [19] and SNoW [13], to train for the features and classify the type and topic of the question. E.g., Li et al. [22] have used supervised learning in the similar tasks.

The method of question classification and reformulation analyzes questions syntactically, which means it is language independent. In such methods, except for few rules or some type-annotated questions, not much manual work is needed, and unstructured data like text documents and web pages can be used as knowledge sources. When the question becomes complicated in sentence structure, the correctness of query generation, especially when based on keywords extraction, is uncertain, even with the topic and type of question classified correctly.

Instead of syntactical analysis, semantic parsing provides more powerful analysis for complex questions. In semantic parsing, the question can be mapped to a query in logical form, such as lambda calculus, which can use a formal system to express computation and supports high-order functions, so it fits better with structural knowledge base.

In 2012, Luke S. Zettlemoyer and Michael Collins introduced an algorithm that parses questions with Combinatory Categorial Grammars (CCG) and expresses the semantics in lambda calculus [37]. For example, the question

"which states border texas"

is parsed as

$\lambda x.state(x) \wedge next\_to(x, tex)$.

In this approach, a lexicon is the core of a CCG which describes a word by capturing its syntactic and semantic information jointly, as in the following:

$Texas \vdash NP : tex$, and $border \vdash S\backslash NP/NP : \lambda x \lambda y.next\_to(y, x)$.

A CCG has a set of combinatory rules which describe how adjacent syntactic categories in a string can be recursively combined, like

$X/Y : f \ Y/Z : g \Rightarrow X/Z : \lambda x.f(g(x))$,

and

$Y\backslash Z : g \ X\backslash Y : f \Rightarrow X\backslash Z : \lambda x.f(g(x))$.

By learning a training set of sentences which are labeled with expressions in the lambda calculus, the semantic parser can be built to parse natural language questions to lambda calculus expressions.

Typically, a semantic parser using CCG needs to be trained from examples of questions annotated with their target logical forms. Instead of spending such expensive cost, in 2013, Percy Liang et al. develop a new semantic formalism, dependency-based compositional semantics, that enables us to learn a semantic parser from question-answer pairs where the

intermediate logical form (a DCS tree or a lambda DCS [23]) is induced in an unsupervised manner [24]. For example, the sentence

"people who have lived in Seattle"

can be parsed as a lambda DCS

"$PlacesLived.Location.Seattle$".

In DCS, given a utterance, a set of lexical triggers is generated. Each lexical trigger is a pair (s, p), that s is a word or a word sequence, and p is a predicate (e.g., $s = California$ and $p = CA$). Based on the lexical triggers, a set of candidate DCS trees is recursively constructed by combining the trees of its subspans. More detailed, trees are combined by inserting relations and possibly other predicates between them. Similar to lambda calculus, the DCS tree or the lambda DCS can be used as a query to retrieve answers from structural knowledge base.

Semantic parsing is becoming more and more popular in recent research in natural language processing. a semantic parser analyzes the question semantically, it can handle questions with complicated sentence structures and automatically generate queries in logical form which can naturally compute the answer from a knowledge base. However, many preconditions are needed in this powerful approach. A semantic dictionary is required to annotate the words and the grammar rules are required to parse the question. These all need significant of costly manual work. In order to utilize the generated logical queries, a well-organized structural knowledge base is essential. Furthermore, semantic parsing is not language independent, so this method is very difficult to adapt to some languages which have the grammars too complicated to generalize correctly. Thus, in languages which have very variable and complicate grammars, semantic parsing may not be able to parse the questions with high degree of correctness.

In recent years, some question answering community websites appear. Users can post questions online, and they can also answer the questions posted by other users. The question answering system which uses the data from such websites as knowledge base is call community QA-based(CQA) system. In the CQA system, having the question answer pairs, the question analysis and answer retrieval are combined as a most similar question search, which searches for an answered question that is relatively most similar to the input question, and uses the answer of this question as the answer to the input. This approach provides a ranked list of answered questions based on their similarity to the query. To rank the similarity, there are many algorithms such as statistical similarity calculation with vector space models [29], and semantic distance estimation using WordNet [30]. Though the similarity analysis is relatively easy to implement, the approaches based on the most similar question search require a large database with question-answer pairs to provide

enough coverage. For the worse cases, slight differences between questions may lead to totally wrong answer, so that the similarity in questions cannot guarantee the correctness of answers.

For our application which involves both structural and unstructured knowledge bases, we introduce a question analysis approach based on pattern matching. In this approach, we define the pattern as an expression in CFG grammar. A series of patterns with their corresponding queries constitute the template files. Each item, including pattern and query, look like the following:

$$\begin{aligned} AskWeather = &What's|(What\ is)\ [the]\ weather\ [like]\ [<time>,<location>] \\ &\{query = weather(time, location)\} \end{aligned} \tag{2.1}$$

In this approach, the input question is tagged by a dictionary-based tagger. Then from the written templates, the pattern matcher finds the pattern that matches the generated tag sequence of the input question, and constructs the query based on the matched pattern. For example, the input "what's the weather like in Waterloo today" is tagged as "what's the weather like $<location><time>$", thus it matches to the category of "AskWeather", and the query "weather(today, Waterloo)" is generated.

The advantages of this approach are obvious. The tokens written in the patterns are all syntactical, which means that no semantics information is needed for building this matcher, so it is language-independent. The query generation is written by humans, so that it can adapt to both structural knowledge base query and other kinds of queries for unstructured data. More importantly, the patterns and query generating methods are readable and editable, so it is relatively convenient for the human editors to edit and debug the module, which ensures a quite high precision in analyzing the questions. In addition, for many QA systems which deal with limited kinds/domains of questions, only a small set of patterns is needed, and a good precision can be achieved, so it does not require much work in these cases. The disadvantage of this approach is also obvious, when serving in open domain QA systems, writing patterns to get good coverage requires a huge amount of work. This is the problem we aim to solve in this thesis.

2.2 Paraphrase Generation Techniques

To make the question analysis module recognize and analyze more variants of input questions, there are two main approaches: query expansion and paraphrase generation. Both

approaches have been researched for a long time and many different studies have been done.

The question expansion is mostly lexical and used for expanding the keyword queries. To expand the keyword queries, previous approaches focused mostly on words and phrases substitutions. Bilotti et al. [7] mention and compare the question expansion method that applies a stemming algorithm at indexing time, and the method that expands query terms with their morphological variants at retrieval time. With the idea of utilizing synonyms, many QA systems [12, 9, 31] use WordNet [30] to expand the query terms. Also, Statistical Machine Translation technology is used for question expansion. Riezler et al. [32] introduce an algorithm that uses foreign languages as pivot to generate synonyms and phrasal paraphrases. To synthesize the results of different methods, Zhao et al. [39] introduce a method of filtering and selecting the patterns, paraphrases and collocation variants which are generated by multiple methods.

Intuitively, the paraphrases of a question contain not only the variants of words and phrases, but also different sentence structures. Though question expansion can improve the coverage of the question analysis at the phrasal level, it can barely generate sentential paraphrases that varies in sentence structure, which might lead to significantly greater improvement. Since paraphrase generation techniques at the sentential level are more likely to improve the question analysis, this is what we focus more on.

Most of the paraphrase generation techniques need a corpus. For collecting paraphrases as a corpus, Dolan et al. [14] introduced and experimented with an unsupervised approach to extract a parallel corpus from news articles. Except for extracting sentences with very low edit distances, they assumed that the first two sentences in news articles are usually the summarization of the whole news article, and similar news articles might be reporting the same event. Thus, the summarization sentences of similar news articles, which can be aligned with low cost, are potentially paraphrases. With this hypothesis, they extracted and filtered the first two sentences from news articles which reported the same event, as paraphrases.

Fader et al. [15] introduced an approach for generating paraphrases of factual questions in structural knowledge-based QA system. The approach focuses on the factual questions which can generate query in relation and entities. Using paraphrase question clusters with noise as corpus, and initial annotated questions as seed, their method aligns the paraphrase questions using existing sentence alignment tool, and learns the derivations of lexicon which contains entities, relations and question patterns. For example, in the paraphrase pair "What is the population of New York?" and "How big is NYC", the pattern of the former question is "what is the R of E", where R is the relation *population* and E is the entity

$new-york$. With the alignment between these two questions, "population" aligns to "big" and "New York" aligns to "NYC", so "big" is mapped to relation *population*, "NYC" is mapped to entity $new - york$, and the pattern of the latter question "How R is E" is considered to be same as "what is the R of E". Additionally, this method uses parameter learning to eliminate the noise in corpus. However, in many question paraphrases, the words and phrases have very little in common (e.g., "taking train how to use phone to book ticket" $\leftrightarrow$ "calling which number can buy train ticket"). The paraphrases are sometimes too hard to align. Also there are many questions with complex sentence structure (e.g., "How many days of delay are permitted for me to pay the bill without fine") which cannot be described as simple relation and entities.

For applications with a structural knowledge base, Lin and Pantel [25] introduced an unsupervised approach that discovers similarity among relations by analyzing similarity of their contexts, and generates paraphrase questions by finding similar relations. This method parses the sentence into a dependency tree, and transforms the dependency tree to the path which can be expressed as the form of X relation Y. For example, for the sentence "John found a solution to the problem", the path between "John" and "problem" is:

[N:subj:V] $\leftrightarrow$ $find$ $\leftrightarrow$ [V:obj:N] $\rightarrow$ *solution* $\rightarrow$ [N:to:N]

(meaning "X $finds$ *solution to* Y").

It also supposes that in large scale of corpus, if the relations in paths share similar constitution of Xs and Ys, the relations are of same semantics. When the input question is parsed as path, the similar relations to the relation in path can be generated, then paraphrase questions as well. For example, the path "X_1 $finds$ *solution to* Y_1" may be similar to "X_2 *solved* Y_2", because there is a large overlap between all the X_1s and X_2s, and a large overlap between all the Y_1s and Y_2s. This approach requires a semantic parser to extract the relations from the sentences, which is not very feasible for some non-English languages like Chinese, as the grammar, word combination rules and word semantics in different contexts are too flexible and complex for semantic parsing to be accurate enough. Additionally, it needs very large scale of corpus to generalize the similarity of relations.

For more general usage, Barzilay et al. [5] introduced an approach that is based on multiple sequence alignment and is also unsupervised. This approach considers the sentence as slotted lattices with backbones and arguments. For example, "The surprise bombing injured twenty people, five of them seriously" can match to the lattice "X (injured/wounded) Y people, Z of them seriously" where X="The surprise bombing", Y="twenty" and Z="five". Based on parallel corpora extracted from the news articles, this method first clusters syntactically similar sentences, then generalizes the sentences in the clusters to

generate the slotted lattices which contain patterns that represent the sentence structures and slots that for filling the variables. During this process, in the sentences, the commonly appeared parts (commonly appeared in same cluster) are called backbones, and the varied parts are called arguments. When two syntactically similar sentences have the argument contents in common, they are considered as paraphrases. By pairing sentences which have similar arguments, in parallel corpora from news articles, the algorithm clusters paraphrase sentence patterns which can be used to derive paraphrase sentences from the input sentence. For example, "X (injured/wounded) Y people, Z of them seriously" is paraphrased to "Y were (wounded/hurt) by X, among them Z were in serious condition". This approach depends on having many paraphrases in the corpus to correctly analyze the slots in a sentence and pair the patterns. In fact, in many cases, the variations between paraphrases (slots) are not in fixed positions or components, almost any components in sentence can be varied to form a paraphrase.

When it comes to QA, sometimes the question queries in QA are just noun phrases. For example, "The advantage of iPhone" and "Why do people like to use iPhone" almost mean the same in QA. In 2011, Zhao et al. [40] introduced their approach for extending incomplete, short, phrasal queries to multiple complete questions, with community-based QA systems and previous user query logs. It associates user's input queries with the templates extracted from the finally selected questions, and generates questions from an incomplete query by applying the query into the associated templates.

Compared to the approaches above, our approach focuses more on generating as many variants as possible, for both factual and other complicated questions in English and non-English, with a smaller-sized corpus. We do this by discovering the structure of question sentences, and then the transformation operations between paraphrases.

Chapter 3

Approach

3.1 Objectives

Introduced in Chapter 2, we use pattern-based matching as an important method for question analysis in QA systems. But this method requires very costly human labour in writing pattern templates. For improving efficiency of writing pattern templates, we propose an approach which generates paraphrases (in CFG patterns) of input questions, so that human editors only need to write question instances in natural language, and get the recommended paraphrasing patterns.

Thus, the main task is paraphrase generation. In this field, as mentioned in Chapter 2, the current approaches mostly focus on English. As for applications in Chinese, there are some difficulties to adapt those approaches:

1. There is not enough corpus in Chinese
2. There is no semantics parser for Chinese
3. Chinese grammar is too flexible for dependency tree parsing

In this work, we mainly focus on generating paraphrases for questions in Chinese, with limited scale of corpus. The objective of this work is to generate more paraphrases that increase the coverage in question analysis, with as few conflicts as possible.

3.2 Sentence Transformation

In order to generate paraphrases, we need to figure out how a sentence transforms to its paraphrases.

The sentential paraphrases of a sentence contain more than phrasal substitutions, and many paraphrases even cannot be aligned with each other. Instead, paraphrases are usually in different sentence structures. E.g., "红烧肉(pork) 要(should) 怎么(How to) 做(cook) 比较好(better)" and "哪种(Which) 做法(recipe) 做(cook) 的 红烧肉(pork) 好吃(delicious)" are paraphrases, but they are in different sentence structures that cannot be well-aligned word-by-word.

In order to generate paraphrases, it is important to discover the inference method about how a sentence transforms to its paraphrases. Conventionally, the edit distance (Levenshtein distance) [21] which gives the minimum steps of elementary operations (add and delete), is commonly used for transforming any sentence to another.

But in paraphrase generation, the cost of transformation is not a main concern, instead, a description of the transformation need to represent the internal relation between paraphrases, and easy to be adapted in different cases.

For example, considering these two paraphrases,

1. 为什么(why)姚明(Yao Ming) 那么(very) 的([*auxiliary*])出名(famous)
2. 姚明(Yao Ming)是(is)因为(because)什么(what)出名(famous)的([*auxiliary*])

Figure 3.1 shows the transformation operations are based on edit distance calculation to transform sentence 1 to sentence 2, but intuitively, we consider that the transformation in paraphrasing is not just adding and deleting words. Instead, we assume that one sentence being a paraphrase of another sentence is because their sentence "templates" have same meaning and two sentences share same contents in their sentence "templates". In the example above, the reason of those two sentence being paraphrases is that "$<$ *someone* $>$ 是(is)因为(because)什么(what) $<$ *adjective* $>$ 的([*auxiliary*])" has the similar meaning to "为什么(why) $<$ *someone* $>$ 那么(very) 的([*auxiliary*]) $<$ *adjective* $>$", and they share the same contents "姚明(Yao Ming)" and "出名(famous)". So, the transformation procedure in our assumption is more like in Figure 3.2.

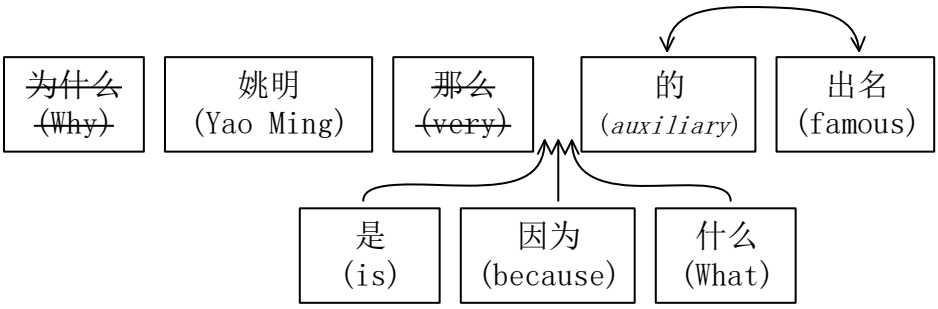

Figure 3.1: Transformation Based On Edit Distance

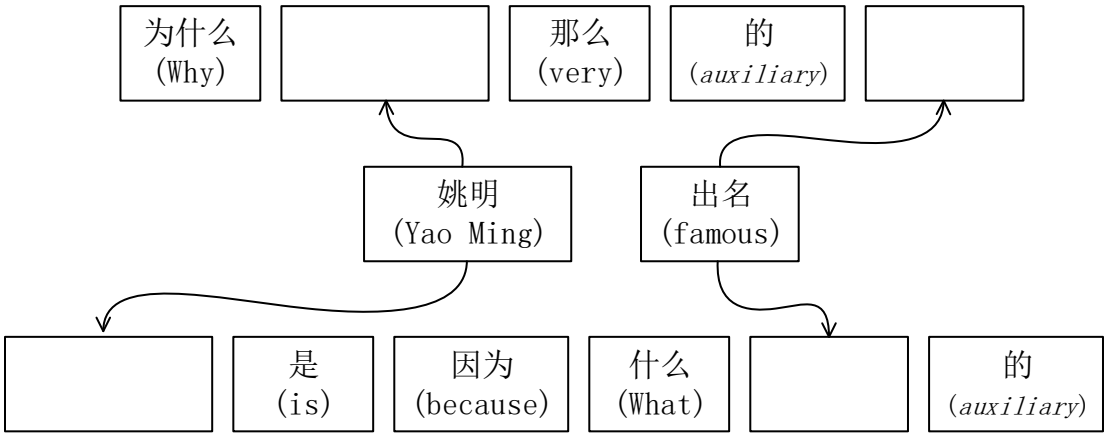

Figure 3.2: Transformation Based On Our Assumption

Inspired by such idea, we propose a method to infer the transformation from a sentence to its paraphrases. The transformation has two steps:

1. Find the common parts between two sentences, and the remaining parts is called the "templates" of sentences.

(Finding common parts is similar but different to Longest Common Subsequence problem [27], in that it does not require one-on-one alignment, the common parts do not have to be in order, and even "in common" does not mean the identical words.)

2. Keep the common parts, and fill them into the "templates" of the paraphrase sentence.

So, for each paraphrase pair, we consider one is transformed to the other by such operations (keep and fill). The detailed implementation will be introduced in Section 4.4.

3.3 Sentence Structure

In our approach, the structure of a sentence has a very important role, that the sentences with same sentence structures can apply same transformation operations to generate their paraphrases.

Unlike many approaches, we do not consider the sentence structure as a dependency tree, which is too difficult to be extracted correctly in Chinese, or as relation-entities, which cannot cover some complex sentences. Instead, we define the structure of sentence as a tag sequence. In such sequence, each tag represents a word/phrase in the sentence, and these tags can be either part-of-speech (POS) tags or functional tags such as $Why, How, Be, Let, Have$.

For example, the structure of

为什么(why)姚明(Yao Ming)那么(very)的([*auxiliary*])出名(famous)

is tagged as

$< Why >< People >< DegreeAdverb >< Auxiliary >< Adjective >$

Represented by such tag sequences, the structures of sentences can be compared and the sentences with same structure can be aligned as preparation for paraphrase generation.

3.4 Paraphrase Generation

With the sentence transformation operations and sentence structure defined, we propose an approach to generate paraphrases for input sentences.

Given an input sentence q_{input}, if there is a sentence q_s, which has the same sentence structure with q_{input}, can transform to its paraphrase by operations op, then q_{input} can also generate a paraphrase by such transformation operations op. Figure 3.3 shows an example of the generation procedure.

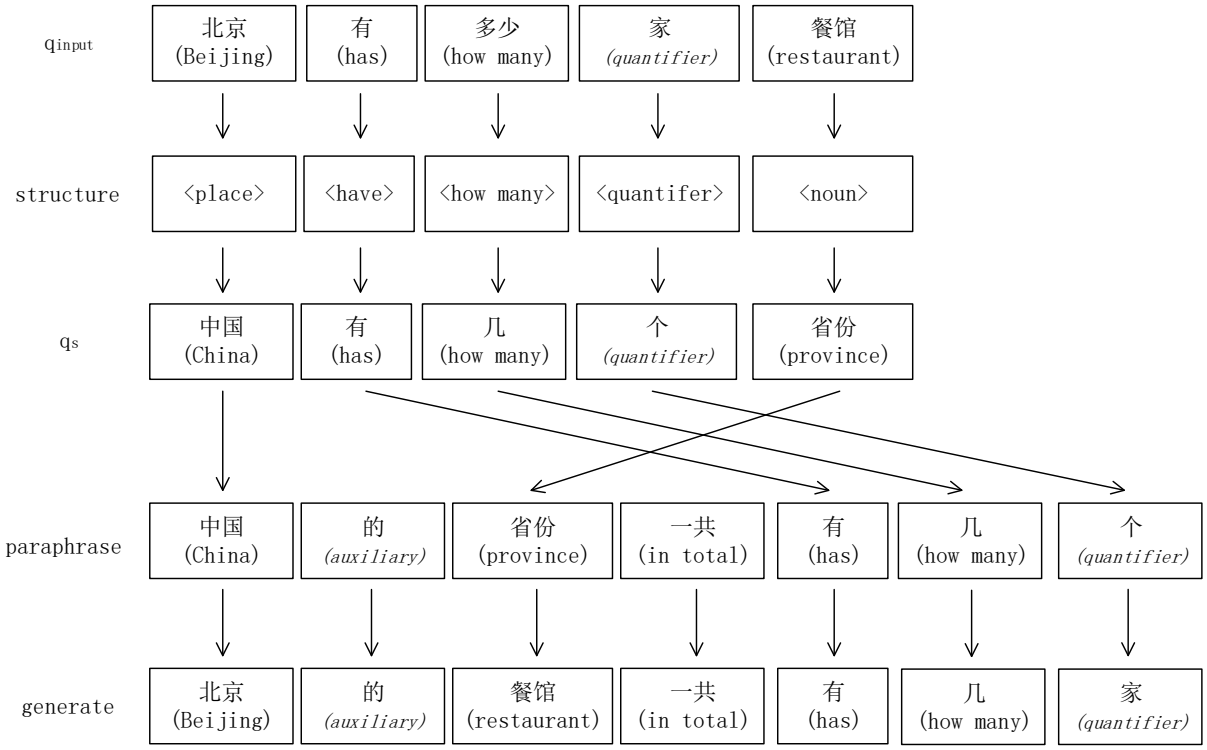

Figure 3.3: Paraphrase Generation Procedure Example

On the other hand, though q_{input} and q_s have same sentence structure, so if they are about very different topics, q_{input} is less likely to apply op to generate correct paraphrases. Thus, we propose that more similar q_{input} and q_s are, more likely q_{input} can apply op to generate paraphrases.

For example:

For the input: 北京(Beijing) 哪里(where) 有(has) 卖(sell) 寿司(sushi)

By the paraphrase pair 1: 多伦多(Toronto) 哪里(where) 有(has) 卖(sell) 龙虾(lobster) ⇒ 在(In) 多伦多(Toronto) 想(want) 吃(eat) 龙虾(lobster) 可以(can) 去(go) 哪(where)

It generates paraphrase: 在(In) 北京(Beijing) 想(want) 吃(eat) 寿司(sushi) 可以(can) 去(go) 哪(where)

By the paraphrase pair 2: 纽约(New York) 哪里(where) 有(has) 玩(play) 滑雪(skiing) ⇒ 纽约(New York) 的([*auxiliary*]) 滑雪(skiing) 公园(resort) 有(has) 哪些(what)

It generates paraphrase: 北京(Beijing) 的([*auxiliary*]) 寿司(sushi) 公园(resort) 有(has) 哪些(what)

As "多伦多(Toronto) 哪里(where) 有(has) 卖(sell) 龙虾(lobster)" is more similar to "北京(Beijing) 哪里(where) 有(has) 卖(sell) 寿司(sushi)", paraphrase "在(In) 北京(Beijing) 想(want) 吃(eat) 寿司(sushi) 可以(can) 去(go) 哪(where)" is more likely to be correct.

The next chapter will describe the implementation in detail.

Chapter 4

Implementation

4.1 Overview

Our approach has two parts: corpus processing and paraphrases generating.

The corpus processing part is done offline. With the corpus prepared, the offline processing part analyzes the questions in corpus, extracts their structures, calculates and stores the transformation operations from each question to its paraphrases. Figure 4.1 shows an example. The detailed algorithm is in Algorithm 1.

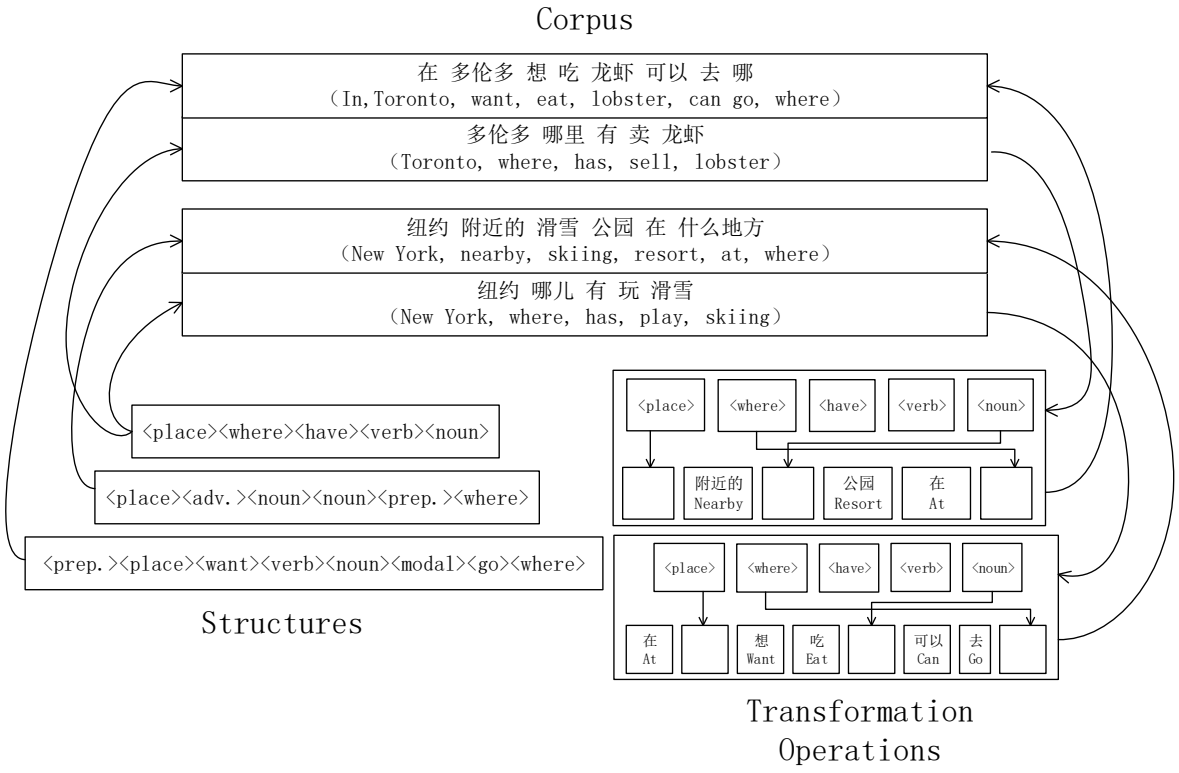

Figure 4.1: Corpus Processing Example

Input:

- C, a set of question clusters, that each cluster contains a set of paraphrase questions.
- $Tagger(q)$, a function that extracts the tag sequence of question q.
- $Transform(q_1, q_2)$, a function that generates the transformation operations from q_1 to q_2.

Output:

- $QuestionToPara$, a map from the questions to all their paraphrases in corpus.
- $TagSeqToQuestion$, a map from the tag sequences to sets of questions, that each set contains all the questions with same tag sequence.
- $QPairToOps$, a map from the question pairs to their transformation operations.

```
for each question cluster c_i ∈ C do
    for each question q_j ∈ c_i do
        t = Tagger(q_j)
        if t ∈ TagSeqToQuestion then
            TagSeqToQuestion(t) = {}
        end
        TagSeqToQuestion(t) = TagSeqToQuestion(t) ∪ q_j
        QuestionToPara(q_j) = {}
        for each question q_k ∈ qc_i that q_k ≠ q_j do
            QuestionToPara(q_j) = QuestionToPara(q_j) ∪ q_k
            QPairToOps(q_j, q_k) = Transform(q_j, q_k)
        end
    end
end
return QuestionToPara, TagSeqToQuestion, QPairToOps
```

Algorithm 1: training procedure

Paraphrase generating part is online process. Given an input question, it generates a set of ranked CFG expressions, by matching the input question to questions in corpus, and applying corresponding transformation operations. Example is like in Section 3.4. The

detailed algorithm is shown below in Algorithm 2.

Input:

- q_{input}, an input question.
- $Tagger(q)$, a function that extracts the tag sequence of question q.
- $QuestionToPara$, a map from the questions to all their paraphrases in corpus.
- $TagSeqToQuestion$, a map from the tag sequences to sets of questions, that each set contains all the questions with same tag sequence.
- $QPairToOps$, a map from the question pairs to their transformation operations.
- $Gen(q, q_s, op)$, a function that generates a paraphrase of q by applying op to q.
- $Score(q_g, q_s, q_p)$, a function that gives a generated paraphrase a score.
- $Merge(Q)$, a function that merges a set of paraphrases into a set of CFG expressions which is used as patterns in matching.

Output:

- Q_g, a set of generated paraphrases
- Sc, a map that maps generated paraphrases to their scores
- Pt, a set of CFG expressions that represents all paraphrases in Q_g

$t = Tagger(q_{input})$
$Q_s = TagSeqToQuestion(t)$
$Q_g = \{\}$
for *each* $q_{s_i} \in Q_s$ **do**
  $Q_{p_i} = QuestionToPara(q_{s_i})$
  for *for each* $q_{p_{ij}} \in Q_{p_i}$ **do**
    $op_{ij} = QPairToOps(q_{s_i}, q_{p_{ij}})$
    $q_{g_{ij}} = Gen(q_{input}, q_{s_i}, op_{ij})$
    $Sc(q_{g_{ij}}) = Score(q_{g_{ij}}, q_{s_i}, q_{p_{ij}})$
    $Q_g = Q_g \cup q_{g_{ij}}$
  end
end
$Pt = Merge(Q_g)$
return Q_g, Sc, Pt

Algorithm 2: generating procedure

In following sections, Section 4.2 will introduce how the corpus is collected, Section 4.3 will explain $Tagger(q)$, Section 4.4 will explain $Transform(q_1, q_2)$, Section 4.5 and Section 4.6 will explain $Gen(q, q_s, op)$, Section 4.7 will explain $Score(q_g, q_s, q_p)$, and Section 4.8 will explain $Merge(Q)$.

4.2 Corpus Preparation

Our parallel corpus is a set of clusters, and each cluster contains a set of paraphrase questions. Instead of collecting all kinds of sentences, we use only question sentences (including phrasal QA queries) as corpus, so that it is possible for our approach to discover the latent semantics in question sentences.

We construct such a corpus in two ways, handwritting and manual labelling.

One part of corpus is handwritten by human editors. We randomly select 260 questions as seeds from our CQA [1] [2] database, and handwrite paraphrases for each seed question (1372 paraphrases in total, including 260 seed questions). This work is costly, but it can provide many typical structure variants for the common question structures.

The other part of corpus is collected by manual annotating. As we need to collect a lot of more paraphrase questions, and the handwritten ones are far from enough, we annotate 588 clusters of paraphrasing questions (2052 paraphrases in total) from CQA.

For better efficiency in collecting the second part of corpus, we first extract questions which are possible to be paraphrases, from CQA, as candidates, then annotate the actual paraphrases among them. In order to extract possible candidates, we suppose that if two questions have some keywords in common and share very similar answers, they are possibly asking for the same thing, i.e. they are paraphrases. With this assumption, we use randomly selected entity names to search all the answers in CQA to get questions with same keywords in answers, then cluster the questions which share at least one common keyword in themself and two in their answers. Then, with such candidates extracted, we use human editors to annotate the actual paraphrases as corpus.

Furthermore, the corpus can be expanded during application, with relatively low cost. When paraphrases generated by the program, human editors can select or even modify them then add them into corpus.

[1] http://zhidao.baidu.com
[2] http://wenwen.sogou.com

4.3 Sentence Structure Extraction

This part of work converts the question sentences into sequences of tags, which represent the sentence structures in our approach. In general, the tagging process has 4 steps:

Segmentation → POS tagging → Functional tagging → Dependency parsing

For example:

Table 4.1: Structure Extraction Example

多伦多(Toronto)哪(where)能(can)找到(find)好吃的(delicious)中餐(Chinese food)?							
Segmentation	Toronto	where	can	find	delicious	Chinese food	?
POS tag	noun	pron.	verb	verb	adj.	noun	-
Functional tag	place	where	modal	-	-	-	-
Dependency	-	-	-	noun			-
Final tag	$< place > < where > < modal > < verb > < noun >$						

1. 2. Segmentation & POS tagging: While in some languages such as Chinese, the words are not naturally segmented in a sentence. Hence, when dealing with a Chinese sentence, before any other processing, we need to do segmentation of the sentence to segment a sequence of characters into a sequence of words. In our work, this part is done by existing tools. At the same time, the tools give POS tags to each segmented word. After this, some trivial trimming is done, such as removing useless punctuation, words and prefix phrases.

3. Functional tagging: After the previous steps, all the words in sentences have gotten POS tags. However, the POS tags are very general that the same tag sequence may relate to many totally different sentence structures. So we construct a dictionary of functional words which can get some words tagged with more details. Such refined tag sequence can make the sentence structure carry more information. For example, all the question words like "when", "where", and "why" should be tagged as different to each other and different to other pronouns. Some verbs such as "is", "have", "let", "can", "want" usually need to be distinguished from other notional verbs as well.

4. Dependency parsing: When each word is tagged with POS tags and functional tags, the sentence structure of question has been represented by a sequence of tags. However, many questions with same or very similar sentence structures may have different tag sequences, and the corpus is unable to cluster all questions into reasonable amount

of clusters, which means an input question may not be able to match enough questions with same structure. Considering this, we have to unify different tag sequences by simplifying them, and still keep same structure questions alignable. Thus, by observing the generated tag sequences of questions, we generalized a small set of simple rules to unify similar POS tags or combine some consecutive POS tags. Thus, some similar tags are considered as one same tag, and some adjacent tags are combined as a simple dependency tree, which can be tagged with the tag of its root node. For example, the words like "the", "a", "this", and the quantifier words, are tagged same as adjectives when they are in front of nouns, since they share similar roles in sentences. With the rules like "superlative + adjective = adjective" and "adjective + noun = noun", the phrase "最(the most) 漂亮的(beautiful) 极光(aurora)" is tagged as a noun, and "最(the most) 漂亮的(beautiful)" in it is considered as the children of "极光(aurora)". Thus, "哪里(Where) 能(can) 看到(see) 最(the most) 漂亮的(beautiful) 极光" and "哪里(Where) 能(can) 买到(buy) 玫瑰(roses)" are in same structure and alignable.

After all these processes, the sequence of tags is generated as the sentence structure of a question. Therefore, each question has a structure, and each structure are related to a bunch of questions. So we can consider a question as an instance of a structure.

With the structures extracted, the questions in corpus are not only clustered with their paraphrases, but also clustered with all the questions that have the same structure.

4.4 Sentence Transformation Inference

In our model, the paraphrases are considered to be generated from the original sentences, by applying a series of operations. Using the similar idea, some operations will be applied to the input question to generate paraphrases. To implement this process, we need to know the operations needed to convert instance of a structure to an instances of another structure. More particularly, we need a method to infer the transformation operations from one question to another.

Introduced in 3.2, the transformation from one question to its paraphrase is by operations of keeping common parts and filling them into new question "templates". When in inference, given a pair of paraphrase sentences, it searches for all the words/phrases which appear in both two sentences, and uses a keep list to record the alignment of them,

then uses an insertion list and a deletion list to record the remaining parts in two sentences, including the content-independent parts ("templates") and some potential phrasal substitutions which are not recognized.

In practice, while it is possible that two paraphrases have some different words/phrases in non-essential components, the method still tries to align as many words as possible, even they are not very similar or related, for remaining less unrecognized words that wrongly considered as parts of structure "templates".

For example, the questions "怎么(How to) 快速(quickly) 减肥(lose weight)" and "有效(efficient) 的([*auxiliary*]) 减肥(lose weight) 方法(method) 有(has) 哪些(what)" are annotated as paraphrases in corpus. If the transformation is considered as keeping "减肥(lose weight)" and filling it into "有效的(efficient) <动词(*verb*) > 方法(method) 有(has) 哪些(what)", it may lead to mistakes in other cases. If "快速(quickly)/有效(efficient)" is considered kept and aligned, the "'template" will be "<形容词(*adjective*) > 的([*auxiliary*]) <动词(*verb*) > 方法(method) 有(has) 哪些(what)", therefore the transformation is closer to our intuition.

According to the ideas above, the steps of the inference are:

1. For each word in paraphrase, find the exact same word/phrase in original sentence and align them
2. Check if there are one or more words in original sentence that are similar enough to the remaining words in paraphrase. If so, align the most similar pair. In this step, the similarity between words depends on the following factors:
   a. The edit distance between words / The number of characters in words
   b. Whether they have the same POS tag
   c. Whether they are synonyms
   d. Cosine distance between their word vectors (provided by Word2Vec [29])
3. According to the structures of the two sentences, if both two aligned words have children in dependencies and their children have same POS tag, align their children
4. Construct the insertion and deletion list with remaining unaligned words in paraphrase and original sentence.

Using this method, the transformations between paraphrases in the corpus can be inferred and stored offline.

4.5 Paraphrase Generation

The process of generating paraphrases has been introduced in Section 2. In general, this process based on using the structure of input question to collect material from corpus, and then using this material to generate the paraphrases of input question.

The collecting part has three steps:

1. Given an input question sentence q_{input}, use the method in Section 4.3 to extract its structure t.
2. Collect a set of questions Q_s, such that all questions in Q_s are in structure t.
3. For each question q_{s_i} in Q_s, collect all its paraphrases Q_{p_i} from corpus.

After Q_s and $\{Q_{p_i}\}$ have been collected as material, each pair of $(q_{s_i}, q_{p_{i_j}})$ can be used to generate one paraphrase for the input question. The general process has three steps:

1. Align q_{input} with q_{s_i}

   In this step, q_{input} needs to be aligned with q_{s_i}. As they share same structure t, the alignment is quite straight forward. For each tag in t, the corresponding word in q_{s_i} is aligned to the corresponding one in q_{input}. If the tag in t relates to a dependency tree of words, the corresponding word/tree can be aligned hierarchically.

2. Get the transformation operations op_{ij} between q_{s_i} and $q_{p_{i_j}}$

   In this step, the transformation operations op_{ij} is computed by the method in Section 4.4 and stored in $QPairToOps$, during corpus processing part. The op_{ij} contains 3 parts, a keep list, an insertion list, and a deletion list. The keep list records the alignment of kept words from q_{s_i} to $q_{p_{i_j}}$, and the insertion list records the words and their indexes in the "template" of $q_{p_{i_j}}$. These two lists will be used in the next step.

3. Apply op_{ij} on q_{input}

   With alignment from q_{s_i} to q_{input}, the transformation op_{ij} between q_{s_i} and $q_{p_{i_j}}$ can be applied on q_{input}, to generate its paraphrase. The algorithm is shown below in Algorithm 3.

Input:

- q_{input}, an input question.
- q_{s_i}, a question in corpus that has the same structure of q_{input}.
- $q_{p_{i_j}}$, a question in corpus which is a paraphrase of q_{s_i}.
- $Alignment$, a map from the indexes in q_{s_i} to the indexes in q_{input}.
- $Keep$, a map from the indexes in $q_{p_{i_j}}$ to the indexes in q_{s_i}, for all words kept from q_{s_i} to $q_{p_{i_j}}$.
- $Insertion$, a map from the indexes in $q_{p_{i_j}}$ to the corresponding words which are in the "template" of $q_{p_{i_j}}$.

Output:

- $q_{g_{i_j}}$, the generated paraphrase

$q_{g_{i_j}}$ = new $ArrayList(len(q_{p_{i_j}}))$
for *each idx in* $Keep.keys()$ **do**
  $q_{g_{i_j}}[idx] = q_{input}[Alignment(Keep(idx))]$
end
for *each idx in* $Insertion.keys()$ **do**
  $q_{g_{i_j}}[idx] = Insertion(idx)$
end
return $q_{g_{i_j}}$

Algorithm 3: Transformation operations application

Because of the limited corpus, sometimes there is no question in the corpus that matches input question in structure. In such cases, we temporarily crop the input question to its longest substring ($q_{input}\prime = q_{input}[i_1, i_2]$) that can get matched in structure, for generating paraphrases, and recover the results ($Q_g\prime$) back to the complete form ($q_{g_{i_j}} = q_{input}[0, i_1] + q_{g_{i_j}}\prime + q_{input}[i_2, len(q_{input})]$). For example, the input sentence "在(In) 北京(Beijing) 什么(what) 地方(place) 有(has) 卖(sell) 河豚(pufferfish)" matches to nothing in corpus, but "什么(What) 地方(place) 有(has) 卖(sell) 河豚(pufferfish)" can match some sentences with same structure in corpus and generate sentences like "我(I) 去(go) 哪(where) 能(can) 吃到(eat) 河豚(pufferfish)". Thus, it completes "我(I) 去(go) 哪(where) 能(can) 吃到(eat)

河豚(pufferfish)" to "在(In) 北京(Beijing)我(I) 去(go) 哪(where) 能(can) 吃到(eat) 河豚(pufferfish)", as a paraphrase to the original input.

4.6 Paraphrase Representation

In the paraphrase generated, each word contains both the word content itself, and its tags, including POS tag and functional tag. If all this information are provided as the final result, the paraphrase can be very confusing. Conversely, if it only keeps the word contents, there may be some incorrect words involved, and the paraphrase can also be confusing. So in order to make it readable, we need to represent each word with one thing, either the word content, or one tag. The principle of the representation is, if the word is certainly reasonable being in the sentence, represent it with itself, if not, represent it with its most detailed tag.

For example, for the input "怎么(How to) 制作(make) 钥匙(keys)", it matches to " 怎么(How to) 煮(cook) 红烧肉(pork)", with paraphrase "红烧肉(pork) 的([*auxiliary*]) 烹调(cooking) 方法(method)". Then the generated paraphrase will be "钥匙(keys) 的([*auxiliary*]) 烹调(cooking) 方法(method)". Apparently, we cannot "cook keys", instead, the correct one should be "make/produce/manufacture keys". Our approach is not able to learn the match between "make/produce/manufacture" and "keys". In this kind of cases, we represent "cooking" as just a "verb", so that the generated paraphrase will look like "钥匙(Keys) 的([*auxiliary*]) <动词(*verb*) > 方法(method)". Though it is not an accurate result, at least it gets less confusing to the editors who can easily fill "制作(make/produce/manufacture)" into the slot "<动词(*verb*) >"

To implement this principle, the particular rules are:

1. For the following kinds of words, represent it with itself:
   i. The words copied from input sentence
   ii. The words with some certain POS tags
   iii. The commonly appearing words
   iv. The words which are very similar to one of the word in q_{input}
   v. The words which have very low similarities to any of the words in q_{s_i}
2. For all remaining words with functional tags, represent with their functional tags.
3. For all remaining words, represent with their POS tags.

4.7 Paraphrase Ranking

A lot of paraphrases can be generated from the previously mentioned procedures in Section 4.5, but usually not all of them are correct. For here, the correctness means the paraphrase is a question with same semantics to the original question and is a reasonable question sentence in natural language which can be potentially ask by human users. In this case, we rank all the generated paraphrases by their probabilities of being correct.

As we mentioned in Section 3.4, the semantics similarity between the words in input question and the words associated to the variation operations influences the probability of the input question being paraphrased by such operations. E.g., "多伦多(Toronto) 哪里(where) 有(has) 卖(sell) 龙虾(lobster)" may not able to apply the transformation operations that from "纽约(New York) 哪里(where) 有(has) 玩(play) 滑雪(skiing)" to its paraphrases, for generating paraphrase, as these two sentences are not that similar in semantics. Specifically, we assume that more similar the question q_{input} and q_{s_i} are, more likely the generated paraphrase $q_{g_{i_j}}$ is correct. Therefore, we give similarity scores to all the generated paraphrases. The similarity score is calculated by following formula.

$$\begin{aligned} score_{similarity}(q_{g_{i_j}}) &= sentence\_similarity(q_{input}, q_{s_i}) \\ &= \frac{\sum_{i=0}^{n} weight(P_i)T_i}{\sum_{i=0}^{n} weight(P_i)} \end{aligned} \tag{4.1}$$

In above formula,

n = the number of tags in structure of q_{input}

$w(P_i)$ = the weight of i^{th} POS tag in structure of q_{input}

For word similarity calculation,

T_i = the similarity of the i^{th} pair of alignment between q_{input} and q_{s_i}.

When the ${i_0}^{th}$ to ${i_{k_1}}^{th}$ words in q_{input} are aligned to the ${j_0}^{th}$ to ${j_{k_2}}^{th}$ words in q_{s_i} (happens in aligned dependency trees),

$T_i = \max_{i \in [i_0, i_{k_1}], j \in [j_0, j_{k_2}]} word\_similarity(q_{input}[i], q_{s_i}[j])$

In above formula,

$$word\_similarity(w_A, w_b) = \begin{cases} 1 & w_A = w_B \\ 0.9 & w_A \text{ and } w_B \text{ are synonyms} \\ \frac{1}{1+6^{3-6 \cdot w_v(w_A, w_B)}} & \text{otherwise} \end{cases} \quad (4.2)$$

$$w_v(w_A, w_B) = \text{the cosine similarity between } w_A \text{ and } w_B \text{ according to Word2Vec} \quad (4.3)$$

Also mentioned in Section 4.4, in the transformations, the words in insertion or deletion lists may be semantic-irrelevant words, or part of "templates", or unrecognized substitutions. So these words may affect the reasonableness of generated paraphrases. Specifically, if the newly inserted words in generated paraphrase $q_{g_{i_j}}$ are reasonable to be in the sentence, they should be either semantically-irrelevant words, or part of "templates", or have enough relevance with input sentence. In the first two cases, such words should have very low relevance to the question q_{s_i}, and in the third case, these words should have high relevance to the input sentence q_{input}.

For example, the generated question "北京(Beijing) 的([*auxiliary*]) 寿司(sushi) 公园(resort) 有(has) 哪些(what)" in Section 3.4 is a negative example. The word "公园(resort)" is newly inserted during generation, but it has relatively high relevance to the word "滑雪(skiing)" in q_{s_i}, and it has low relevance with any word in q_{input}, so we consider it as a word that highly related to q_{s_i} and $q_{p_{i_j}}$, which makes the generated question a bad result. Thus, we use following formula to calculate the reasonableness scores.

$$score_{reasonableness} = \max\left(\frac{Bag\_of\_words\_similarity(T_{q_{input}}, T_{q_{g_{i_j}}})}{Bag\_of\_words\_similarity(T_{q_{s_i}}, T_{q_{g_{i_j}}})}, 1\right) \quad (4.4)$$

$$T_{q_{input}} = \text{All words in } q_{input} \text{ which are not copied to } q_{g_{i_j}} \quad (4.5)$$

$$T_{q_{s_i}} = \text{All words in } q_{s_i} \text{ which are not copied to } q_{p_{i_j}} \quad (4.6)$$

$$T_{q_{g_{i_j}}} = \text{All words in } q_{g_{i_j}} \text{ which are not copied from } q_{input} \quad (4.7)$$

$$Bag\_of\_words\_similarity(T_1, T_2) = w_v(\textstyle\sum \text{all words in } T_1, \sum \text{all words in } T_2) \quad (4.8)$$

Another way to calculate the reasonableness of a sentence is N-gram language model [11]. Such model usually can be used to check whether a sequence of words is a reasonable sentence, but in our approach, it is more important to calculate how reasonable the paraphrase is generated from the input, instead of how reasonable the paraphrase itself is. So mining the information from generation procedures should be a more accurate solution to this.

In addition, we give a confidence score for each paraphrase, which implies how certainly the paraphrase is generated. The results like "<地点($place$) > 的([$auxiliary$]) <名词($noun$) > <动词($verb$) > 哪些(what)" will be too general and confusing to the pattern editors to use. In our method, the words copied from input sentence mean more certainty than the newly inserted one, and the words represented as words themselves or functional tags mean more certainty than the ones represented as POS tags. So the formula for confidence score is:

$$score_{confidence} = \frac{\sum_{i=0}^{n} weight(P_i)R(w_i)}{\sum_{i=0}^{n} weight(P_i)} \tag{4.9}$$

$$n = \text{number of words in } q_{g_{i_j}} \tag{4.10}$$

$$weight(P_i) = \text{the weight of } i^{th} \text{ POS tag in } q_{g_{i_j}} \tag{4.11}$$

$$R(w_i) = \begin{cases} 1 & \text{if } w_i \text{ is copied from } q_{input} \\ 0.8 & \text{if } w_i \text{ is not from } q_{input} \text{ but represented as itself} \\ 0.9 & \text{if } w_i \text{ is represented as its functional tag} \\ 0.5 & \text{if } w_i \text{ is represented as its POS tag} \end{cases} \tag{4.12}$$

Finally, the similarity score, reasonableness score and confidence score are generalized as one score, by the formula below, for ranking all the generated paraphrases.

$$score_{overall} = score_{similarity} \cdot \sqrt{score_{reasonableness}} \cdot \sqrt{score_{confidence}} \tag{4.13}$$

In our experiment, the paraphrases with negative scores are removed.

4.8 Pattern Merge

Although ranked and filtered, the amount of generated paraphrases still have potential to get reduced for better readability. In the pattern expressions in CFG that we use for the pattern-based match, each expression can contain multiple patterns. Like the example in Chapter 2, the expression "What's|(What is) [the] weather [like] [< $time$ >, < $location$ >]" contains 32 kinds of questions, such as "What's the weather like < $time$ >< $location$ >", "What is the weather < $location$ >< $time$ >", etc. Thus we can merge the generated paraphrases into pattern expressions, to improve the readability.

First, all the identically duplicated paraphrases are removed. Then for each paraphrase, we consider it as a pattern expression that contains a sequence of words, both essential

and "optional" ones. So we use some manually defined rules to recognize for the words that are optional to appear in any kinds of question sentences, and label these words as optional in expressions. For example, if we generate a paraphrase "what is the weather like in Waterloo", the words "the" and "like" are considered as optional, thus the pattern can be like "what is [the] weather [like] in Waterloo".

To merge the pattern expressions, we compare the expressions thought the generated ranked list of patterns. If one expression can transform to another, by only adding or deleting or substituting one word or a few consecutive words, they can be merged into one expression. For example, the paraphrases "What is the population of Beijing" and "What is the population of the city of Beijing" can be merge into "What is the population of [the city of] Beijing". With this rule, the correctness of expression is not decreased because no other irrelevant paraphrase is concluded in the merged expression.

Considering that too complex expressions are very confusing to read, we set a limitation that at most two paraphrases can merge as one expression.

Chapter 5

Evaluation

There are no standard methods to do evaluation for paraphrase generation. Most of related existing works are evaluated manually, so is our work. In the field of Chinese QA, there is no similar system that generates paraphrases to improve the coverage of question analysis module. So instead of comparing with other approaches, we focus more on evaluating how much assistance our method can provide to the pattern writing in question analysis.

Preparing for the experiment, we collect 3424 question sentences as corpus, which are in 848 clusters of paraphrases, and each cluster has 4.038 questions in average. After preprocessing the corpus, we cluster all questions in 2082 kinds of structures, and each kind of structures has 1.645 question instances in average. Figure 5.1 and Figure 5.2 show the distribution of structures and their corresponding paraphrases.

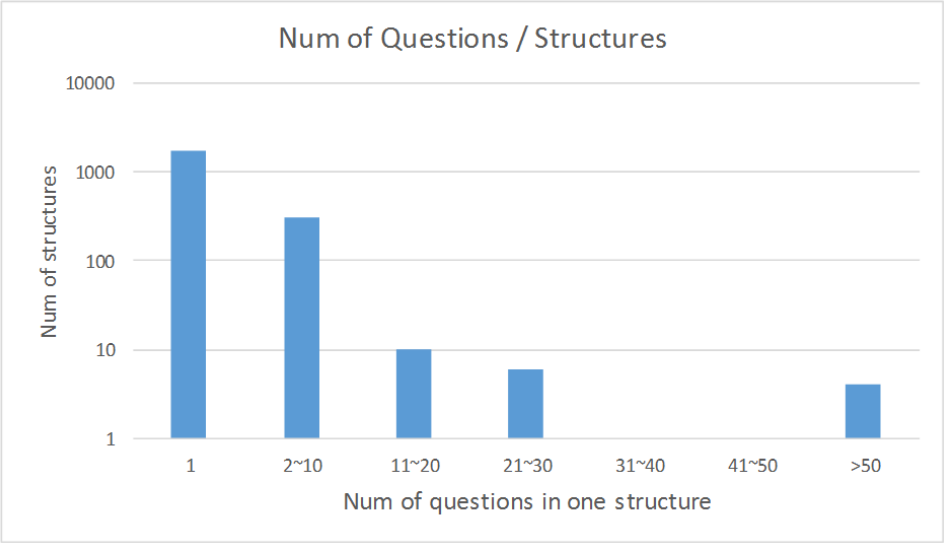

Figure 5.1: Number of Questions per Structure

Figure 5.2: Cluster Size

Our test cases are 150 question sentences randomly extracted from our CQA database. The statistics of test cases are shown below in Figure 5.3, Table 5.1 and Table 5.2.

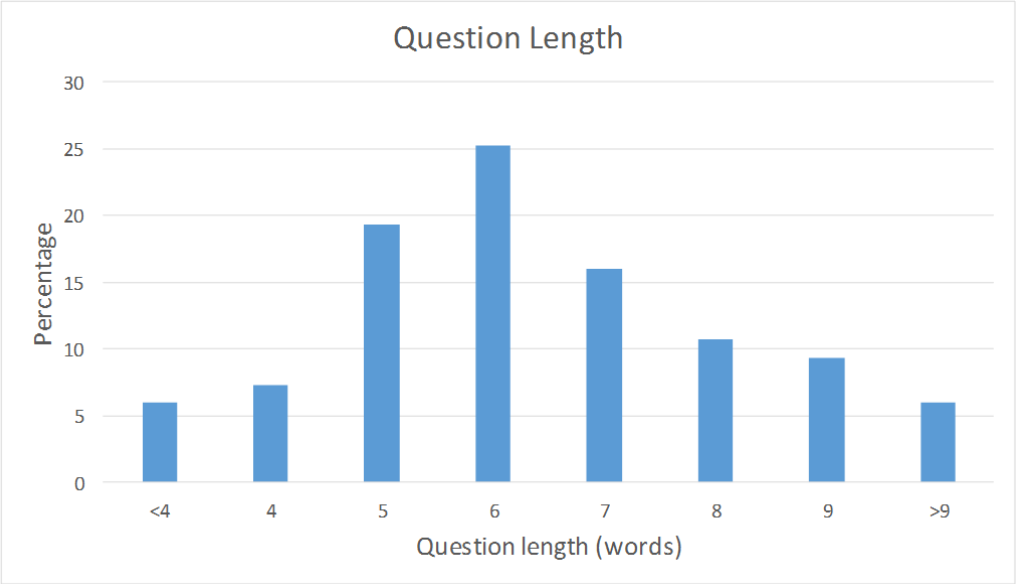

Figure 5.3: Lengths of Testcases

Table 5.1: Types of Test Questions

Category	Phrase	Yes-no questions	WH-questions
Percentage	7.3	13.3	79.3

Table 5.2: Simple Questions / Complicate Questions

Category	Simple	Complicate
Percentage	21.3	78.7

The evaluation has done in five main aspects.

1. The correctness (Precision) of generated paraphrases
2. The coverage (Recall) of generated paraphrases

3. The effectiveness of ranking algorithm
4. The performance of generation process
5. How does the scale of the corpus make the influence

5.1 Precision

As only the correct paraphrases can help in the question analysis, we need to evaluate the correctness of the generated paraphrases.

For the requirement of our application, the generated paraphrases are used in pattern matching. A possible question with incorrect grammar should still be able to get matched in some cases, so covering this kind of cases is helpful to expand the coverage of question analysis. On the other side, if sentence is apparently impossible to be a question, it won't be matched in any case, which is useless and confusing to the pattern editors, but barely harmful to the performance of question analysis. For the worst, the sentence with different semantics might wrongly match another different question, which is harmful and can lead to mistakes in question analysis.

Inspired by this, all the generated paraphrases can be classified as 3 types:

Type 1. Sentence that has same semantics to the input, and can be a possible question, even it is not in proper grammar

Type 2. Sentence that is not grammatically correct and is not a possible question.

Type 3. Sentence that has reasonable but different semantics to the input

For example, for the input "正确(Correctly) 使用(use) 电动(electronic) 牙刷(toothblush) 的([*auxiliary*]) 方法(method)",

- "怎样(How to) 正确(correctly) 使用(use) 电动(electronic) 牙刷(toothblush)" - Correct
- "正确(Correctly) 使用(use) 电动(electronic) 牙刷(toothblush) 的([*auxiliary*]) 步骤(procedures) 都有(have) 什么(what)" - Correct
- "在(At) 电动(electronic) 牙刷(toothblush) 方法(method) 如何(how)" - Useless

- “电动(Electronic) 牙刷(toothblush) 的([*auxiliary*]) 好处(benefit) 都有(have) 什么(what)” - Wrong

In addition, as we mentioned in Section 4.6, the words/phrases in generated paraphrases sometimes are represented as functional tags or POS tags. Such tags need to be interpreted to specific words/phrases, by human editors, in practical application. So if the tags in a generated paraphrase can be interpreted to some words/phrases which make the paraphrase sentence “correct”, the paraphrases will be labelled as “correct”.

In our approach, the generated paraphrases are concluded as expressions in CFG. So the correctness of these expressions can be evaluated as 3 levels:

1. The expression is correct, that means, at least one of the possible paraphrases contained in the expression are of Type 1, and none of them is of Type 3.
2. The expression is useless, that means, all the possible paraphrases contained in the expression are of Type 2
3. The expression is wrong, that means, one or more of the possible paraphrases contained in the expression are of Type 3

The results of correctness evaluation show as in Table 5.3 and Table 5.4.

Table 5.3: Paraphrases Numbers

Number of paraphrases generated	0	$1 \sim 5$	$6 \sim 10$	$11 \sim 20$	$21 \sim 30$	> 30
Percentage of test cases	17.3	26	12.7	26	8.7	9.3

Table 5.4: Precision

	Avg. Paraphrases	Avg. Patterns	Correct%	Useless%	Wrong%
Overall	17.97	14.67	51.79	37.07	11.13
For top 5 patterns	-	-	70.42	22.32	7.26

5.2 Recall

Except for the correctness, we also want the generated paraphrase to cover as many of potential paraphrases as possible. Thus we define the recall of our approach as:

$$recall = \frac{|\text{potential questions asked by human users} \cap \text{generated paraphrases}|}{|\text{potential questions asked by human users}|} \tag{5.1}$$

It is impossible to enumerate all the potential paraphrases and even the number of them is unknown. In order to evaluate the recall, we approximate it by using handwrite paraphrases as the samples of all potential paraphrases. Thus the definition of recall has been modified as:

$$recall = \frac{|\text{handwritten paraphrases} \cap \text{generated paraphrases}|}{|\text{handwritten paraphrases}|} \tag{5.2}$$

In the experiment, the number of human written paraphrases of each test case is not fixed, as for some questions, its not easy for human to come up many paraphrases. The results of coverage evaluation show as below in Table 5.5.

Table 5.5: Recall

Avg. paraphrases by human	Avg. correct paraphrases by automatic generation	Recall (%)	
		Overall	For top 5 Patterns
2.66	7.60	22.8	15.5

When the CFG expressions are used in pattern-based matching, with custom built synonym dictionary, most of words can match their synonyms. So this can improve the recall in actual use.

For example, for the input "矫正(Straightening) 牙齿(teeth) 要(need) 多少钱(how much)", there is a paraphrase "矫正(Straightening) 牙齿(teeth) 的([*auxiliary*]) 价位(price level) 是(is) 多少(what)" written by human. If our system has generated a paraphrases as "矫正(Straightening) 牙齿(teeth) 的([*auxiliary*]) 价格(price) 是(is) 多少(what)", and "价格(price)"/"价位(price level)" are defined as synonyms in dictionary, this generated paraphrase can match the human written one.

In addition, there are some pre-processing in our pattern-based matching, so that the differences in some words/phrases in questions can be ignored.

Re-evaluated with the synonyms and the pre-processing, the results are shown in Table 5.6.

Table 5.6: Recall with Synonyms

Avg. paraphrases by human	Avg. correct paraphrases by automatic generation	Recall (%), with synonyms and pre-processing	
		Overall	For top 5 Patterns
2.66	7.60	35.6	24.6

In order to compare to human editors, we also evaluated what coverage human editors can have. We invited 11 editors to handwrite paraphrases for half of our testcases (time limit is 5 min per one testcase), and see how many of handwritten paraphrases we used above for recall evaluation can be covered by these human editors. For this evaluation, the formula of recall is:

$$recall = \frac{|\text{handwritten example paraphrases} \cap \text{paraphrases written by 11 human editors}|}{|\text{handwritten example paraphrases}|} \tag{5.3}$$

This coverage can be considered as the baseline of recall. Table 5.7 shows the results.

Table 5.7: Recall by Human

Avg. paraphrases by human in above	Avg. paraphrases by 11 human editors	Recall (%)	
		Original	With synonyms and preprocessing
3.08	4.28	6.9	19.0

5.3 Scoring

In Section 4.7, we mentioned that the generated paraphrases are given scores and ranked, to approximately represent their different probabilities to be a proper paraphrase to the input. In Section 5.1, we evaluated the precision of the generated paraphrases by labelling them as "correct", "useless", and "wrong". Ideally, if the paraphrases are scored correctly, the paraphrases with high scores are more likely to be "correct" ones, in opposite, the "useless" and "wrong" ones should have relatively lower scores.

The relation between paraphrases scores and their precision labels are shown in Figure 5.4.

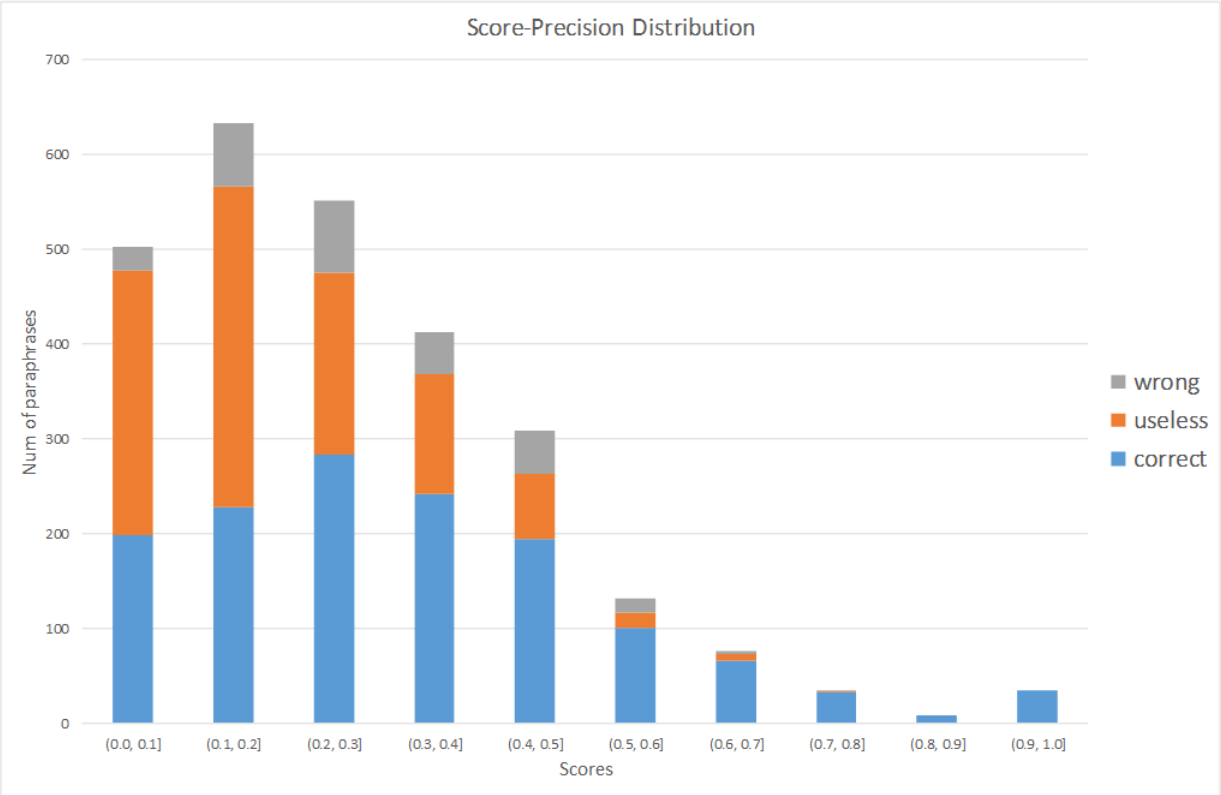

Figure 5.4: Precision-Score

5.4 Performance

On average, each test case takes 4.9031 seconds to generate all the paraphrases in our current implementation. Figure 5.5 and Figure 5.6 below show the stats for the running time of our implementation.

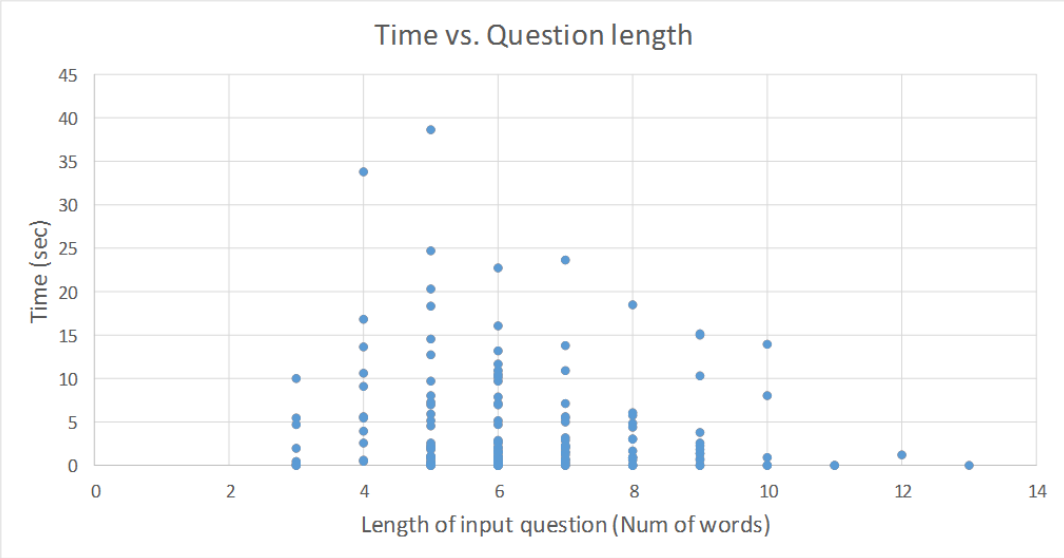

Figure 5.5: Time / Question Length

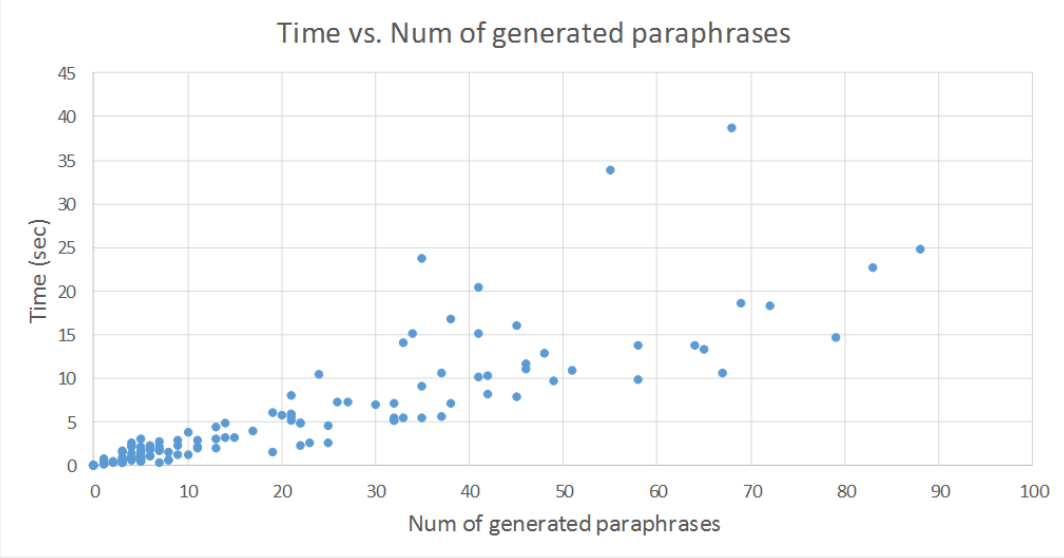

Figure 5.6: Time / Number of Generated Paraphrases

In another aspect, the automatically generated patterns sometimes contain the tag representations. As such tags have to be interpreted by manual work, too many tags in one pattern will be both inefficient and confusing to human editors. For current approach, each pattern has 0.669 tag representation, and for the top 5 patterns, only 0.510 tag representation in each of them.

5.5 Scaling

As mentioned in Section 4.2, the corpus can be expanded during application, by selecting (with or without modification) the generated paraphrases as corpus. In such expansion, the total amount of questions in corpus will be increased, but it won't bring new sentence structure into corpus (when the transformation operation from q_s to q_p applied on q_{input}, the structure of generated paraphrase q_g is same as the one of q_p, that no new structure generated), so the amount of unique structures in corpus remains the same. Thus, we need to evaluate how the expansion of corpus improves the precision and recall. Without long-term application, its very hard to get enough expansion. So we scale down the initial corpus with the total structure amount unchanged, in order to simulate different scales of corpus.

Table 5.8 below shows the results.

Table 5.8: Scaling Influence

Corpus Scale	Avg. Patterns	Overall(%)		Top 5 patterns(%)		Avg. Time
		Correct	Recall	Correct	Recall	
60%	2.6	60.7	9.3	64.3	7.3	0.9
80%	2.9	59.6	10.8	64.8	9.0	0.9
100%	14.8	51.8	35.6	70.4	24.6	4.9

Chapter 6

Discussion

6.1 Precision

The main purpose of this work is to assist human editors building up the pattern-based matching system more efficiently, so in the actual application, all the generated paraphrases/patterns will be reviewed and the human editors suppose to pick the correct ones and discard the useless or wrong ones. In this case, there is not a very high requirement for the precision of generated patterns. Instead, some useless patterns are acceptable as long as there are not too many useless or wrong patterns which confuse the editors.

The evaluation results shows that more than half of generated patterns are correct, which means our method can generate more than seven ($14.67 \times 51.79\% = 7.60$) correct patterns per input, on average. And about half ($\frac{5 \times 70.42\%}{14.67 \times 51.79\%} = 46.34\%$) of these seven patterns are ranked in top five. Thus, we consider such results helpful to the editors with not too much confusion.

Reviewing those generated paraphrases which are annotated as "useless" or "wrong", we find 5 main reasons that lead to those "mistakes".

1. Error in POS tagging / dependency parsing:

   For example, while the input

   "这样(this) 的([*auxiliary*]) 配置(configuration) 能(can) 玩(play) 什么(what) 单机(PC) 游戏(games)"

   should generate a paraphrase like

"有(Has) 什么(what) 单机(PC) 游戏(games) 适合(suitable) 在(in) 这样(this) 的([*auxiliary*]) 配置(configuration) 玩(play)",

instead, the actual generated one is

"这样(this) 的([*auxiliary*]) 有(has) 什么(what) 单机(PC) 游戏(games) 适合(suitable) 在(in) 配置(configuration) 玩(play)".

The mistake happens in the dependency parsing on "这样(this) 的([*auxiliary*]) 配置(configuration)". In this phrase, "这样(this) 的([*auxiliary*])" describes "配置(configuration)", that the phrase should have been considered as a whole.

2. Tag sequence (without enough correct dependency information) does not equal to sentence structure in all time, that sometimes two sentences with same tag sequence have totally different sentence structures. In such cases, though q_s has same tag sequence as q_{input}, it should not be extracted and aligned. This kind of situation happens a lot:

   For example, the input

   "减肥(Lose weight) 最(most) 有效(effective) 的([*auxiliary*]) 方法(method)"

   matches to q_s

   "增强(Improve) 记忆力(memory)",

   since they share same tag sequence of "$< verb > < noun >$" ("最(most) 有效(effective) 的([*auxiliary*]) 方法(method)" is considered as a whole noun phrase that aligns to "记忆力(memory)").

   Apparently, they are in totally different structure, that '记忆力(memory)" is the object of " 增强(Improve)", but "减肥(Lose weight) 最(most) 有效(effective) 的([*auxiliary*]) 方法(method)" means "the most effective method to lose weight". So in this case, they should not be matched and aligned.

3. Though two sentences have same sentence structures, the transformation operations which work for one shouldn't be applied to the other one. This is mainly because some transformation operations are more related to the specific word semantics, instead of sentence structure. By our observation, this is the major reason of generating "useless" or "wrong" paraphrases. Our ranking algorithm suppose to prevent these cases from appearing in top ranks, but sometime the word similarity calculation tools used in ranking are just not that precise.

   For example, the input

   "矫正(Straightening) 牙齿(teeth) 的([*auxiliary*]) 价格(price) 是(is) 多少(what)"

matches to q_s

"蚂蚁(Ant) 的([$auxiliary$]) 腿(leg) 的([$auxiliary$]) 数目(amount) 是(is) 多少(what)".

With paraphrase $q_{p_{i_j}}$

"蚂蚁(Ant) 一般(usually) 有(has) 几个(how many) 腿(legs)",

it generates

"矫正(Straightening) 一般(usually) 有(has) 几个(how many) 牙齿(teeth)"

This is obviously an incorrect result. Between q_s and $q_{p_{i_j}}$, the transformation from "腿(leg) 的([$auxiliary$]) 数目(amount) 是(is) 多少(what)" to "有(has) 几个(how many) 腿(legs)" is related to both the word " 数目(amount)" and the sentence structure. So such transformation should not be applied on q_{input}, since q_{input} does not have same topic (asking amount of something) with q_s.

4. In representation, due to the incorrectness of the word similarity calculation tools or other reasons, the words are not represented properly:

   For example, the input

   "怎么(How to) 查找(look up) 公务员(civil servant) 考试(exam) 的([$auxiliary$]) 信息(information)"

   with a $q_{p_{i_j}}$

   "记忆力(memory) 不好(not good) 怎样(how) 能(can) 增强(improve)"

   generates a paraphrase

   "公务员(civil servant) 考试(exam) 的([$auxiliary$]) 信息(information) 不好(not good) 怎样(how) 能(can) 查找(look up)"

   The result does not make sense unless the word "不好(not good)" changes to something like "不清楚(unknown)". In this case, our approach is not expected to generate words like " 不清楚(unknown)", but representing the word as " 不好(not good)" is still wrong here. Instead, there should be represented as "<negative adjective>".

5. When a same-structure question (q_s) cannot be found for some questions in corpus, their substring are extracted for paraphrase generation, but the paraphrases of their substring are not correct when get completed in full. This also happens a lot:

   For example, for the input

   "减肥茶(Dietic tea) 要(need) 怎样(how to) 搭配(go with) 最(most) 合理(reasonable)",

its substring

“减肥茶(Dietic tea) 要(need) 怎样(how to) 搭配(go with)”

matches to q_s

“红烧肉(Pork) 要(need) 怎么(how to) 做(cook)”.

Then it generates

“减肥茶(Dietic tea) 的([*auxiliary*]) $<$ *noun* $>$”. (Phrases like “搭配方法(mix method)” could be filled in $<$ *noun* $>$)

which can be completed as

“减肥茶(Dietic tea) 的([*auxiliary*]) $<$ *noun* $>$ 最(most) 合理(reasonable)”

The result is an “useless” one. The paraphrasing between “ 减肥茶(Dietic tea) 要(need) 怎样(how to) 搭配(go with)” and “ 减肥茶(Dietic tea) 的([*auxiliary*]) $<$ *noun* $>$” is correct, but when in whole sentence of “减肥茶(Dietic tea) 要(need) 怎样(how to) 搭配(go with) 最(most) 合理(reasonable)”, the such paraphrasing can not lead to a correct result.

Besides, the evaluation shows that for 17.3% of inputs, it generates nothing, and for 9.3% of inputs, the amount of generated patterns might be too large for human editors to use.

By analyzing the test cases, we find that for all those 17.3% of inputs which generate nothing, they cannot be matched to any same-structure question in corpus. There are three main reasons causing that.

1. Error in POS tagging:

   For example, the input

   “一道(one) 高中(high-school) 物理(physics) 题(problem)”

   which cannot matches any q_s from corpus, is tagged as:

   “$<$ *adverb* $>$ $<$ *noun* $>$”

   which is wrong. If “$<$ *adverb* $>$” can be correctly tagged as “$<$ $quantifier$ $>$”, it should be able to correctly match a q_s

   “一些(some) 好的(good) 国产(domestic) MV(MV)”.

2. Some structures are similar and should be alignable, but in our implementation they are considered as "different":

   For example, for the input

   "吹(Dry out) 头发(hair) 前(before) 可以(can) 化妆(make up)",

   there is a question in corpus that has a similar but different structure

   "月子(Right after giving birth) 里(in) 可以(can) 吃(eat) 韭菜(leek)".

   They should be able to align, but in our evaluation, the input cannot find any q_s since there is no "same" structure appeared in corpus.

3. There is not any similar structure in corpus:

   For example, for the input

   "法律(Law) 和(and) 会计(accounting) 我(I) 应该(should) 选(choose) 那一个(which)",

   the most similar structure appeared in corpus is

   "地震(Earthquake) 时(during) 我们(we) 应该(should) 做(do) 些(some) 什么(what)"

   which is totally different.

   For these cases, there just has nothing as q_s in corpus.

From these observations, there is still considerable work to do to improve the precision in the future. With our current approach, there are three aspects of potential improvements:

1. Enlarge the scale of corpus. More questions in corpus can cover more structures of input questions, and generate more paraphrases. With more possible paraphrases generated, we can rise the scoring threshold to get higher precision, with maintaining a reasonable amount of generated patterns.

2. Use better tools for segmentation, POS tagging and word similarity calculation, to reduce the mistakes that caused by these tools.

3. Improve the dependency parsing. The dependency parsing in our current implementation is quite sketchy, with only some simple combination rules. Such dependency parsing has caused a lot of mistakes. If we can do better of this, or use other better methods to represent and align the structures of sentences, the precision could get improved greatly.

Still, the major problem in current approach won't be solved by the methods above. As we find that the paraphrasing is related to both sentence structure and word semantics, a machine learning model should be useful to learn such relations and provide higher precision. In learning model, the current inferred transformation operations constraint the diversity of generated paraphrases as it considers the sentence structure as a whole, instead of more fine-grained sub-patterns. For example, the paraphrases pair

"红烧肉(Pork) 怎么(how to) 做(cook)"

"请(Please) 告诉(tell) 我(me) 红烧肉(pork) 的([*auxiliary*]) 做法(recipe)"

has contained two kinds of information:

1. $< food >$ 怎么(how to) 做(cook) $\Leftrightarrow$ $< food >$ 的([*auxiliary*]) 做法(recipe)
2. 请(Please) 告诉(tell) 我(me) <a question sentence> $\Leftrightarrow$ <a question sentence>

Such information isn't reflected in our current transformation operations, and current definition of sentence structure does not support this kind of information. If we can refine the transformation operations as separate ones like above, and represent the sentence structures in fine-grained sub-patterns, more diverse paraphrases should be generated.

6.2 Recall

Since the generated patterns are used to cover the natural language questions, it is more important to evaluate how much of questions that potentially asked by human, can be covered by the generated patterns. So the recall evaluation mainly reflects the improvement of coverage brought by our work.

The evaluation results show that the generated paraphrases can cover 22.8%/35.6% (without/with synonyms or preprocessing) of our example questions. This is not a very high coverage. But with the baseline comparison evaluation done by human editors, we find it even a much harder task for human, that for each testcase, human editor can generate only 4.28 paraphrases in 5 minutes on average, and these paraphrase can cover only 6.9%/19.0% (without/with synonyms or preprocessing) of human written examples. So at least our system can outperform and assist the human editors by giving a much better coverage.

In average there are more than seven correct patterns generated per input, but these patterns can only cover a very small part of human written questions. Several reasons cause this.

1. Many correctly generated patterns are in similar structure. For example,

   - "哪个(Which) 品牌(brand) 笔记本电脑(laptop) 比较好(better)"
   - "哪个(Which) 品牌(brand) 的([*auxiliary*]) 笔记本电脑(laptop) 比较好(better)"
   - "什么(What) 品牌(brand) 的([*auxiliary*]) 笔记本电脑(laptop) 好(good)"
   - "哪个(Which) 品牌(brand) 的([*auxiliary*]) 笔记本电脑(laptop) 好(good)"

   are four generated paraphrases for one input, but they are so similar, even with same structure. So, in fact, there are less than seven "unique" structures covered by those seven correct patterns.

2. In many cases, a long phrase is considered a whole, which stay unchanged in all generated paraphrases. This restricts the diversity, and coverage of generated results. For example, in

   - "有(Has) 坏道(bad track) 的([*auxiliary*]) 硬盘(hard disk) 修复(recover) 后(after) 数据(data) 还(still) 在(be there)"
   - "阜阳(Fuyang) 的([*auxiliary*]) 宽带(broadband) 资费(plan) 标准(standard)"

   the phrases "有(Has) 坏道(bad track) 的([*auxiliary*]) 硬盘(hard disk)" and "宽带(broadband) 资费(plan) 标准(standard)" are all considered as unchanged wholes.

3. For more common cases, the low recall is just simply because none of the generated paraphrase is of same structure with human written examples, which means there is no question that both covers such structure, and paraphrases with any q_s, in corpus.

The top 5 ranked patterns has a recall of 15.5%, that first half of correct patterns provide about two-third ($\frac{15.5\%}{22.8\%} = 67.98\%$) of coverage. It means that high ranked patterns not only have better precision but also are more likely to be useful in covering.

With the synonyms involved, the recall has improved to 35.6%. This improvement (22.8% $\rightarrow$ 35.6%) indicates that the phrasal substitution provides relatively less help in coverage, compare to structure variations.

To generate more diverse paraphrases which provide better coverage to potential user questions and improve the recall, obtaining larger corpus should be the main task. Besides, currently many generated patterns are similar to each other, so merging such cases can also make the results less confusing. Also, the idea mentioned in Section 6.1 which based on machine learning should be helpful in generating more variations in paraphrases, as well as solving the problem that some long phrases stay unchanged.

6.3 Scoring

From the results, we can see that the score of each paraphrase works well for ranking. Basically, paraphrases with higher scores do have higher precision, so such correlation between scores and precision does make the correct paraphrases appear in higher ranks. But as the scores distribution shows, the majority of generated paraphrases are with relatively low score, where the correlation between score and precision is not that significant. Thus, with this scoring strategy, the high-ranked patterns are usually correct with better confidence, while the remaining parts might not be properly ranked.

Currently, the parameters in formulas that we used for scoring are estimated based on observation. In the future, involving some machine learning methods like Learning to Rank [26] may help giving more reasonable scores for ranking.

6.4 Performance

On average, the generation of one input question takes nearly 5 seconds, which is quite acceptable as an offline assistance for human editors. For the cases which can generate larger amount of paraphrases, the running time increases linearly.

6.5 Scaling

The scaling experiment has evaluated the precision, recall, and performance with reduced corpus. By evaluation results, we can estimate how the precision, recall and performance will change after the corpus accumulates during the application.

With the accumulation of corpus, the average amount of generated paraphrases increases greatly, and such increase helps to produce better coverage (with paraphrases amount increases by 5.7 times, the recall increases by 3.8 times). As for the precision, larger the scale of corpus is, more useless and wrong paraphrases are generated, that the precision is slightly decreased by the increase of corpus. When with more corpus and more generated paraphrases, our ranking strategy works better to rank and gather the correct paraphrases into the top ranks, that the precision of top 5 ranked patterns is slightly raised in larger scale of corpus. Unfortunately, the increased amount of generated paraphrases also linearly lowers the performance.

Thus, we guess that if the corpus can be accumulated during use, more patterns/praphrases can be generated, the coverage can get better. At the same time, to maintain an acceptable precision and performance, better scoring algorithm and some pruning work may help.

Chapter 7

Conclusion

Our previous work has built up a pattern-based matching module for question analysis in the QA system. This module uses handwritten patterns to match and analyze user's input questions. So far the patterns are all written by human editors, and such work is not only too costly, inefficient but also having low coverage with errors.

In this thesis, we intend to use paraphrase generation techniques to assist human editors to create patterns for the pattern-based matching module, that with input questions in natural language, human editors only need to select from the generated paraphrases/patterns as patterns for matching. This is expected to make the pattern writing more efficiently and provide a better coverage to cover potential questions, even some ungrammatical queries, in the QA system.

Despite the limited corpus and the lack of dependency parser, we propose an approach of generating paraphrases in Chinese. In this approach, we introduce a method for inferring the transformation operations from a question to its paraphrases. This method infers the transformation operations by extracting common parts in paraphrases pair from original question and filling the common parts into the structure "templates" of the paraphrase. We also introduce a representation of sentence structure. This representation represents the structure of a sentence as a sequence of tags, that the tags are either POS tags or more detailed "functional" tags, and each of them sequentially relates to a word or a phrase in the sentence. In addition, with some simple tag combination rules, we merge some consecutive tags into a dependency tree for easier comparison and alignment between structures. Thus, the sentences with same structure can align with each other. Based on the transformation operation inference method and the sentence structure representation, our approach of paraphrase generation takes a question as input, extracts all the questions

with same structure to the input question from corpus, and applies the transformation operations, which inferred from the extracted questions to their paraphrases, on the input question to generate its paraphrases.

We implement the approach of paraphrase generation we propose. The corpus contains both handwritten paraphrases and paraphrases extracted from CQA database. Except for generating paraphrasing based on the our approach, we also rank the generated paraphrases by scoring each of them on the semantics similarity between input question and extracted same structure question, and the reasonableness and certainty of the paraphrase. Thus, the paraphrases that are more likely to be correct are ranked in front of the others. Besides, for the use of pattern-based matching, some of the words in the paraphrases are represented as their tags to reduce the error, and the generated paraphrases are reformulated into patterns to improve the coverage and readability.

In evaluation, with 3424 questions in corpus, we generate 14.67 patterns (merged from 17.97 paraphrases) for each of the 150 test input questions on average. 51.79% of the generated patterns are correct. To simulate potential question asked by users, we handwrite 399 paraphrases examples for the 150 test input questions, and our generated patterns can cover 35.6% of them which simulate potential user questions, while human editors can only cover 19.0%. We evaluate the performance of our implementation, that the generation takes 4.9 seconds for each input question on average, which is quite acceptable for an offline assistant. The ranking method used in implementation is also evaluated as effective that the correctness of generated paraphrases has a significant positive correlation to their scores. As the corpus can be expanded during application, we scale down the corpus, then from the evaluation results we estimate that the expanded corpus should benefit the coverage and ranking accuracy.

The evaluation results indicate that compared to human editors, our approach can provide a much better coverage to human asked questions by generating paraphrases with acceptable precision. Therefore, Our approach is effective to assist human editors to build up the pattern-based matching module more efficiently. Meanwhile, the case analysis shows that the accurate paraphrasing is still a very difficult task. The structural and syntactical variations in paraphrasing are complex and flexible. Neither the sentence structure representation nor the transformation operation inference we propose is able to represent the true nature of paraphrasing. So we think that based on large scale of corpus, some machine learning approaches with fine-grained representation of sentence structures and transformation operations might learn more of the mechanism in paraphrasing.

References

[1] Ali Mohamed Nabil Allam and Mohamed Hassan Haggag. The Question Answering Systems: A Survey. *International Journal of Research and Reviews in Information Sciences*, 2(3):211–221, 2012.

[2] Yoav Artzi, Nicholas FitzGerald, and Luke S Zettlemoyer. Semantic Parsing with Combinatory Categorial Grammars. *ACL (Tutorial Abstracts)*, 3, 2013.

[3] Jason Baldridge. Lexically specified derivational control in combinatory categorial grammar. 2002.

[4] Henk P Barendregt and Erik Barendsen. Introduction to lambda calculus. *Nieuw archief voor wisenkunde*, 4(2):337–372, 1984.

[5] Regina Barzilay and Lillian Lee. Learning to paraphrase: an unsupervised approach using multiple-sequence alignment. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 16–23. Association for Computational Linguistics, 2003.

[6] Charles H Bennett, Péter Gács, Ming Li, Paul MB Vitányi, and Wojciech H Zurek. Information distance. *Information Theory, IEEE Transactions on*, 44(4):1407–1423, 1998.

[7] Matthew W Bilotti, Boris Katz, and Jimmy Lin. What works better for question answering: Stemming or morphological query expansion. In *Proceedings of the Information Retrieval for Question Answering (IR4QA) Workshop at SIGIR*, volume 2004, pages 1–3, 2004.

[8] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In

Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pages 1247–1250. ACM, 2008.

[9] Igor A Bolshakov and Alexander Gelbukh. Synonymous paraphrasing using wordnet and internet. In *Natural Language Processing and Information Systems*, pages 312–323. Springer, 2004.

[10] Eric Brill, Susan Dumais, and Michele Banko. An analysis of the AskMSR question-answering system. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing-Volume 10*, pages 257–264. Association for Computational Linguistics, 2002.

[11] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

[12] Davide Buscaldi, Paolo Rosso, and Emilio Sanchis Arnal. A wordnet-based query expansion method for geographical information retrieval. In *Working Notes for the CLEF Workshop*, 2005.

[13] Andrew Carlson, Chad Cumby, Jeff Rosen, and Dan Roth. The SNoW learning architecture. Technical report, UIUCDCS, 1999.

[14] Bill Dolan, Chris Quirk, and Chris Brockett. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proceedings of the 20th International Conference on Computational Linguistics*, page 350. Association for Computational Linguistics, 2004.

[15] Anthony Fader, Luke S Zettlemoyer, and Oren Etzioni. Paraphrase-Driven Learning for Open Question Answering. In *ACL (1)*, pages 1608–1618. Citeseer, 2013.

[16] Guangyu Feng, Kun Xiong, Yang Tang, Anqi Cui, Jing Bai, Hang Li, Qiang Yang, and Ming Li. Question Classification by Approximating Semantics. In *Proceedings of the 24th International Conference on World Wide Web Companion*, pages 407–417. International World Wide Web Conferences Steering Committee, 2015.

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

[18] Poonam Gupta and Vishal Gupta. A survey of text question answering techniques. *International Journal of Computer Applications*, 53(4):1–8, 2012.

[19] Ron Kohavi. Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid. In *KDD*, pages 202–207. Citeseer, 1996.

[20] Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1223–1233. Association for Computational Linguistics, 2010.

[21] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710, 1966.

[22] Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics, 2002.

[23] Percy Liang. Lambda Dependency-based Compositional Semantics. *arXiv preprint arXiv:1309.4408*, 2013.

[24] Percy Liang, Michael I Jordan, and Dan Klein. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446, 2013.

[25] Dekang Lin and Patrick Pantel. Discovery of inference rules for question-answering. *Natural Language Engineering*, 7(04):343–360, 2001.

[26] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[27] David Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM (JACM)*, 25(2):322–336, 1978.

[28] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for Naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, volume 752, pages 41–48. Citeseer, 1998.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[30] George A Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[31] Santosh Kumar Ray, Shailendra Singh, and Bhagwati P Joshi. Exploring Multiple Ontologies and WordNet Framework to Expand Query for Question Answering System. In *Proceedings of the First International Conference on Intelligent Human Computer Interaction*, pages 296–305. Springer, 2009.

[32] Stefan Riezler, Alexander Vasserman, Ioannis Tsochantaridis, Vibhu Mittal, and Yi Liu. Statistical machine translation for query expansion in answer retrieval. In *Annual Meeting-Association For Computational Linguistics*, volume 45, page 464, 2007.

[33] Svetlana Stoyanchev, Young Chol Song, and William Lahti. Exact phrases in information retrieval for question answering. In *Coling 2008: Proceedings of the 2nd Workshop on Information Retrieval for Question Answering*, pages 9–16. Association for Computational Linguistics, 2008.

[34] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[35] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[36] Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *Addison Wesley Publishing Company, USA., ISBN*, 10:020102988X, 1979.

[37] Luke S Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*, 2012.

[38] Yu Zhang, Wei-Nan Zhang, Ke Lu, Rongrong Ji, Fanglin Wang, and Ting Liu. Phrasal paraphrase based question reformulation for archived question retrieval. *PloS One*, 8(6):e64601, 2013.

[39] Shiqi Zhao, Xiang Lan, Ting Liu, and Sheng Li. Application-driven statistical paraphrase generation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 834–842. Association for Computational Linguistics, 2009.

[40] Shiqi Zhao, Haifeng Wang, Chao Li, Ting Liu, and Yi Guan. Automatically Generating Questions from Queries for Community-based Question Answering. In *IJCNLP*, pages 929–937. Citeseer, 2011.