Department of Creative Informatics
Graduate School of Information Science and Technology
THE UNIVERSITY OF TOKYO

Doctoral Thesis

Latent Relational Web Search Engine Based on the Relational Similarity between Entity Pairs

---

Nguyen Tuan Duc
****

Supervisor: Professor Mitsuru Ishizuka

December 2011

Abstract

The World Wide Web contains a huge number of Web pages which refer to numerous semantic relations. When a user wants to search for an entity in a specific semantic relation using a keyword-based Web search engine, the user must formulate a query with some keywords related to the entity and the relation. The user then inputs this query into the keyword-based Web search engine to retrieve a set of text snippets which the user must read to find out the answer. Moreover, when one does not explicitly know appropriate keywords to formulate a query, one can not get answers by using keyword-based Web search engines. With the growing number of entities and semantic relations on the Web, Web search engine users frequently face with such situations. Therefore, new entity retrieval paradigms based on semantic relations between entities are required to alleviate this problem. In this thesis, we study the problem of latent relational search, a novel entity retrieval method that enables Web search engine users to directly retrieve appropriate entities in an implicitly stated semantic relation. Specifically, given a latent relational search query {(A, B), (C, ?)}, in which A, B, C are entities, a latent relational search engine is expected to retrieve a list of entities **L** containing candidate answers to fill in the question mark (?) in the query. In the list **L**, each entity D satisfies the condition that the semantic relation between A and B is highly similar to that between C and D. For example, given the query {(Japan, Tokyo), (France, ?)}, a latent relational search engine is expected to retrieve and rank the entity "Paris" as the first answer in the result list, because the relation between Japan and Tokyo is highly similar to that between France and Paris.

To perform latent relational search on the Web, one must overcome several challenges: discovering entity pairs to build an index for high speed retrieval, exploring and representing the semantic relations between entities, and ranking the candidate answers according to the degree of relational similarity between the candidate entity pairs and the input pair. We propose a method for extracting entity pairs from a text corpus to build an index for a high speed latent relational search engine. Following previous work on relational similarity measuring algorithms, we represent the relation between two entities in an entity pair using lexical patterns of the context surrounding the two entities. We propose a lexical pattern extraction algorithm which enables the search engine to precisely measure the relational similarity between two entity pairs and therefore to accurately rank the result list of a latent relational search query. Different from previous work on latent relational search, the proposed retrieval model allows supporting sentences to be retrieved as evidences for each result. These evidence sentences provide the users of the search engine with further knowledge concerning the common semantic relations between the input entity pair and each retrieved candidate entity pair.

Moreover, we propose cross-language latent relational search, an advanced latent relational search paradigm that allows answering the query {(A, B), (C, ?)} when the input pair (A, B) is written in another language from the language of the entity C. By slightly extending the proposed retrieval model for monolingual latent relational search, we could adapt the model for processing cross-lingual latent relational search queries. Specifically, to capture the similarity between relations across languages, we must transfer the meaning of lexical patterns from one language to another. We propose a novel lexical pattern clustering algorithm to recognize paraphrased lexical patterns across languages, thereby effectively ranking candidates and retrieving evidence sentences for cross-lingual queries.

We evaluate the proposed search engine on both monolingual query sets and English-Japanese cross-lingual query sets. The experimental results show that, the proposed method outperforms existing latent relational search engines on monolingual query sets.

The search engine also achieves a moderate Mean Reciprocal Rank (MRR) on cross-lingual latent relational search query sets. Importantly, for the majority of cross-lingual queries, the search engine retrieves supporting sentences that are semantically similar in two different languages. This implies that the results of the search engine can be used for building parallel corpora or for supporting human translators. In particular, when evaluating with an ideal corpus, the proposed search engine retrieves the correct answers in the Top 1 ranked result for 94% of monolingual queries in English and 88% in Japanese. When evaluating with Japanese - English cross-language latent relational search queries, the proposed method achieves an MRR of 0.605 while requiring a short query processing time. Finally, we show that the proposed model can be applied to build a large-scale latent relational search engine with real-world corpora. Specifically, we use seven million articles in the English and Japanese Wikipedia data dumps to build an index for the search engine and use the search engine to answer several sophisticated questions in the INEX 2008 Entity Ranking task. The results show that, the search engine was able to answer 15 (out of 35) queries in monolingual mode, where as, in cross-lingual settings, the number of successfully answered questions was 12 (out of 35). The average query processing time of monolingual queries (for which the search engine relies only on information in the index) is three seconds, which is a practical time for normal search sessions. This demonstrates the capability to answer sophisticated questions concerning entities and relations on the Web of the search engine, provided that the index size is large enough for the search engine to recognize the semantic relations between the entities in a query.

Although there is a limitation in the coverage of the current system over the query spectrum due to the lack of data and processing power, the evaluation results reveal that the proposed method could achieve high precision on the task of latent relational search. Therefore, we expect that the proposed method could open a new direction in information retrieval and question answering on the Web.

Acknowledgements

I am extremely grateful to my supervisor, Professor Mitsuru Ishizuka for all the supports he has given me during three years of the Ph.D course. He guides me not only on the direction of my research, but also on choosing conferences and journals for publications. He popularizes the research on latent relational search by delivering the problem in his keynote speeches at conferences, his patent applications and his proposal to the search engine giant Google. All of the achievements in this work would have not been created without his endless efforts in promoting latent relational search to the information retrieval community.

I am grateful to the professors in the thesis committee, Prof. Kei Hiraki, Prof. Masayuki Inaba, Prof. Hiroshi Esaki, Prof. Mary Inaba and Prof. Kumiko Tanaka-Ishii, who suggest precious ideas and comments to this research.

I would like to express my gratitude to Dr. Danushka Bollegala for guiding me through my Ph.D research. Without his insightful ideas, comments and suggestions, this work would have not been done. I have learned a lot from his methods in conducting research, doing experiments and writing papers. As an excellent researcher, he kindly taught me how to write papers and how to pass through high-standard academic conferences and journals.

I appreciate precious suggestions on this work from Dr. Peter D. Turney of the Canada National Research Council (CNRC).

I would like to thank all members of the Matsuo-gumi, especially, Prof. Yutaka Matsuo, Prof. Naoaki Okazaki, Dr. Junichiro Mori, for their interesting presentations in each meeting and for valuable comments on this work.

I would also like to thank Hiroshi Dohi and Meiko Fujita for helping me in my research life, especially in travel planning, thesis and slides preparation.

I thank Shohei Tanaka for his friendship during the time I stay at Ishizuka Laboratory, and for his excellent supports in operating a large PC cluster. I thank Tomokazu Goto, Sho Kawasaki, Kazuki Hirashima, Nobuhiko Ochiai and Yorihiro Nobuta for being my colleagues in this research.

I thank Hong Anh for creating many brilliant latent relational search queries in the evaluation of the system on the INEX 2008 Entity Ranking task.

I would like to express my sincere thanks to all other members of Ishizuka Laboratory for their encouragements and supports for my research, as well as my daily life.

This work was submitted to various conferences and journals. I would like to thank the anonymous reviewers for valuable comments and suggestions during the review process, which greatly improved the quality of this research.

This research received financial supports from several companies and organizations. I and my colleagues would like to take this opportunity to convey our gratitude to the IPA Mitoh Program of the Information-technology Promotion Agency, Japan (IPA), especially Kenji Hiramoto, the project manager who guided me in the development of Milresh. We would also like to thank Google Inc. for the grant from Google Research Award.

Finally, I wish to thank everyone else for their aids and encouragements that allow me to complete this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

The World Wide Web contains a huge number of Web pages referring to numerous entities and semantic relations between those entities. Traditional keyword-based Web search engines enable users to search for Web pages containing some specified keywords. When a user wants to search for an entity in a given semantic relation using a keyword-based Web search engine, the user must formulate a query with some keywords related to the entity and the relation. The user then inputs this query into the keyword-based Web search engine to retrieve a set of text snippets which the user must read to find out the answer. Moreover, when one does not explicitly know appropriate keywords to formulate a query, one can not get answers by using keyword-based Web search engines. According to an analysis, approximately 71% of Web search queries contain named entities [1], and a significant portion of Web search queries (20–30%) exactly target named entities [2]. With the continuing growth of the number of entities and relations on the Web, these ratios would increase in near future. This indicates that, Web search engine users frequently face with the query formulation problem regarding named entities. Therefore, relying only on traditional keyword-based Web search engines is not sufficient to fulfill users' information needs. To alleviate this problem, a large amount of research on extracting entities [3, 2] and relations from the Web [4, 5, 6] has been conducted. Moreover, many studies have focused on information retrieval and question answering with entities and relations on the Web [7, 8, 9, 10].

For a Web search engine user, enumerating all appropriate keywords concerning a semantic relation is not a practical method to retrieve a comprehensive list of entities that participate in the relation. Numerous query expansion techniques have been proposed to refine the input query with appropriate keywords to precisely and comprehensively retrieve a ranked list of answers [11, 12, 13, 14]. However, traditional query expansion techniques do not support searching for an entity in a specific relationship because these techniques mainly focus on finding related keywords in the same documents with the entity. Therefore, new retrieval models and techniques are required to support Web search engine users to easily specify the keywords related to an entity in a relationship. This motivates us to explore the *latent relational search* approach, an entity retrieval approach based on semantic relations between entities, in which the semantic relations are implicitly stated by only a single relation instance as an example to guide the search process. Figure 1.1 shows an example of a monolingual latent relational search query. In the figure, the search engine is given three entities: two entities in an entity pair *(Japan, Mt. Fuji)* and a third entity (*Germany*) with a question mark (*?*). The search engine relies on some sentences in its corpus such as "Japan's highest mountain is Mt. Fuji." and "Germany's tallest mountain is Zugspitze." to output a ranked list of entities with *Zugspitze* as the top of

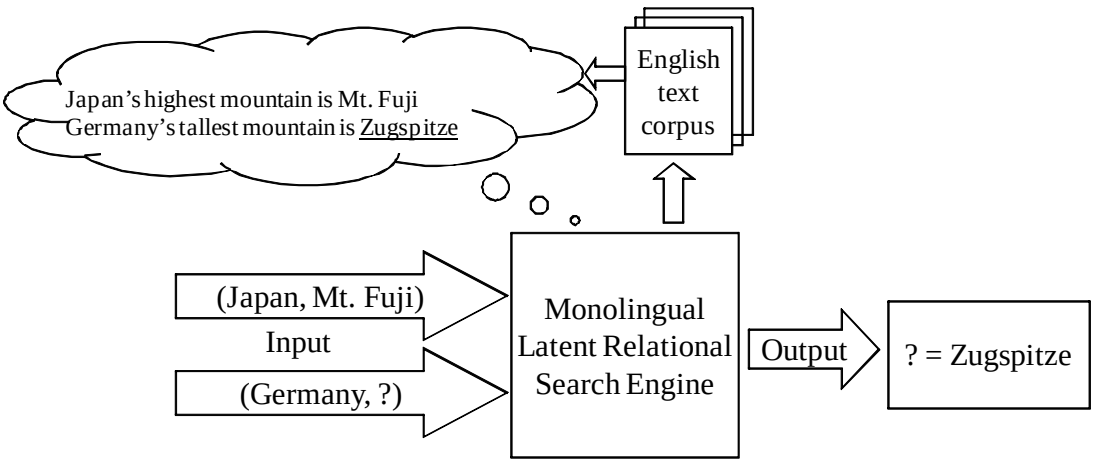


Fig. 1.1. An example of a monolingual latent relational search query.

the list to fill in the position of the question mark. We denote this query as {(Japan, Mt. Fuji), (Germany, ?)}. Therefore, a latent relational search query has the form of {(A, B), (C, ?)}, in which A, B, C are input entities. We call the entity pair (A, B) as the "source entity pair" (or the "source pair") and the entity C as the "key entity". The objective of latent relational search is to retrieve a ranked list $\mathbf{L}$ of entities so that for each entity $D \in \mathbf{L}$, the semantic relation between A and B is similar to that between C and D. We call the entity pair (C, D) as the "target entity pair" (or the "target pair"). We call the sentences that the search engine relies on to output the answer as "supporting sentences" or "evidence sentences" for the input query. It is important to note that, there are multiple keywords to express the relations between *Mt. Fuji* and *Japan*, such as "highest mountain", "tallest mountain", "highest peak", "volcano in", ... Therefore, enumerating all of these keywords is more difficult than simply input the entity pair (Japan, Mt. Fuji) as an example to guide the search engine. In this query, the search engine is using *relational similarity* to search for the target entity but the relation between the two entities in the source entity pair is not explicitly stated. Therefore, we name this paradigm of search as "*latent relational search*". Latent relational search can be effectively used when a user does not know the keywords to search for (e.g., the keyword *Zugspitze* in the above example). In such situations, the user can use a latent relational search engine to find the target keywords and then use an existing keyword-based Web search engine to obtain relevant results.

The idea of latent relational search has been discussed and partially implemented in many previous studies, such as the CopyCat system [15, 16], the structure mapping theory [17], the analogical thesaurus [18], the latent relational mapping engine [19] or recently in the work of Bollegala et al. [20] and Kato et al. [8]. However, building a practical latent relational search engine which can accurately answer queries in high speed is still a challenge for the information retrieval and question answering research community. In the first part of this thesis, we address this problem. We propose methods for extracting entity pairs and relations from a text corpus to build an index to precisely retrieve answers for latent relational search queries in high speed.

Because a large portion of the Web is non-English, cross-language information retrieval and cross-language question answering using Web data have become important than ever before. Many attempts have been made in the field of Web-based question answering [21, 22] and cross-language question answering [23, 24] to overcome the lan-

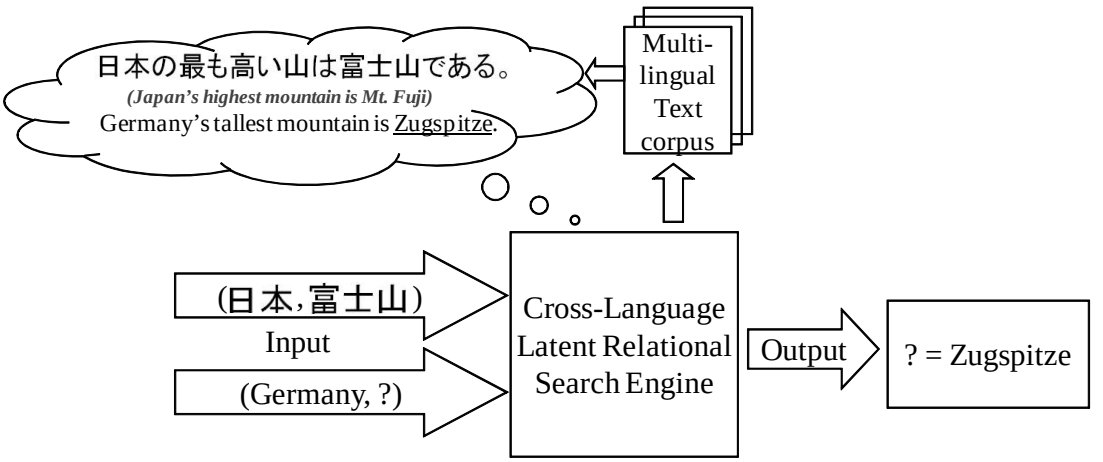


Fig. 1.2. An example of a cross-lingual latent relational search query, the input pair is in Japanese, meaning (Japan, Mt. Fuji).

guage barrier in information retrieval. However, cross-language information retrieval and question answering systems rely heavily on machine translation, which might produce poor results because of the noise in Web text and the lack of resources such as parallel corpora or translation dictionaries for some language pairs [25]. This motivates us to explore new approaches to tackle the problem of cross-lingual information retrieval that can work with noisy Web data and are tolerant to errors in underlying machine translation systems. Specifically, in this research, we propose *cross-language latent relational search*, in which only simple phrases are required to be translated, to alleviate adverse effects attributable to machine translation in cross-language information retrieval and question answering systems. In cross-language latent relational search, the input entity pair and the target entity pair of a query might be written in two different languages, possibly with different writing systems (e.g., English and Japanese) and the supporting sentences for these entity pairs are also in two different languages.

An example of cross-language latent relational search is answering the question "ドイツの最も高い山の名前は何ですか。" (meaning "What is the name of the highest mountain in Germany?" in Japanese) when a user only knows that the highest mountain in Japan is "富士山" ("Mt. Fuji") and there are not enough Japanese web pages concerning the highest mountain in Germany for a Web-based question answering system to find the answer. In this situation, the user can formulate the query {(日本, 富士山), (Germany, ?)} (the first entity pair is (Japan, Mt. Fuji) written in Japanese), to obtain the answer "Zugspitze", as shown in Figure 1.2. This kind of queries might be useful when a Japanese user is traveling to Germany and wants to visit some places like Mt. Fuji in Japan. In Figure 1.2, the search engine relies on some supporting sentences in Japanese (to identify the relation between 日本 (Japan) and 富士山 (Mt. Fuji)) and some other sentences in English (to identify the relation between Germany and Zugspitze) to output the answer "Zugspitze". A cross-lingual latent relational search engine therefore must recognize the similarity of semantic relations across languages. We propose a method for extracting entities and relations between these entities to efficiently search for similar entity pairs even when the entity pairs are in different languages. Moreover, we propose a novel two-phase clustering algorithm to capture the semantic similarity of lexical patterns across languages. Using the result of this algorithm, we can measure the relational

similarity of two entity pairs when they are in different languages. In this work, we focus on Japanese-English cross-language latent relational search because the Japanese-English pair is one of the most difficult language pairs for machine translation [26]. We believe that if we can get a reasonable result in Japanese-English cross-language latent relational search then there is a high probability that we can achieve better performance for other language pairs. In a Japanese-English cross-language latent relational search query, the source entity pair (A, B) might be in English, whereas, the target entity pair (C, D) might be in Japanese and vice versa. Moreover, the evidences (supporting sentences) that the search engine can rely on are also in different languages (Japanese or English).

There are several methods for answering latent relational search queries [8, 4, 9]. However, these methods focus on monolingual latent relational search, as they represent the semantic relations between two entities in an entity pair by terms or lexico-syntactic patterns from the context surrounding the two entities and compare them in the same language. Consequently, if there are not any sentence pairs that mentioned the source pair and the target pair in the same language, then these search engines do not have sufficient contexts to measure the relational similarity between the two entity pairs. Moreover, even while searching for an entity in another language, users often easily imagine a source entity pair in their own languages. For example, if a non-native Japanese speaker can only write down the source pair in English (not in Japanese), but the target entity is mainly mentioned in Japanese web pages, then the user can not use monolingual latent relational search to retrieve the answer. With the growing number of non-English documents on the Web, Web search engine users frequently encounter such situations. This indicates that there is a strong requirement for latent relational search across languages and the proposal in this thesis is important to overcome the language barrier in Web information retrieval.

1.2 Contributions of this Work

The main contributions of this work are as follows:

- We propose a new retrieval model for processing monolingual latent relational search queries in high speed. Specifically, we describe a method for extracting entity pairs and the relationships between those entities to build an index for high speed retrieval. Different from previous work, the problem of monolingual latent relational search that we try to solve in this research includes retrieving and ranking not only the candidate answers for a latent relational search query {(A, B), (C, ?)}, but also the supporting sentences that provide search engine users with additional knowledge concerning the semantic relation in the query. With the proposed retrieval model, supporting sentences can be easily retrieved.
- We propose a relation extraction method in which semantic relations between two entities are represented by lexical patterns of the context surrounding the two entities. Because the proposed relation extraction method is compatible with those in previous research concerning relational similarity measuring algorithms, it allows applying state-of-the-art relational similarity measuring algorithms to precisely calculate the relational similarity between entity pairs, and thereby accurately rank the result list.
- We propose the problem of cross-lingual latent relational search, an advanced latent relational search paradigm in which the input entity pair and the target pair are in different languages, possibly with different writing systems. The evidences (supporting sentences) that the search engine can rely on are also in different languages. This extends the capability of latent relational search from cross-domain knowledge

mapping to cross-domain and cross-language knowledge mapping.

- We propose a novel method for recognizing paraphrased lexical patterns across languages by using a hybrid lexical pattern clustering algorithm. Using this result, we were able to extend the proposed monolingual retrieval model to achieve a retrieval model for cross-lingual latent relational search by adding only simple modifications to the retrieval model for monolingual latent relational search.

To show the effectiveness of the proposed method, we evaluate the method with both monolingual query sets and Japanese-English cross-lingual query sets. When evaluating with an ideal corpus, the proposed method achieves a mean reciprocal rank (MRR) of 0.971 for English monolingual queries and 0.889 for Japanese monolingual queries. On Japanese-English cross-lingual query sets, the proposed method achieves an MRR of 0.605 and it retrieves semantically similar supporting sentences in two different languages (i.e., the language of the input entity pair and of the target entity pair). Moreover, when evaluate with a large corpus containing the entire English and Japanese Wikipedia data dumps, the proposed method could answer several sophisticated questions from the INEX 2008 Entity Ranking task.

Although in this work, we could not create a very large-scale latent relational search engine (e.g., a search engine with similar index size with those of currently operating commercial keyword-based Web search engines) because of the lack of data and processing power, the evaluation results imply that the proposed method could be used with huge corpora to answer a broad range of queries concerning entities and relations on the Web.

1.3 Organization of this Thesis

This thesis is organized as follows. In the next chapter (Chapter 2), we describe the problem of monolingual latent relational search and cross-lingual latent relational search that we solve in this research. We also introduce the concept of relational similarity, the basic concept underlying latent relational search. Next, in Chapter 3, we introduce related work on monolingual latent relational search, relational similarity and relation extraction systems. Chapter 4 proposes a retrieval model to accurately process monolingual latent relational search queries in high speed. It presents the algorithm for extracting and indexing entities and relations, as well as the method for retrieving and ranking candidate answers for a query. We also describe an implementation of the model and evaluate the performance of the proposed model on English and Japanese monolingual latent relational search queries. Chapter 5 proposes a method to extend the retrieval model in Chapter 4 for retrieving and ranking answer entities in cross-lingual latent relational search queries. Specifically, we propose a novel method to transfer semantic relations across languages to be able to retrieve and rank candidate answers of cross-lingual latent relational search queries. We describe an implementation of the proposed method and evaluate the method with an ideal corpus in this chapter. Next, in Chapter 6, we describe an implementation of the proposed retrieval method for processing large-scale corpora based on the MapReduce programming model. We evaluate the proposed search engine with the INEX 2008 Entity Ranking task to show that the search engine can answer sophisticated questions of various relation types. Finally, in Chapter 7, we conclude the thesis and discuss future research directions.

Chapter 2

Relational Similarity and Latent Relational Search

In this chapter, we describe the concept of *relational similarity*, which is important to understand the ranking function of latent relational search. We then describe the problems of latent relational search and cross-lingual latent relational search, the main targets of this research.

2.1 Relational Similarity

Relational similarity is the correspondence between two relations [27]. For example, the relation between Moon and Earth is similar to the relation between Phobos and Mars because Moon is a satellite of Earth, whereas, Phobos is a satellite of Mars. Consequently, we say that the relational similarity between the word pair (Moon, Earth) and the word pair (Phobos, Mars) is high. When two word pairs have a high degree of relational similarity, they are considered to be *analogous* [27]. We denote the degree of relational similarity between two word pairs (A, B) and (C, D) as $\text{RelSim}((A, B), (C, D))$. Therefore, $\text{RelSim}((Moon, Earth), (Phobos, Mars))$ is high, whereas, $\text{RelSim}((Moon, Earth), (lion, cat))$ is low, because there is not a clear correspondence between the relations of the first pair (Moon, Earth) and those of the second pair (lion, cat). There are several methods for measuring the degree of relational similarity (or simply the "relational similarity") between two pairs of words [28, 29, 30, 20]. Turney presents an application of relational similarity measuring algorithms to automatically answer Scholastic Aptitude Test (SAT) analogy questions [29, 27]. He reports that the proposed measure achieves an SAT score of 56.4, where as, the average score for high school students (human) is 57.0 [29, 31].

There are two types of similarities often referred in artificial intelligence and cognitive science: attributional similarity and relational similarity [32, 27]. As discussed above, relational similarity is the correspondence between two relations and is defined between two pairs of words, such as (Moon, Earth) and (Phobos, Mars). On the other hand, attributional similarity (or semantic similarity) is the correspondence between attributes and is defined between two words. When two words have a high degree of attributional similarity, they are called *synonyms* [27]. For example, the nouns *candy* and *sweet* are synonyms because they share several attributes such as they are eatable and they are made from sugar. In this thesis, we denote the attributional similarity (semantic similarity) between two words a and b as $\text{Sim}(a, b)$. Many previous studies on measuring the semantic similarity between two words using text corpora (especially, the Web) have been conducted [33, 34, 35, 36, 37, 38, 39]. Semantic similarity has numerous applications in

Web mining [40], Web page link detection and generation [41], paraphrase recognition [42], natural language processing [43, 37, 44], and information retrieval [11, 45, 46].

The work in this research is based on these two fundamental concepts of similarity, that is, attributional similarity and relational similarity. Although there are numerous practical applications of attributional similarity (semantic similarity) in natural language processing and information retrieval, there are few studies on practical applications of relational similarity. This thesis presents an application of relational similarity in information retrieval: the *latent relational search* paradigm.

2.2 Relational Similarity Measures

A *relational similarity measure* expresses the degree of relational similarity between two word pairs. There are several methods to measure the relational similarity between two word pairs, such as the method using WordNet taxonomy [18], using scores of χ^2-tests to verify the association between (C, D) and (A, B) [8] or based on the cosine similarity between two feature vectors [29, 30, 20]. In this work, we suppose that the relational similarity $\text{RelSim}((A, B), (C, D))$ between two word pairs (A, B) and (C, D) is always greater than or equal to zero:

$$\text{RelSim}((A, B), (C, D)) \geq 0 \tag{2.1}$$

The relational similarity measures proposed by Turney [29, 27] and Bollegala et al. [30, 20] satisfy this condition.

Indications in psychological science suggest that relational similarity is not symmetric. For example, we often say "an ellipse is like a circle" rather than "a circle is like an ellipse" [47]. This is because the above statement is directional: it has a subject and a referent. The choice for the subject and the referent depends on the relative salience of the objects. Human tend to select the more salient entity as a referent and the less salient entity as a subject [47]. Because "circle" is more salient (frequently used) than "ellipse", the statement "an ellipse is like a circle" would appear more frequently than the statement "a circle is like an ellipse". Consequently, the relational similarity $\text{RelSim}((ellipse, circle), (rectangle, square))$ might be larger than the relational similarity $\text{RelSim}((circle, ellipse), (square, rectangle))$. Similarly, because the pair (rectangle, square) is more salient than the pair (ellipse, circle) [*1], we would say "the pair (ellipse, circle) is like the pair (rectangle, square)", rather than "the pair (rectangle, square) is like the pair (ellipse, circle)". This indicates that $\text{RelSim}((ellipse, circle), (rectangle, square))$ is larger than $\text{RelSim}((rectangle, square), (ellipse, circle))$. Consequently, in this work, we do not assume that the relational similarity between two word pairs is symmetric. Therefore, in general:

$$\text{RelSim}((A, B), (C, D)) \neq \text{RelSim}((C, D), (A, B)) \neq \text{RelSim}((D, C), (B, A)) \tag{2.2}$$

However, experiments with similarities of word pairs suggest that, for a word pair (X, Y), the average difference between the similarity of X to Y and the similarity of Y to X is less than five percents (the average difference is 0.96, compared to the maximum similarity of 20) [48]. Therefore, the relational similarity between (A, B) and (C, D) would not be too different from that between (B, A) and (D, C). Consequently, we assume that the relational similarity $\text{RelSim}((A, B), (C, D))$ is not too different from $\text{RelSim}(((B, A), (D, C))$. We use this assumption in Section 4.7 while defining the relevance score of a candidate answer in latent relational search.

[*1] A Google query "+rectangle +square" returns about 43 million results, whereas, the query "+ellipse +circle" returns only 7 million results.

2.3 Latent Relational Search

In Section 1.1, we have given an overview of *latent relational search*. In this section, we describe the latent relational search problem based on the concept of relational similarity. We first introduce monolingual latent relational search, in which the search process mainly relies on supporting sentences written in the same language. We show that latent relational search could be a device for mapping knowledge across two different domains. We then propose cross-lingual latent relational search, an advanced latent relational search paradigm, in which supporting sentences are in different languages. We overview several potential applications of latent relational search in natural language processing, information retrieval and question answering.

2.3.1 Monolingual Latent Relational Search

Monolingual Latent Relational Search (or simply "Latent Relational Search") is an entity retrieval model based on the relational similarity between entity pairs. Specifically, latent relational search is an entity retrieval model in which,

- The inputs are three entities: an entity pair (A, B) which holds some specific semantic relations and an entity C. We call the entity pair (A, B) as the "source entity pair" (or the "source pair") and the entity C as the "key entity".
- The first goal of the retrieval process is to retrieve a list L of entities, such that for each entity $D_i \in L$, the relational similarity between the pair (A, B) and the pair (C, D_i) is larger than zero. Moreover, the list L is sorted in descending order of a relevance score based on relational similarity. That is, if $\text{RelSim}((A, B), (C, D))$ is the relational similarity between (A, B) and (C, D) and the relevance score is denoted as $\text{Rel}((A, B), (C, D))$ then each candidate answer $D_i \in L$ satisfies the following conditions:

$$\text{RelSim}((A, B), (C, D_i)) > 0 \tag{2.3}$$

$$\text{Rel}((A, B), (C, D_i)) \geq \text{Rel}((A, B), (C, D_j)), \forall i \leq j \tag{2.4}$$

  in which i and j are indices in the list L. We call the entity D as the "target entity" or the answer. Moreover, (C, D) is called the "target entity pair" or "target pair". We denote this latent relational search query as {(A, B), (C, ?)}. The relevance score $\text{Rel}((A, B), (C, D))$ is not required to be identical with $\text{RelSim}((A, B), (C, D))$, because we want to consider the reversed query {(B, A), (?, C)} while processing the original query {(A, B), (C, ?)}.
- The search engine has its local indices which can be obtained by analyzing some text corpora. In this work, the input text corpus contains crawled Web pages (HTML documents).
- The second goal of the retrieval process is to retrieve a set of "supporting sentences" (or "evidence sentences") that identify the semantic relations between the two entities A, B in the source pair as well as between the two entities C, D in the target pair. A latent relational search engine uses these sentences to extract the semantic relations that are held in the source pair and the target pair.

Although we only discuss the algorithm to process queries of the form {(A, B), (C, ?)}, we can easily use similar algorithm to process other forms of queries such as {(B, A), (?, C)} or {(C, ?), (A, B)}. Moreover, in this work, we only consider named entities (proper nouns) as arguments of a latent relational search query. This is because named entities

Table. 2.1. Example result list and supporting sentences for the latent relational search query {(Japan, Mt. Fuji), (Germany, ?)}

Rank	Entity	Supporting sentences
1	Zugspitze	- Japan's highest mountain is Mt. Fuji. - Germany's tallest peak is Zugspitze. - Mt. Fuji is the highest mountain in Japan. - The Zugspitze, at 2962 metres above sea level, is the highest mountain in Germany. - ...
2	Milseburg	- Mt. Fuji is a dormant volcano in Japan, which most recently erupted in 1708. - Milseburg is an extinct volcano in Hesse, Germany.
3	...	...

are of great interest to Web search engine users (71% of Web search queries contain named entities [1] and 20–30% of the queries exactly target named entities [2]). However, the methods discussed in this thesis are not restricted to named entities. We can use the same algorithm to extract and index common nouns or verbs, thereby allowing the queries to contain common nouns or verbs.

An example relational search query is shown in Figure 1.1 in Chapter 1. In this example, the source pair is (*Japan*, *Mt. Fuji*), the key entity is "Germany". The first result in the result list is "Zugspitze" and the supporting sentences for this result are "Japan's highest mountain is Mt. Fuji." and "Germany's tallest mountain is Zugspitze". The first sentence mentions the source entity pair and is called a *supporting sentence for the source entity pair*. The second sentence mentions the target entity pair and is called a *supporting sentence for the target entity pair*. There might be several entities that are appropriate to fill in the question mark in the above query. For example, the second result in the result list could be "Milseburg", because Milseburg is a volcano in Germany, whereas, Mt. Fuji is a volcano in Japan. Therefore, if the input source pair holds several different semantic relations then the result list will contain several appropriate answers. This indicates that, supporting sentences are important for search engine users because they provide the users with additional information concerning the reason why a result is output. Previous work on latent relational relational search [8, 4] does not focus on retrieving and ranking these supporting sentences. On the other hand, the method proposed in this thesis is designed for easily retrieving and ranking these sentences. This makes the proposed method more practical than previous methods. Table 2.1 gives an example of the result list and the supporting sentences for each result of the query {(Japan, Mt. Fuji), (Germany, ?)}.

Latent relational search can be used for mapping knowledge between different domains as can be seen in the above example. The user has knowledge about the highest mountain in Japan (the source domain). Using this knowledge, the user can formulate the query {(Japan, Mt. Fuji), (Germany, ?)} to query for the name of the highest mountain in Germany (the target domain). The search engine uses the input entity pair and its global knowledge concerning relational similarities that it extracted from a corpus to map the knowledge from the source domain (Japan's mountains) to the target domain (Germany's mountains) and retrieve the answer. That is, the knowledge of a familiar domain can be mapped to a novel domain to discover new knowledge in this novel domain. Therefore, latent relational search is useful for knowledge discovery and knowledge acquisition. Especially, when a user does not know which keywords are appropriate to formulate a query, latent relational search can be used very effectively. Using latent relational search, a user

can find an appropriate keyword (e.g., "Zugspitze") and can then use this keyword to formulate queries to a keyword-based search engine to retrieve related documents. Another situation which is demonstrated by Kato et al. [8] is when an Apple user wants to search for information related to a music player by Microsoft, but the user does not know the name of the product because he/she is not familiar with Microsoft's products. In such situations, the user can use a latent relational search query such as {(Apple, iPod), (Microsoft, ?)}, to retrieve the keyword *Zune*, the name of the music player.

Latent relational search has numerous potential applications in natural language processing, web mining, question answering and recommender systems. For example, Turney [49] proposed a unified approach to find synonyms, hypernyms, and antonyms using relational similarity. One can obtain hyponyms of the word "animal" as the result of a latent relational search query such as {(fruit, orange), (animal, ?)}. Another example is answering the question of "What is the name of the highest mountain in Germany?" when a user has known that the highest mountain in Japan is "Mt. Fuji". In this situation, the user can formulate the query {(Japan, Mt. Fuji), (Germany, ?)} to obtain the answer "Zugspitze".

When the supporting sentences that a latent relational search search engine relies on to retrieve and rank the result list for a query are in the same language, we call the search process "monolingual latent relational search" (or simply "latent relational search"). On the other hand, when the language of the source entity pair (the "source language") is different from the language of the key entity (the "target language"), there might be not enough supporting sentences in the source language to rely on to retrieve a result list. In this situation, we must use supporting sentences from two different languages and we must perform "cross-lingual latent relational search".

2.3.2 Cross-Lingual Latent Relational Search

Cross-lingual latent relational search is a latent relational search paradigm in which the language of the supporting sentences for the source entity pair is different from the language of the supporting sentences for the target entity pairs. We call the language of the supporting sentences for the source entity pair as the "*source language*" and the language of the supporting sentences for the target entity pairs as the "*target language*".

An example of a cross-lingual latent relational search query is shown in Figure 1.2 in Chapter 1. In the example, the source pair (日本, 富士山) (meaning (Japan, Mt. Fuji)) is written in Japanese and the key entity ("Germany") is written in English. A cross-lingual latent relational search engine is expected to retrieve a ranked list of answers in English. For example, the first answer in the result list is "Zugspitze" (the highest mountain in Germany), the second answer is "Milseburg" (a volcano in Germany), ... Moreover, for the first answer, "Zugspitze", the search engine is expected to retrieve some supporting sentences in Japanese for the source entity pair and some supporting sentences in English for the target entity pair, as shown in Table 2.2

Cross-lingual latent relational search can be effectively used when there are not any sentence pairs that mention the source entity pair and the target pair in the same language. In this situation, a monolingual latent relational search engine does not have any clues or supporting sentences to measure the relational similarity between the two entity pairs. Therefore, to retrieve a list of entities as the result list, the search engine must rely on supporting sentences in two different languages, the source language and the target language. Moreover, in many situations, the candidate answers are only written in a language that the user does not familiar to. For example, if a user can only write the source entity pair in English, but the target entity is mainly mentioned in Arabic or Japanese web pages, then monolingual latent relational search can not exploit these

Table. 2.2. Example result list and supporting sentences for the cross-lingual latent relational search query {(日本, 富士山), (Germany, ?)}, the first entity pair means (Japan, Mt. Fuji) in Japanese.

Rank	Entity	Supporting sentences
1	Zugspitze	_ 日本の最も高い山は富士山である。 (*Japan's highest mountain is Mt. Fuji*) - Germany's tallest peak is Zugspitze. _ 富士山は日本の最高峰である。 (*Mt. Fuji is Japan's highest peak.*) - The Zugspitze, at 2962 metres above sea level, is the highest mountain in Germany. - ...
2	Milseburg	_ 富士山は日本の火山活動歴史で重要な山である。 (*Mt. Fuji is an important mountain in the history of volcanic activities in Japan.*) - Milseburg is an extinct volcano in Hesse, Germany.
3	...	...

pages to search for the answer. A real-world example to illustrate this situation is when a Japanese user wants to find a list of recording companies that sell the *Kingston Trio*'s song. This is the topic number 23 in the TREC 2010 Entity Ranking track[*2] [50]. An English user can easily form a query such as {(Lady Gaga, UMG), (Kingston Trio, ?)} to retrieve the list. However, because many Japanese users might not be familiar with the UMG (Universal Music Group), they can not imagine such query. The user might use a Japanese-to-Japanese monolingual query such as {(*Hamasaki Ayumi*, *Eibekkusu*), (*Kingusuton Torio*, ?)} (all entities are written in Japanese) to retrieve the list. However, there is much little information concerning the Kingston Trio in Japanese than in English. As another option, the user might use traditional cross-lingual information retrieval systems to input the query in Japanese and retrieve the answers in English. However, the user must formulate some queries such as "*Kingusuton Torio hanbai kaisha*" (meaning "companies selling Kingston Trio" in Japanese). Moreover, the above query would not return a comprehensive list of recording companies that sell songs of the Kingston Trio because there are many sentences that describe the target relationship but do not contain the term "*hanbai*" (selling), even when we translate the term into English. For example, the sentence "Sony BMG records and distributes Kingston Trio's songs." mentions a company that sells the Kingston Trio's songs, but it does not strongly match the query. In this case, the user can use a Japanese-to-English cross-language latent relational search query (e.g., {(*Hamasaki Ayumi*, *Eibekkusu*), (Kingston Trio, ?)}) to get the answers because the pair (*Hamasaki Ayumi*, *Eibekkusu*) (meaning (Ayumi Hamasaki, Avex)) co-occurs with many lexical patterns that exactly describe the desired semantic relation. Therefore, latent relational search across languages would be very useful.

As mentioned in the previous section, latent relational search can be used as a device for mapping knowledge across two domains. Cross-lingual latent relational search extends the capability of latent relational search from cross-domain knowledge mapping to cross-domain and cross-language knowledge mapping. For example, in the cross-language latent

[*2] http://trec.nist.gov/data/entity/10/10.entity_topics

relational search query {(日本, 富士山), (Germany, ?)}, as shown in Figure 1.2, we are using the knowledge concerning the highest mountain in Japan (the source domain) in Japanese (the source language) to discover new knowledge concerning the highest mountain in Germany (the target domain) from documents written in English (the target language).

Chapter 3

Related Work

In this chapter, we review related work about analogical reasoning, relational similarity, relational search and information extraction from the Web. First, we describe previous work on analogical reasoning, which provides the theoretical basis for latent relational search. Next, in Section 3.2, we highlight some state-of-the-art relational similarity measuring algorithms, especially the algorithm that we use in this work to compute the relational similarity for ranking candidate answers. We then review previous studies on relation extraction from the Web, one of the main challenges that we tackle in this research in Section 3.3. We present a brief comparison of this work with previous research on latent relational search in Section 3.4. Quantitative comparisons between the proposed method with previous methods are provided in Chapter 4 and Chapter 6. Finally, in Section 3.5, we describe previous studies on cross-lingual information retrieval, which are related to cross-lingual latent relational search.

3.1 Analogical Reasoning

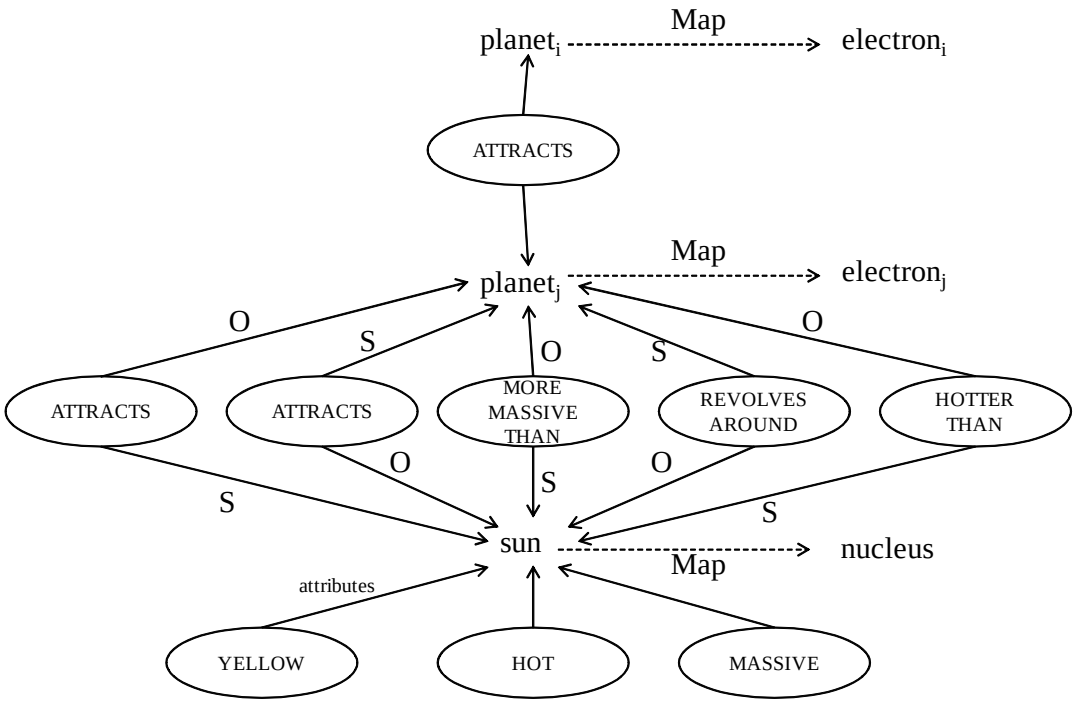
Analogical reasoning is an important topic in Artificial Intelligence. Many models have been proposed to solve analogy questions, such as the Structure Mapping Theory (SMT) [17, 51] and the Latent Relational Mapping Engine [19].

The Structure Mapping Theory (SMT) [17] is a framework for mapping knowledge from one domain to another domain. The SMT tries to preserve relations between objects and to discard attributes of objects. For example, the SMT allows mapping from a representation of the Solar System to a representation of the Rutherford-Bohr model of an atom [17], as shown in Figure 3.1. Gentner describes the mapping process as follows [17]:

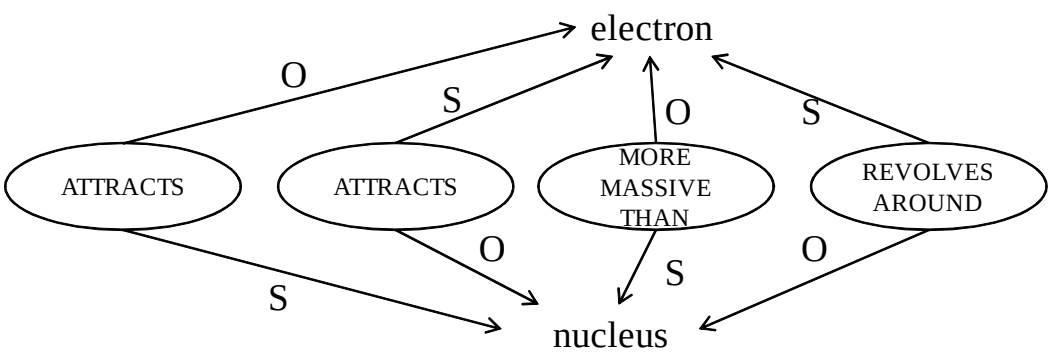
> Intended inferences concern chiefly the relational structure: e.g., "The electron REVOLVES AROUND the nucleus, just as the planets REVOLVES AROUND the sun", but not "The nucleus is YELLOW, MASSIVE, etc., like the sun".

Figure 3.1(b) shows the result of the mapping process (the mapped relations). In this example, the sun is mapped to the nucleus, the planets are mapped to the electrons, instead of the fact that the sun is very different from the nucleus in term of their attributes (the sun is very large, whereas, the nucleus is very small). Likewise, the planets have little common attributes with the electrons. However, the relation is very similar: the planets revolve around the sun and the electrons revolves around the nucleus. Therefore, SMT concentrates on mapping the relations between objects, rather than mapping the attributes of objects. This is similar to the process of latent relational search, where the common semantic relations between the source entity pair and the target entity pair are discovered.

However, the Structure Mapping Theory and its implementation, the Structure Map-



(a) The mapping process



(b) The result of the mapping process

Fig. 3.1. Structure-mapping for the Rutherford analogy: "The atom is like a solar system", from Gentner [17]. The label S denotes the subject of the relation, the label O denotes the object.

```
(defEntity sun :type inanimate)
(defEntity planet :type inanimate)
(defDescription solar-system
  entities (sun planet)
  expressions (((mass sun) :name mass-sun)
    ((mass planet) :name mass-planet)
    ((greater mass-sun mass-planet) :name >mass)
    ((attracts sun planet) :name attracts-form)
    ((revolve-around planet sun) :name revolve)
    ((and >mass attracts-form) :name and1)
    ((cause and1 revolve) :name cause-revolve)
    ((temperature sun) :name temp-sun)
    ((temperature planet) :name temp-planet)
    ((greater temp-sun temp-planet) :name >temp)
    ((gravity mass-sun mass-planet) :name force-gravity)
    ((cause force-gravity attracts-form) :name why-attracts)))
```

Fig. 3.2. The description of the solar system in the Structure Mapping Engine, from Falkenhainer et al. [51]

ping Engine (SME) [51] require complex hand-coded descriptions of the domain. For example, the description for the solar system in the SME must be hand-coded in LISP as shown in Figure 3.2. Several attempts have been made in order to assist users in describing the model, such as in Yan et al. [52] or Forbus et al. [53]. These approaches provide Graphical User Interface (GUI) to help the user to avoid coding the model in LISP, but still require burden of manual work to achieve the mapped knowledge. In this work, we do not require any hand-coded representation of the source entity pair and the target entity pair (we automatically extract the relations from text). Moreover, in latent relational search, it is not required to map the relations between two domains, but we only recognize similar semantic relations in the source pair and the target pair.

Turney [19] proposes the Latent Relational Mapping Engine (LRME) to avoid the problem of hand-coding knowledge representation in the Structure Mapping Engine. The Latent Relational Mapping Engine is a combination of the Structure Mapping Engine (SME) and Latent Relational Analysis (LRA). The input for the LRME are two lists of terms from two domains, the source domain and the target domain, as shown in Table 3.1.

LRME represents the relations between two terms using a feature vector whose elements are lexical pattern frequencies extracted from a large text corpus. Therefore, it does not require any hand-coded representation of the source and target domain. Given the two lists of terms in Table 3.1, LRME uses the corpus to build representations of the relations among the terms and then constructs the mapping between the two lists [19]. The mapping result is shown in Table 3.2. In this work, we also use a feature vector whose elements are lexical pattern frequencies to represent the semantic relations between two entities in an entity pair. However, different from the Latent Relational Mapping Engine, in latent relational search, only one pair of entities is given. For example, given the correspondence between *sun* and *nucleus*, a latent relational search engine is required to find the correspondence between *planet* and *electron*, as in the query {(sun, nucleus), (planet, ?)}. Therefore, the task in latent relational search is more difficult than in LRME in the sense that little information is given. On the other hand, in latent relational search, a correspondence (e.g., sun-nucleus) is given as an example, while in LRME, no

Table. 3.1. Example input lists for the Latent Relational Mapping Engine, from Turney [19]. The task is to find the correspondence between terms in the two lists.

Source (Solar system model)	Target (Rutherford atom model)
planet	revolves
attracts	atom
revolves	attracts
sun	electromagnetism
gravity	nucleus
solar system	charge
mass	electron

Table. 3.2. The output of the Latent Relational Mapping Engine for the input list in Table 3.1, from Turney [19]

Source (Solar system model)	Mapping M	Target (Rutherford atom model)
solar system	$\rightarrow$	atom
sun	$\rightarrow$	nucleus
planet	$\rightarrow$	electron
mass	$\rightarrow$	charge
attracts	$\rightarrow$	attracts
revolves	$\rightarrow$	revolves
gravity	$\rightarrow$	electromagnetism

correspondence is provided in the input.

Hofstadter et al. propose "a model of high-level perception and analogical thought in which perceptual processing is integrated with analogical mapping" [15, 16]. Using this model, they can solve the analogy puzzles such as "If **abc** changes to **abd**, what does **iijjkkll** change to?" when the computer knows only the orders of characters in the alphabet, and has no other knowledge. They describe that the analogical reasoning process is influenced by belief, goals and external contexts. Therefore, analogical mapping models must take into account these factors. A user of a latent relational search system might also be affected by this behavior. For example, given the query {(Google, YouTube), (Yahoo, ?)}, the answer might be any of the companies that Yahoo has acquired. Depend on the pragmatic understanding of the user, the best answer might be different (for example, if the goal of the user is to find a video sharing service that Yahoo has acquired, the best answer should be "Broadcast.com"). In this research, we rely on the frequencies of common (shared) lexical patterns to determine the relational similarity between entity pairs, as seen in [29, 30]. Therefore, we would not directly take into account the characteristics of entities and the pragmatic understanding of a specific user, but we do consider the pragmatic understanding of the majority of users, who write text on the Web, as the proposed search engine analyzes text on the Web to create its knowledge base (the index). For example, if the term YouTube co-occurs with the term "video sharing" with high frequency, then we would extract many lexical patterns that contain this term for the pair (Google, YouTube).

3.2 Relational Similarity Measuring Algorithms

Relational similarity has numerous potential applications in natural language processing and information retrieval. For example, Turney [27] applies relational similarity to classify the relationship between a noun and its modifier. In English, noun-modifier pairs, such as *flu virus*, *weekly game*, are frequently used. In these pairs, there is a head noun and a modifier which gives additional information concerning the noun. For example, in the pair *weekly game*, the head noun is *game* and the modifier *weekly* gives information about the frequency of the game [54]. The semantic relation between a modifier and a noun can be classified into five classes: causal (e.g., *flu virus*), participant (*student protest*), spatial (*home town*), temporal (*morning coffee*) and quality (*heavy rock*) [55, 54]. Classification of semantic relations in noun-modifier pairs is an important task to understand the meaning of noun phrases. If we know that the relational similarity between the pairs (printer, tray) and (automobile, wheel) is high, then we can group them together into a same semantic relation class (representing the relation part-of). This implies that the noun-modifier pairs *printer tray* and *automobile wheel* hold similar semantic relationship.

There are several methods to measure the relational similarity between two word pairs [29, 27, 30, 56, 8]. These methods mainly use terms [8] or lexical patterns [29, 56, 30] of the context surrounding a word pair to represent the relation between two words in the word pair. Latent relational search aims to retrieve a list of entities L as the result list for the query $\{(A, B), (C, ?)\}$ in which each entity $D \in L$ satisfies that the degree of relational similarity between (C, D) and (A, B) is high. Consequently, the result list L could be ranked by the relational similarity measure between the source pair and candidate target pairs. The methods in Turney [29] and Bollegala et al. [30, 20, 57] give highest precision in the task of measuring the relational similarity between two word pairs. In these studies, the relation between two entities in an entity pair is represented by lexical patterns, i.e., the context where the two entities appear. Using lexical pattern frequencies (or point-wise mutual information values) as elements of feature vectors of entity pairs, these studies achieve high precision on the task of measuring semantic similarity between two entity pairs. Therefore, in this research, we apply the similarity measuring algorithm proposed by Bollegala et al. [30] for measuring the relational similarity between entity pairs in the same language. Moreover, we adapt the above algorithm to be able to measure the relational similarity of semantic relations across languages. Details of the relational similarity measuring algorithms will be described in Chapter 4 and Chapter 5.

3.3 Relation Extraction from the Web

Because large text corpora such as the Web often contain numerous entities and relationships, several studies have addressed the problem of extracting entities and relationships between those entities from text corpora [58, 59, 60, 61, 62, 63, 7]. Hearst first uses lexico-syntactic patterns to extract hyponyms from a large text corpus [64]. He proposes to use the patterns "NP_0 *such as* NP_1", "NP_0*, including* NP_1" (in which NP_i is a noun phrase) to recognize that NP_1 is a hyponym of NP_0. Base on the work of Hearst, Berland and Charniak propose to use the lexical patterns such as "*the* NP_0 *of the* NP_1" to extract the *part-whole* (or *meronym*) relation [65]. For example, they extract the relation part_of(basement, building) from the lexical pattern "the *basement* of the *building*". Girju et al. further improve the extraction of the part-whole relation by using machine learning approach to select good lexical patterns representing the relation [66, 67].

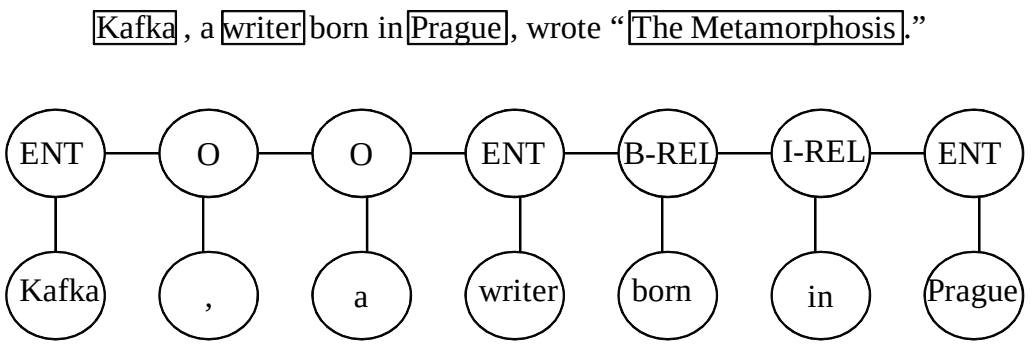


Fig. 3.3. Information extraction as sequence labeling. A CRF is used to identify the relationship, born in, between Kafka and Prague.
Entities are labeled as ENT. The B-REL label indicates the start of a relation, with I-REL indicating the continuation of the sequence. (from Etzioni et al. [74])

Brin proposes DIPRE (Dual Iterative Pattern Relation Expansion), a method for extracting entities and relations from the Web based on the pattern relation duality [58]. Given some entity pairs as exemplars for a specified semantic relation, DIPRE first extracts lexical patterns from the context that these exemplars appear. Using these lexical patterns, DIPRE searches for other entity pairs that co-occur with these patterns. There is a high probability that the obtained entity pairs also hold the same semantic relation with that of the input entity pairs. Therefore, DIPRE adds these entity pairs to the original set of entity pairs and repeats the lexical pattern generation process. The important idea is that, if we achieve a good lexical pattern set, then we can generate a good entity pair set. Likewise, if we have a good entity pair set, we can extract a good lexical pattern set. This is called the "pattern relation duality". Many subsequent studies have exploited this duality to extract entities and relations from the Web [68, 69, 70, 71].

Hierarchical clustering has been used to discover the hypernym (*is-a*) relation between nouns [72]. The method in [72] first uses a hierarchical clustering algorithm to cluster nouns into a hierarchy represented as a binary tree. Each leaf node of the tree is a noun. It then labels each internal node of the tree with an appropriate hypernym of the nouns in the sub-trees of the node. Pantel and Ravichandran [73] also use clustering to group semantically similar nouns into clusters, and then assign label for each cluster. We can then discover the hypernym relation between each element of a cluster and its label.

Banko et al. [5] propose the Open Information Extraction paradigm, which does not require pre-specifying the types of the relations to be extracted. They implemented the TextRunner system [74, 7] for large-scale open information extraction from the Web. TextRunner is a self-supervised relation extraction system. First, it uses a rule set to extract several relation instances as training data. This training data is then used to train a CRF (Conditional Random Field [75]) classifier to assign label for each entity/word in a sentence to classify if the current entity/word is participated in a relation or not. From the output of the classifier, they can extract entities and relations. An example of the extraction process is shown in Figure 3.3. From the sentence *Kafka, a writer born in Prague, wrote "The Metamorphosis."*, TextRunner first identifies four entities in the sentence: Kafka, writer, Prague and "The Metamorphosis". It then uses the CRF classifier to assign the label "B-REL" to the word *born* and "I-REL" to the word *in* to indicate that these words create a relation (the birthplace relation). From the output of

the classifier, we can extract the relation born_in(Kafka, Prague).

In this work, we use an unsupervised method to extract entities and relations from sentences in a document. Therefore, the proposed extraction algorithm is also a kind of open information extraction algorithm. Similar to TextRunner, we do not require any specific information concerning the types of the relations to be extracted. We simply extract all named entity pairs and filter out rare entity pairs. However, when extending the proposed method to common nouns and verbs (not named entities), we might need a method to prevent the explosion in the number of extracted entity pairs.

3.4 Existing Latent Relational Search Systems

The idea of latent relational search has been discussed by Hofstadter et al. [15, 16], Veale [18] and Bollegala et al. [20]. Veale [18] proposes a search paradigm in which queries such as "Muslim church" or "Greek A" can be answered. In these queries, the second concept (e.g., "church") is in different domain from the first concept (church is a concept in the domain of "Christian", not "Muslim"). The answer of this kind of queries should be the corresponding concept in the domain of the first concept. For example the answer for the query "Muslim church" should be "mosque" because mosque is to Muslim as church is to Christian. Similarly, the answer for the query "Greek A" is "α" (the Greek letter Alpha).

Veale also presents a method for answering such kind of queries by adding fine-grained concepts into WordNet*[1]. For example, to answer the query "Greek A" using WordNet, consider that all English letters and Greek letters are direct hyponyms of the same synset ("letter of the alphabet") in WordNet, so if we use only taxonomic information of WordNet, the concept "A" is similar to the concept "Alpha" as is to "Beta". Therefore, the relational similarity between two pairs {*(English, A), (Greek, Alpha)*} is identical to the relational similarity between {*(English, A), (Greek, Beta)*}. To be able to differentiate between these relational similarities, the method in [18] automatically adds concepts such as "English letter", "Greek letter" and "1^{st} letter", "2^{nd} letter" under the synset "letter of the alphabet", as shown in Figure 3.4. Using the newly created concepts, we make the relational similarity between {*(English, A), (Greek, Alpha)*} higher than between {*(English, A), (Greek, Beta)*} because "A" is now more similar to "Alpha" than to "Beta" ("A" and "Alpha" have a common direct hypernym "1^{st} letter"). Although the method relied on WordNet is intuitive to understand, it can not handle almost all named entities on the Web because WordNet does not cover all named entities. Because named entities are often interesting to search engine users, we need a method that allows the retrieval of these kinds of entities.

Latent relational search can be considered as a kind of template filling task which is one of the targets of MUC [76] and TREC [77]. The only slot for filling in latent relational search is the question mark in the query (e.g., {(A, B), (C, ?)}). Although there is only one slot, it is not simple to fill this template because there is very little information about the relation that the template describes. On the other hand, in MUC or TREC, the template often provides information concerning the relation (i.e., the system does not need to extract the relation, it only needs to extract the target entity). The Kiwi/Tonguen systems [78, 79] also fill templates which are provided by the users. For example, given the input phrase "$*$ is the president of France", Kiwi retrieves several usage examples that match the phrase (in which $*$ is an asterisk, represents any combination of words) from the Web. This effectively retrieves the answer for the question "Who is the

*[1] http://wordnet.princeton.edu/

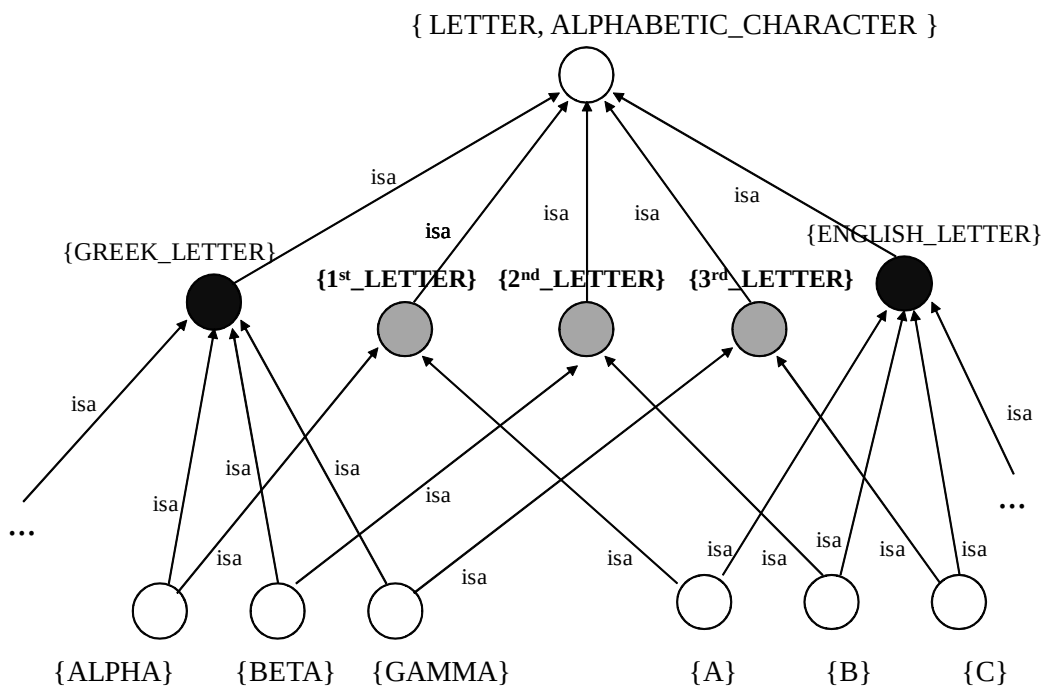


Fig. 3.4. Adding fine-grained concepts to WordNet for latent relational search. Concepts with background color of gray and black are added. (from Veale [18])

president of France?", because it fills in the position of the asterisk with the answer (e.g., "Nicolas Sarkozy"). Kiwi/Tonguen can retrieve phrase-based concordance in multiple languages. Therefore, Kiwi/Tongue are very effective for multi-lingual entity oriented question answering, provide that the user must know some appropriate keywords (or phrases) that the target entity often co-occurs with (e.g., "*president of*"). However, in latent relational search, there is no keyword or phrase that describes the target entity is provided. To find the answer, the relation provided in the source pair must be extracted or represented by some means. For example, we might represent the explicit semantic relation between the two entities in the source pair by some statistical proxies for the relationship, such as by a set of lexical patterns of the context surrounding the two entities [29, 20].

WWW2REL [4] is a system that can fill templates of the forms $R(arg_1, ?)$ or $R(?, arg_2)$, in which R is a relation and arg_1, arg_2 are arguments of the relation. For example, WWW2REL can find *apoptosis* as an answer for the query $INDUCES(aspirin, ?)$. It first discovers the lexical patterns that represent the relation R using 40 seed pairs that hold the relation R. The seed pairs are automatically selected by randomly taking word pairs from the UMLS Metathesaurus, a thesaurus for biomedical science. Using the seed pairs, it queries a Web search engine with queries of type "$arg_1 * arg_2$" to obtain text snippets from the search engine. The snippets are then used for extracting the lexical patterns that represent the relation R. In the second phase, it uses the given argument in the query and the extracted lexical patterns for querying the Web search engine to find the answer (e.g., for the query $INDUCES(aspirin, ?)$, it uses the word *aspirin* and the lexical pattern "*may cause*" to find the answer *apoptosis*). Therefore, the method requires that the thesaurus contains several instances of the relation in the query. If the thesaurus did not contain the relation's instances or there was no thesaurus, WWW2REL would give

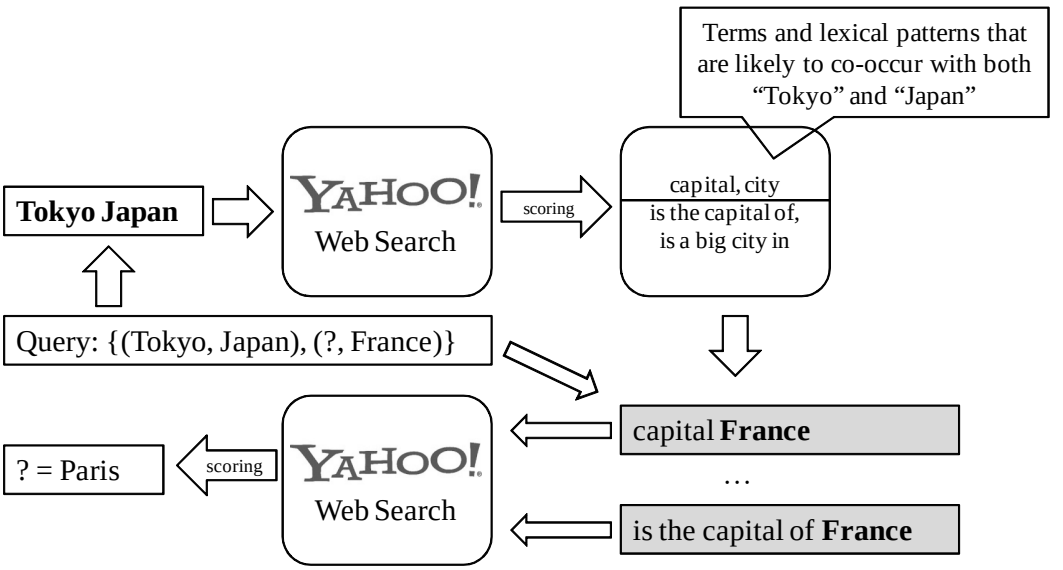


Fig. 3.5. The latent relational search method proposed by Kato et al. [8]

low precision or even would not find the answer.

An implementation of monolingual latent relational search is described by Kato et al. in [8]. They use two phases to find and rank candidates for the query $\{(A, B), (C, ?)\}$, as shown in Figure 3.5. Suppose that the answer for this query is D. First, they build a "relation extractor" $\mathrm{E}(A, B)$ for the pair (A, B) using a Web search engine. The relation extractor $\mathrm{E}(A, B)$ extracts terms or lexico-syntactic patterns that represent the relations between A and B (this term or pattern set is denoted by T). $\mathrm{E}(A, B)$ can be built by querying a Web search engine for terms or lexical patterns that are likely to appear only in documents which contain both A and B. When they use terms to represent semantic relations, they call the method as **TC** (term co-occurrence). When they use lexico-syntactic patterns to represent semantic relations, they call the method as **SP**. For example, the relation extractor of the **TC** method would extract terms such as "capital", "city", where as, in the **SP** method, it would extract lexical patterns such as "is the capital of", "is a big city in" for the pair (Tokyo, Japan). Then, in the second phase, they use the keyword C and the term or lexical pattern $t \in T$ for querying the Web search engine for documents that contain both C and t. The candidate set for the answer D is then the set of terms that are likely to appear only in these documents. For example, for the query {(Tokyo, Japan), (?, France)}, after extracting terms for the pair (Tokyo, Japan), the method **TC** issues queries such as "capital France" to a Web search engine to retrieve the answer "Paris", as shown in Figure 3.5. Therefore, the method **TC** represents the relations between two words in a given word pair by using a bag-of-words model. The method **TC** in [8] has advantages such as it can find a large range of term D (because it finds all terms D that are likely to appear with C and $t \in T$), it does not require a local index for searching (it uses an existing keyword-based Web search engine to find $\mathrm{E}(A, B)$ and D). However, it does not use an explicit relational similarity measure for ranking the result list, but instead it uses the likelihood of co-occurrence of term D and the pair (C, t) in a document for ranking. Term D might appear in a document with the terms C, t but the relation between C and D might not be expressed by the term t. For example, if C is *Microsoft* and t is *CEO*, the term *Windows* might co-occur with C and

t with high frequency. In these cases, the term *Windows* is in documents which contain both *Microsoft* and *CEO* but they might not be in a same sentence such as *"Microsoft's CEO is ... "* . Therefore, to achieve high precision, the relational similarity between (A, B) and (C, D) should be measured using a well-defined method such as in [20, 27], in which the relation between C and D is represented by lexical patterns that are in the same sentence with the pair (C, D).

On the other hand, the method **SP** is based on lexico-syntactic patterns in the gap between two entities in each entity pair. Given the source pair (A, B), **SP** first extracts lexical patterns in the gap between A and B or between B and A. It then uses these lexical patterns and the given key entity (C) to extract and assign scores to the answers. Therefore, the method **SP** directly compares the lexical patterns in the gap between (A, B) and (C, D). If there are not any identical lexical patterns in the gap between A and B and the gap between C and D, then the method **SP** would not find any answer. For example, in the two sentences "Obama is the 44th president of the U.S" and "Sarkozy is the current president of France", the method **SP** would not recognize the similarity between (Obama, U.S) and (Sarkozy, France) because lexical patterns it extracts do not match between the two pairs.

From the above discussion, we notice that the method **TC** might retrieve a broad range of entities (high recall), but with low precision because **TC** does not maintain the order of words in the contexts of the two entities in each entity pair. On the other hand, **SP** strictly maintains the order of words in the contexts, thereby resulting in high precision, but it might cause problem while searching for candidates and measuring relational similarities. The conjunction (**CNJ**) of the method **TC** and **SP** gives the best result, as reported by Kato et al. [8]. In the method **CNJ**, they heuristically combine the rank of an entity from the method **TC** with that from **SP** to achieve the final rank. This boosts the performance of the system because the strengths of **TC** (retrieving answers with high recall) and of **SP** (high precision) are combined together.

However, in the method **CNJ**, Kato et al. [8] heuristically define the coefficients in the combination (they fix the coefficients by a heuristic). In our proposal, we describe a method that directly combines both of the strengths of **SP** and **TC** from the lexical pattern extraction phase: the extraction algorithm maintains the order of words in the context (for high precision retrieval), but it is flexible for lexical pattern matching (to maintain high recall level). This combination is performed *before* calculating relational similarities (as well as ranking candidate answers). Therefore, the result of the combination is reflected into the representation of semantic relations, which is in turn used for calculating relational similarities. Consequently, it is a systematic approach for combination and it does not require any parameters to be heuristically set. More detailed comparisons and discussions concerning this issue are presented in Section 4.10.6.

It is also worth noting that, although the method by Kato et al. [8] does not require to build a local index for the retrieval process, it must issue several queries to external keyword-based Web search engines. This implies that the response time of a search engine based on the method in [8] is slow, comparing to a method that uses local indices. We will compare the query processing time of the proposed method with that of the method in [8] in Section 6.8.3.

Goto et al. [9, 80] propose a method for improving performance of latent relational search by exploiting the symmetries of relational similarity. Because the degree of relational similarity between (A, B) and (C, D) is not too different from that between (B, A) and (D, C), Goto et al. [9] propose to incorporate the score of D in the reversed query $\{(B, A), (?, C)\}$ when ranking results of the query $\{(A, B), (C, ?)\}$. Moreover, they investigated eight types of symmetries such as $\{(A, B), (?, D)\}$, $\{(A, ?), (C, D)\}$, ...and propose a method to combine the scores obtained from those queries to rank the final result list

of the query $\{(A, B), (C, ?)\}$. We also use the reversed query $\{(B, A), (?, C)\}$ in this research to calculate the rank in the final result list.

In this work, we propose a method for indexing entity pairs and relations to perform latent relational search in high speed. Moreover, we apply the method proposed by Bollegala et al. [30] to recognize paraphrased lexical patterns in the same language. This helps us to precisely measure the relational similarity between two entity pairs, and therefore achieve a high precision in the task of monolingual latent relational search. Furthermore, we propose a method for recognizing paraphrased lexical patterns across languages to process cross-lingual latent relational search queries.

3.5 Previous Work on Cross-Lingual Information Retrieval

Because the Web contains documents written in many languages, there are numerous studies that address the problem of cross-lingual information retrieval [81, 82, 83, 84, 85, 86]. In cross-lingual keyword-based information retrieval, the queries are written in a different language from the language of the documents (for example the query is in English, while the target documents are in Japanese). A cross-lingual information retrieval system often translates queries [87, 88] to match with documents in the target language. Dumais et al. propose a cross-language retrieval approach that does not require query translation [81]. They use Latent Semantic Indexing [89, 90] in multiple languages to map both documents and queries into a same vector space, which represents the semantic of documents. The similarity of a query with each document can be directly computed in this vector space (using cosine similarity). In cross-lingual latent relational search, we can translate the source entity pair into the target language (the language of the target entity pairs) and then perform monolingual retrieval. There are several methods for translation [85] and transliteration [91, 92, 93] of a named entity from a source language into a target language. However, if we use transliteration to retrieve the candidate answers in latent relational search, we can not retrieve the supporting sentences in two different languages, as specified in Section 2.3.2. We can use transliteration to achieve the result list before retrieving the supporting sentences for each result. Then we must use some translation techniques to retrieve and rank the supporting sentences. The proposed method in this work directly supports the retrieval of supporting sentences in different languages because it uses machine translation to translate lexical patterns which represent the relations between two entities in the source pair. Multi-lingual latent semantic analysis (LSA) has been applied in many studies to improve performance of cross-lingual information retrieval systems [81] and cross-lingual text classification [94, 95]. However, to the best of our knowledge, latent relational analysis (LRA [29]) has not been performed in multi-lingual. In this work, we propose *multi-lingual latent relational analysis* to improve the performance of a cross-lingual latent relational search system.

In cross-lingual latent relational search, we must recognize semantically similar lexical patterns across languages. Specifically, in this work, we propose a method to recognize paraphrased lexical patterns across languages by a hybrid (hard/soft) clustering algorithm of lexical patterns. Davidov and Rappoport [96] propose an approach for automated translation of semantic relationships across languages. The target of the work that they tackled is translation of lexical pattern clusters from a source language into a target language. Therefore, it is a sub-task of cross-language latent relational search, that is recognizing paraphrased lexical patterns across languages. Given a lexical pattern cluster that represents a semantic relation in a source language (e.g., English), they first use a Web search engine to retrieve a set of entity pairs that might hold this semantic relation, as shown in Figure 3.6. For example, from the pattern cluster {"X is the CEO of Y", "X, CEO

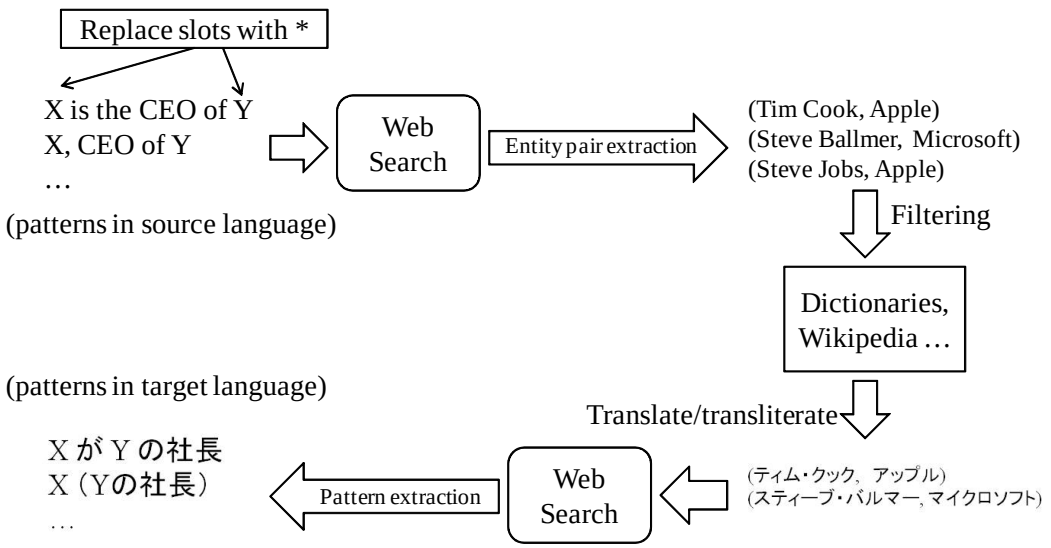


Fig. 3.6. The lexical pattern translation method by Davidov and Rappoport [96]

of Y"}, the method formulates some queries such as "∗ is the CEO of ∗", "∗, the CEO of ∗" and query a keyword-based Web search engine to retrieve text snippets such as "Steve Jobs is the CEO of Apple." or "Steve Ballmer, CEO of Microsoft". From these text snippets, they can retrieve entity pairs (e.g., (Steve Jobs, Apple), (Steve Ballmer, Microsoft), ...) that hold the semantic relation (the *CEO-COMPANY* relation). They then use Web hit count ratios to rank the entity pairs according to their degree of representativeness (*"specificity"*) for the given semantic relation. For example, the specificity of the pair (Steve Jobs, Apple) against the CEO-COMPANY relation might be calculated as the hit count ratio between the number of hits for the query "Steve Jobs, CEO of Apple" and the number of hits for the query "Steve Jobs ∗ ∗ ∗ Apple". At this step, an entity pair that holds more than one semantic relations (such as the pair (Steve Jobs, Apple) holds both the CEO-COMPANY relation and the FOUNDER-COMPANY relation) might be excluded because the specificity of the pair is low (i.e., this pair is not a representative pair for the CEO-COMPANY relation). The reason that causes the specificity to be low is that when an entity pair holds different semantic relationships, the hit count ratio between the number of hits for the query that matches the relation (e.g., "Steve Jobs, CEO of Apple") and the query that matches only the entities ("Steve Jobs ∗ ∗ ∗ Apple") will be small. After calculating the specificities, entity pairs with highest specificities are selected and are translated into the target language (e.g., Japanese) using dictionaries. At this step, cross-language links in Wikipedia can be utilized to translate named entities because an English Wikipedia page that refers to a named entity often has a link to the Japanese version of that page. From a translated entity pair (A, B), they formulate queries such as "A ∗ B" and "A ∗ ∗ B" to retrieve text snippets, which are the context in which this pair appears. For example, from the entity pair (*Sutivu Baluma, Maikurosofuto*) (in Japanese Katakana, which means (Steve Ballmer, Microsoft)), they can retrieve some text snippets such as "*X ha Y no shacho*", "*X ga Y no daihyou torishimari yaku*", etc. which represent the *CEO-COMPANY* relation in Japanese. They then rank the translated lexical patterns by using a confident score which represents the salience of each pattern to the translated entity pair set. Only patterns with high salient scores are selected in the final translated pattern cluster.

The method by Davidov and Rappoport [96] does not require a machine translation system and it works well for many language pairs. Therefore, it can be effectively used for translating lexical patterns in cross-lingual latent relational search when we can not utilize machine translation.

However, the final target of our research is not limited to translation of lexical patterns between languages, but to optimize the performance of cross-lingual latent relational search. In cross-lingual latent relational search, incorrectly translated lexical patterns might result in retrieving inappropriate candidate entity pairs, which do not hold the same semantic relation with the input pair. Consequently, we try to achieve highest accuracy in recognizing paraphrased lexical patterns across languages by exploiting both parallel entity pairs and parallel lexical patterns. Specifically, the most different point between the proposed method in this work and in [96] is the starting point. Our approach starts from parallel entity pairs to measure the semantic similarity between lexical patterns across languages. Furthermore, the proposed method also *directly* translates lexical patterns to obtain several high-confident parallel patterns for pattern clustering. A lexical patterns often contains a short and simple sequence of words. Consequently, it can be translated by a machine translation system with high accuracy. Moreover, with the verification scheme (will be described later in Section 5.2), we can filter out incorrect translation results and keep only correctly translated patterns.

On the other hand, the method proposed by Davidov and Rappoport [96] starts from pattern clusters in the source language to achieve some representative entity pairs that hold the relation in the source language. It then translates these entity pairs into the target language (using dictionary lookups) and then infers the translated lexical patterns from the contexts in which the translated entity pairs occur. Consequently, the method in [96] *indirectly* translates lexical patterns. A future research direction would be to use those indirectly translated lexical patterns to verify the result of the pattern clustering algorithm proposed in this work to further improve the accuracy while recognizing semantically similar lexical patterns across languages.

It is noteworthy that although the proposed method for cross-lingual latent relational search in this work is mainly for retrieving entities, it directly supports sentence retrieval in multiple languages. Consequently, the results of a cross-language latent relational search query also contain supporting sentences, which refer to similar semantic relations in two different languages. In many cases, these sentences are translations or paraphrases of each other. This indicates that, using the result of this work, we can build parallel corpora or support human translators to find example usages at sentence level in the target language.

Chapter 4

Retrieval Model for Monolingual Latent Relational Search

In this chapter, we first propose the retrieval model for monolingual latent relational search in Section 4.1. Next, in Section 4.2, we propose a method for extracting entity pairs and relations from text corpora, such as the Web, to create an index for latent relational search. The semantic relations between two entities in an entity pairs are represented by lexical patterns of the context surrounding the two entities. We describe the lexical pattern clustering method to recognize paraphrased lexical patterns in Section 4.3. The relation extraction and clustering process result in an index model, which will be presented in Section 4.4. We then present in detail the retrieval and ranking method in sections from 4.5 to 4.8. In particular, we discuss the algorithm for measuring the relational similarity between two entity pairs in Section 4.6. We present a prototype system that implements the proposed model in Section 4.9. Finally, we evaluate the proposed method using an ideal corpus in Section 4.10.

4.1 Retrieval Model

An information retrieval model is a model for retrieving and ranking relevant documents of a query. A retrieval model specifies the method for representing documents, descriptive features (such as index terms), queries and the relationships between these concepts. There are different retrieval models that support different types of queries [11]. For example, the *Boolean model* represents each document as a set of terms (words). Each query in this model is then a Boolean expression of terms (e.g., "university AND Tokyo"). The documents that match the above query must contain both of the terms in the query. There is another important model for information retrieval: the *Vector Space Model* (VSM) [97]. Each document in the VSM is represented as a vector of term frequencies (or other measures that reflect the importance of each term in the document, such as the TF-IDF measure [97]). Each query is considered as a small document, and is also represented as a vector of term frequencies. The similarity between a query and a document is then defined as the cosine of the two corresponding vectors. The relevant documents can be ranked by this similarity.

In latent relational search, each entity pair can be considered as a "document" (i.e., the target of the retrieval process). Moreover, the input source entity pair can be considered as a query. The goal of the entity retrieval process in latent relational search is to retrieve "'documents" (entity pairs) that are "relevant" to the query, as can be seen in other information retrieval models [98, 97]. The term "relevant document" in latent relational search refers to a document (a candidate entity pair) that has high relational similarity

with the input source entity pair. Consequently, a retrieval model for latent relational search must define the following:

- How a "document" is represented. In other words, how the semantic relation in each entity pair is represented.
- What features are used to characterize a "document" (an entity pair) or a query. What is the format of a feature vector.
- How to retrieve a set of relevant documents (i.e., entity pairs that contain the candidate answers) for a query.
- How to define the degree of relevance and how to rank the candidate answers.

Using lexical patterns to represent the semantic relation between two entities yields high precision in measuring the relational similarity between two entity pairs [27, 30, 20], extracting synonyms, antonyms [49] and question answering [59]. Following these previous studies, we represent an entity pair ("a document") by a feature vector, in which each dimension corresponds to a lexical pattern extracted from the context surrounding the two entities in the entity pair. We consider all occurrences of an entity pair in a same sentence, and from the sentence, we extract lexical patterns of the window of text surrounding the two entities in the pair. These lexical patterns represent the context of the entity pair and can be used to characterize the semantic relation between the two entities in the entity pair. Therefore, each "document" (as well as each query) in the proposed model is characterized by a set of lexical patterns and the elements of the feature vector for an entity pair (a "document" or a query) are derived from the numbers of co-occurrences between the entity pair and the lexical patterns. This model is similar to the Vector Space Model (VSM) [97], but instead of indexing individual terms (or words) and document IDs as seen in traditional document retrieval systems [99], we index lexical patterns and entity pairs. This reflects the main difference between a latent relational search engine and a traditional keyword-based search engine: the retrieval process in latent relational search is based on the semantic relations between entities, rather than the occurrences of terms in (text) documents. Because computing and representing the semantic relation between two entities are much more difficult than computing term statistics in a text document, an index for a latent relational search engine is much more complex than an index of a keyword-based search engine using the Vector Space Model. Therefore, creating an efficient index is a challenge that one must address to achieve a high speed latent relational search engine.

The entity retrieval and ranking process of latent relational search is depicted in Figure 4.1. The first goal of latent relational search (retrieving and ranking the answer entity list L, as stated in Section 2.3.1) can be divided into two sub-goals: entity filtering and entity ranking. Each sub-goal can be expressed by a function. Therefore, the "entity filtering function" is for retrieving candidate answers which will be elements of the list L, and the "entity ranking function" is for ranking these candidates to achieve the final order in the list L.

The entity filtering function is a function which takes as input the query q = {(A, B), (C, ?)} and an entity D and outputs a binary value, 1 or 0. If the function outputs 1, then the input entity D is supposed to be relevant to the query q, otherwise the entity D is not relevant to the query q. If we denote the set of all entities as $\mathbb{E}$ and the set of all queries as $\mathbb{Q}$ then the entity filtering function $\mathbf{F}_e$ can be described as follows:

$$\mathbf{F}_e : \mathbb{Q} \times \mathbb{E} \rightarrow \{0, 1\} \tag{4.1}$$

$$\mathbf{F}_e(q, D) = \mathbf{F}_e(\{(A, B), (C, ?)\}, D) = \begin{cases} 1 & \text{if } D \text{ is relevant to } q \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

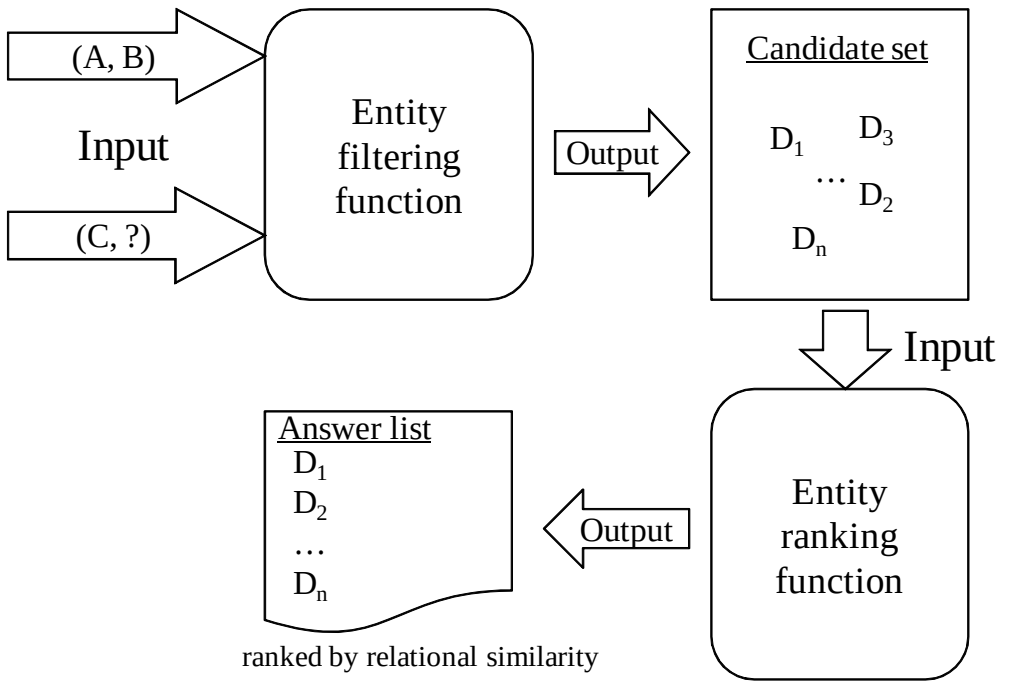


Fig. 4.1. The candidate retrieval and ranking process of latent relational search

The candidate answer set $\mathbb{S}$ for a query $q = \{(A, B), (C, ?)\}$ is a subset of $\mathbb{E}$ and can be defined as follow:

$$\mathbb{S}(q) = \{D \in \mathbb{E} \quad | \quad \mathbf{F}_e(q, D) = 1\} \tag{4.3}$$

The entity ranking function is a function that takes as input a query $q = \{(A, B), (C, ?)\}$ and a candidate entity D in the candidate set $\mathbb{S} \subseteq \mathbb{E}$ and outputs an integer, which is the rank (index) of the entity D in the final result list L. The output of the ranking function $\mathbf{R}_e$ must satisfy the condition that, for an arbitrary entity $D \in \mathbb{S}$, the rank of this entity is an integer larger than 0 and smaller than or equal to the size of the candidate set $\mathbb{S}$. Moreover, for each entity pair (D_i, D_j), if the relevance score (based on relational similarity) between (A, B) and (C, D_i) is larger than that between (A, B) and (C, D_j), then the rank of D_i must be smaller than the rank of D_j. We denote the relevance score (for ranking) between (A, B) and (C, D) as $\text{Rel}((A, B), (C, D))$. The score $\text{Rel}((A, B), (C, D))$ can be defined by the relational similarity between (A, B) and (C, D) and other combinations such as between (B, A) and (D, C). The above conditions can be written as follows:

$$\mathbf{R}_e : \mathbb{Q} \times \mathbb{E} \to \mathbb{N} \qquad (\mathbb{N} \quad \text{is the set of natural numbers}) \tag{4.4}$$

$$1 \leq \mathbf{R}_e(q, D) = \mathbf{R}_e(\{(A, B), (C, ?)\}, D) \leq |\mathbb{S}| \tag{4.5}$$

$$\forall D_i, D_j \in \mathbb{S} : \text{Rel}((A, B), (C, D_i)) > \text{Rel}((A, B), (C, D_j)) \Rightarrow \mathbf{R}_e(q, D_i) < \mathbf{R}_e(q, D_j) \tag{4.6}$$

If we combine the result of the entity filtering function $\mathbf{F}_e$ and the entity ranking function $\mathbf{R}_e$, we have the final rank of an entity in the result list (we assume that a rank of zero is an invalid rank and is outside the result list):

$$\text{Rank}(q, D) = \mathbf{F}_e(q, D) \times \mathbf{R}_e(q, D) \tag{4.7}$$

The above equation expresses the retrieval and ranking model for latent relational search. There might be several functions $\mathbf{F}_e$ and $\mathbf{R}_e$ that satisfy the conditions in equations 4.1 – 4.6. Each of the combination of these functions corresponds to an implementation of the retrieval system to achieve the ranking model in Equation 4.7. The main problem is to reduce the complexity of these functions and to accurately calculate the relational similarity between two entity pairs. In this chapter and in Chapter 5, we propose methods to achieve both of these goals: to perform retrieval in high speed and to achieve high precision in ranking results.

4.2 Entity and Relation Extraction

To perform latent relational search in high speed, we propose a method for building an index on which we can implement the ranking model in the Equation 4.7. Unlike other latent relational search systems [8, 9, 4], the proposed search engine has its local index. Therefore, it does not require to query an external keyword-based search engine (such as Google[*1]) while processing a latent relational search query. This makes the query processing time of the proposed system fast enough for real-world users' search sessions.

We adapt the previously proposed relation extraction algorithms [30, 29] to improve the performance of latent relational search. These algorithms represent the relations between two entities in an entity pair by lexical patterns of the context surrounding the two entities.

We use a single-pass extraction method [6] to extract entity pairs and lexical patterns from a given multi-lingual text corpus (a corpus that might contain documents in multiple languages, but not necessary an aligned or parallel corpus). This extraction method traverses each document in the text corpus only once and it requires only a single-pass to process a document. Therefore, it is scalable to large corpora. Note that even when building an index for a monolingual latent relational search engine, we must consider the language of the document, because the lexical pattern extraction algorithm might be different for each language. A *monolingual* latent relational search engine that is capable of processing *monolingual* latent relational search queries in *multiple languages* (e.g., the queries {(Tokyo, Japan), (?, France)} and {(東京, 日本), (?, フランス)}) must therefore identify the language of a document before performing the relation extraction process. Consequently, to extract entity pairs from a document in the text corpus, we first identify the language of the document. Methods with high accuracy in linear time, have been proposed in previous work on language identification in a document [100, 101, 102].

In our case, because we only experiment with Japanese and English documents, we can identify the language of a document by counting the number of Japanese characters in the document. If this number is larger than 5% of the total number of characters then we assume that the document is in Japanese, otherwise we assume that it is in English. We set the threshold to 5% because many English documents (such as the English Wikipedia article about "Tokyo"[*2]) contain Japanese characters, but the number of Japanese characters is not large while comparing with the total number of characters. Japanese characters are in a completely different writing system from English. Therefore we can identify a Japanese character by simply checking the code of the character. Algorithm 1 shows the pseudo-code of the language identification process that we employed.

After identifying the language of a document, we use a Sentence Boundary Detector to split the document into sentences. Japanese language has an unambiguous sentence-ending marker, the Unicode character "u3002" (the "*Ku-ten*" symbol). Therefore, in

[*1] http://www.google.com
[*2] http://en.wikipedia.org/wiki/Tokyo

Algorithm 1 English/Japanese document language identification

Input: a document d, written in English or Japanese
Output: the language of the document

```
1: jpCharCount ← 0
2: totalCharCount ← 0
3: /* Count the number of Japanese characters in d */
4: for each character ch ∈ d do
5:    c ← charCode(ch)
6:    /* Check if the character is in Japanese character ranges */
7:    if ( (0x3000 ≤ c) and (c ≤ 0x9FA5) ) or
      ( (0xFF01 ≤ c) and (c ≤ 0xFF9F) ) then
8:       jpCharCount ← jpCharCount + 1
9:    end if
10:   totalCharCount ← totalCharCount + 1
11: end for
12: if (jpCharCount/totalCharCount) ≥ 0.05 then
13:   return Japanese
14: else
15:   return English
16: end if
```

Japanese documents, a sentence boundary can be identified using this symbol. However, there is not an unambiguous sentence boundary marker in English, as the period symbol (".") can be used for abbreviations (such as in U.S.) and other purposes. An English sentence boundary detector extracts all candidate sentence boundaries (for example, the period symbol "." or the question mark "?") and recognizes which are used for separating sentences, which are used for other purposes. In our algorithm, we use the Sentence Boundary Detector in the Stanford POS Tagger[*3] to identify sentence boundaries in an English document.

We then use the Stanford POS Tagger and Stanford Named Entity Recognizer[*4] to split an English sentence into words and recognize named entities containing in the sentence. We use the MeCab POS Tagger[*5] for Japanese word segmentation and named entity recognition. While Open Information Extraction (OpenIE) systems require deep linguistic analysis such as dependency parsing [5] or co-reference resolution [103], the proposed system requires only shallow linguistic processing tools, namely, Sentence Boundary Detectors, POS Taggers and Named Entity Recognizers. This helps us to achieve short pre-processing time and scale the proposed method to large datasets.

After splitting a sentence into tokens (words) and recognizing named entities, we extract all named entity pairs that preserve the order of each entity in the sentence. For example, from the sentence "Microsoft acquired San Francisco based company Powerset for $100M", we extract three entity pairs (Microsoft, San Francisco), (San Francisco, Powerset) and (Microsoft, Powerset). Other combinations such as (Powerset, Microsoft) are not extracted because they do not preserve the order of the entities. We only extract entity pairs that preserve the order of each entity in a sentence because the relation between two entities in a pair might not be symmetric, and hence the relation in the reversed pair is not described in the sentence. In the above example, the sentence only describes the

[*3] http://nlp.stanford.edu/software/tagger.shtml
[*4] http://nlp.stanford.edu/software/CRF-NER.shtml
[*5] http://mecab.sourceforge.net/

Table. 4.1. The lexical pattern extraction process for the entity pair (Sarkozy, France).

POS, NE tagged sentence	Sarkozy who is the current president of France was born in Budapest.
Stemming	Sarkozy who is the current presid of France wa born in Budapest.
Variable substitution	X who is the current presid of Y wa born in Budapest.
Window extraction	X who is the current presid of Y wa born in
Generating n-grams	X who is current presid of Y, X * is current presid of Y, X * presid * Y, X * is current presid * Y, X * current * Y, ...

relation between Microsoft and Powerset, which is different from that between Powerset and Microsoft.

It is important to note that, we extract all entity pairs (that preserve the order) from a sentence, irrespective of the relations that those pairs might hold. Therefore, the proposed method does not require the relation types to be given in advance. We will filter-out noisy entity pairs, that include misspelling etc. using a frequency filter.

The most important task in latent relational search is calculating the relational similarity between two entity pairs. For example, we must recognize that the semantic relation between Tokyo and Japan is highly similar to that between Paris and France. To accomplish this task, one must represent the relation between two entities in an entity pair by some features. To capture the semantic relation between two entities, we extract lexical patterns from the contexts in which those entities co-occur. Using lexical patterns to represent semantic relations between two entities yields high precision in measuring the relational similarity between two entity pairs [27, 30, 20], extracting synonyms, antonyms [49] and question answering [59]. Therefore, we also represent the semantic relations between two entities in an entity pair by a feature vector whose elements are the number of co-occurrences (or the Point-wise Mutual Information (PMI) values) between the entity pair and their lexical patterns. We can then define the relational similarity between two entity pairs as the cosine similarity between their feature vectors.

We propose a method to adapt the lexical pattern extraction algorithms in previous work on measuring relational similarity between two entity pairs [27, 30, 20] to the task of latent relational search. In latent relational search, when two entity pairs actually hold similar semantic relations, we must make the relational similarity between them significantly higher than when they do not actually hold similar relations. That is, we want the two relationally similar entity pairs share many common lexical patterns so that the cosine between their feature vectors is large. Consequently, we propose the following algorithm for lexical pattern extraction between two entities in a sentence, as shown in Table 4.1.

The first step of the algorithm is to identify the positions of the two entities in the pair that we need to extract lexical patterns. In Table 4.1, the two entities are Sarkozy and France.

The second step is to stem all words other than named entities in the sentence. We use the Porter Stemmer[*6] for stemming English words. For a Japanese word, the MeCab POS tagger provides the primitive form of the word so we can use this primitive form. We found that stemming is an important step to improve both recall and precision of latent

[*6] http://nltk.googlecode.com/svn/trunk/doc/api/nltk.stem.porter-module.html

relational search. We will compare the experimental results with and without stemming later in Section 4.10.4. Intuitively, stemming eliminates the differences between inflected forms of a word. Because different inflected forms of a word are considered as equal after stemming, various lexical patterns (that contains inflected forms of a same word) are considered as equal. This makes the association between a stemmed lexical pattern and the semantic relation that it represents stronger. Moreover, in monolingual latent relational search, the recall level will be improved after stemming, because stemming makes the probability that two entity pairs share some common lexical patterns (which are not the same before stemming) higher.

In the next step, we replace the two entities with their symbolic representation, to make the extracted lexical patterns independent from the entity pairs with which they co-occur. In Table 4.1, the first entity is replaced with the variable X, the second entity is replaced with the variable Y. We then consider only a window of text surrounding the entity pair for extraction because we want to prevent an explosion in the number of extracted lexical patterns. Specifically, we consider the following sub-string (window) of the sentence:

$$b_1 b_2 ... b_k X w_1 w_2 ... w_m Y a_1 a_2 ... a_p$$

That is, the sub-string contains of k words before X, the variable X, the gap between X and Y (the gap contains m words, $m \leq M$), the variable Y and p words after Y. We only consider two entities for lexical pattern extraction if the distance between them is not larger than M words. This is because the probability that two entities hold some semantic relations is likely low if the distance is too far. For example, if $k = p = 3$ then the sub-string is "X who is the current president of Y was born in", as shown in Table 4.1. Finally, we generate all n-grams ($n \leq M + 2$) from the above sub-string. We allows n-grams with $n = (M + 2)$ because we want to take the complete pattern $X w_1 w_2 ... w_m Y$ when $m = M$. We omit all n-grams that contain only b_i or contain only a_i (e.g., $b_1 b_2 b_3$ or $a_3 a_4 a_5$). This is because these n-grams contain neither X nor Y, which might be irrelevant to the entity pair. For n-grams that contain only w_i (i.e, $w_i w_{i+1} ... w_j$), we change them into the form "$X * w_i w_{i+1} ... w_j * Y$". By this modification, we can mark the relative positions of the words in the n-grams against the entities (e.g., these words are inside the gap between X and Y). Similarly, for n-grams that do not contain Y (e.g, $b_k X w_1 w_2$), we change them into $b_k X w_1 w_2 * Y$. And finally, for n-grams that do not contain X, we append "$X*$" before them: $X * w_i w_{i+1} ... w_m Y a_1$. Only n-grams with at least one content word (i.e., not stop word and not the variable X or Y) are selected. This makes the probability that the selected n-grams represent a clear semantic relations between the two entities higher. For example, we generate lexical patterns as shown in the last line of Table 4.1.

Although our lexical pattern extraction algorithm is similar to the algorithms in previous research [20, 30, 27], we make two important modifications to adapt the algorithm to latent relational search. First, we eliminate differences between inflected forms of a word by stemming the input sequence. Second, we allow sub-sequences that neither contain X nor Y. Instead, we append the prefix "$X*$" or sub-fix "$*Y$" to the patterns for discriminating patterns' positions when comparing them. Allowing sub-sequences that neither contain X nor Y makes the probability that two entity pairs have common lexical patterns higher because we do not require a complete match between the sequences in the gap of each pair. For example, consider the two sentences: "Obama is the 44th and current president of the U.S" and "Sarkozy is the current president of France". If we allow the pattern "current president of" (i.e., we generate the pattern "X * current president of * Y") then we have a common pattern between two pairs *(Obama, U.S)* and *(Sarkozy, France)*.

During the pattern extraction phase, we record (or update) the frequency of each lexical pattern for filtering out low frequency patterns which are too specific (e.g., "offici: X acquir San Francisco * Y") or noisy. Therefore, we have a set of entity pairs (and their

Table. 4.2. The pattern vs. entity pair matrix $\mathbf{M}$

Pattern vs. Entity pair	(Google, YouTube)	(グーグル, ユーチューブ)†	(Microsoft, Powerset)	(日産, 横浜)‡	(Yahoo, Sunnyvale)
X acquires Y	80	0	70	0	0
X が Y を買収	6	75	2	0	0
X bought Y	67	0	60	0	0
X is headquartered in Y	0	0	0	0	105
XはYに本社を置く	0	0	0	90	0

Note: Each Japanese lexical pattern has the same meaning with the corresponding pattern above it. † meaning (Google, YouTube) ‡ meaning (Nissan, Yokohama)

frequencies) in the index. Each entity pair is associated with a list of lexical patterns (and patterns' frequencies) in which the pair co-occurs. We denote $\mathbf{P}(w)$ as the set of all lexical patterns with which the entity pair w co-occurs:

$$\mathbf{P}(w) = \{p_1, p_2, \ldots, p_n\} \tag{4.8}$$

Moreover, to efficiently retrieve entity pairs that share at least one lexical pattern with the source entity pair, we also store an inverted index from a lexical pattern to the set of entity pairs that the pattern co-occurs. We denote $\mathbf{W}(p)$ as the set of all entity pairs with which the pattern p co-occurs:

$$\mathbf{W}(p) = \{w_1, w_2, \ldots, w_m\} \tag{4.9}$$

We denote the number of co-occurrences between the entity pair w_i with the pattern p_j in a same sentence as $\mathrm{f}(w_i, p_j)$. The entity pair frequency vector $\mathbf{\Phi}(p)$ of a lexical pattern p is then defined as:

$$\mathbf{\Phi}(p) = (\mathrm{f}(w_1, p), \mathrm{f}(w_2, p), \ldots, \mathrm{f}(w_m, p))^{\mathrm{T}} \tag{4.10}$$

Similarly, the pattern frequency vector of an entity pair w is defined as:

$$\mathbf{\Psi}(w) = (\mathrm{f}(w, p_1), \mathrm{f}(w, p_2), \ldots, \mathrm{f}(w, p_n))^{\mathrm{T}} \tag{4.11}$$

At this step, we can create a matrix $\mathbf{M}$ of co-occurrences between lexical patterns and entity pairs. The value of each element $\mathbf{M}_{ij}$ of the matrix $\mathbf{M}$ can be the number of co-occurrences between the pattern p_i and the entity pair w_j, as shown in Table 4.2. We can also use the point wise mutual information (PMI) between p_i and w_j as the value of $\mathbf{M}_{ij}$. PMI has been successfully used for assessing the strength of the association between an entity pair and a lexical pattern in previous research [73, 69]. We experimentally compare the method using numbers of co-occurrences and the method using PMIs in Section 4.10.3 to show that PMI actually improves the performance of the proposed latent relational search system. Following Pantel and Ravichandran [73], we define the PMI between an entity pair w and a lexical pattern p as follows:

$$\mathrm{pmi}(w, p) = \left(\frac{\mathrm{f}(w, p)}{\mathrm{f}(w, p) + 1} \times \frac{\min(\mathrm{f}(w), \mathrm{f}(p))}{\min(\mathrm{f}(w), \mathrm{f}(p)) + 1} \right) \log \left(\frac{\frac{\mathrm{f}(w,p)}{N}}{\frac{\mathrm{f}(w)}{N} \frac{\mathrm{f}(p)}{N}} \right) \tag{4.12}$$

where $\mathrm{f}(w, p)$ is the number of co-occurrences between w and p, whereas, N is the total number of co-occurrences between all entity pairs and all lexical patterns. PMI is known

to has a bias towards infrequent entity pairs and lexical patterns (i.e., an infrequent pattern has a higher PMI than high frequency patterns). Therefore, the first factor in Equation 4.12 is to prevent this bias [73]. When using PMI, the elements of the vectors $\mathbf{\Phi}$ (in Equation 4.10) and $\mathbf{\Psi}$ (in Equation 4.11) are also changed from $\mathrm{f}(w, p)$ into $\mathrm{pmi}(w, p)$ and these vectors are called the *entity pair PMI vector* and the *lexical pattern PMI vector*, respectively.

It is worth noting that the index that we build is a multi-lingual index. We do not differentiate between entity pairs that appear in a Japanese document and those in an English document. Moreover, lexical patterns that are associated with an entity pair might be in different languages. Therefore, we call this indexing method "multi-lingual entity pair and pattern indexing". This indexing method enables a monolingual latent relational search engine to process queries of multiple languages (such as Japanese monolingual queries and English monolingual queries) using the same algorithm. Moreover, in many situations, a language other than English (e.g., Japanese) also uses an identical orthography to represent an entity name. For example, in many sentences, Japanese write the name of the Google Inc. as "Google", instead of the Katakana expression グーグル. Our indexing method exploits this phenomenon to capture the semantic similarity between lexical patterns in different languages. Because in several Japanese sentences, "グーグル" is also written as "Google" and "ユーチューブ" is also written as "YouTube", we can extract some Japanese lexical patterns of the pair (Google, YouTube) and associate them with other English patterns of the same pair. Because these patterns express the semantic relations of the same entity pair (Google, YouTube), there is a high probability that they are semantically similar or related. Because there are many semantic relations that an entity pair might hold, we can not assume that all lexical patterns of an entity pair are semantically similar. For example, the pair (Google, YouTube) has the relation "X acquired Y", but it also holds the relation "X operates Y as a service". However, using the co-occurrences of an entity pair with lexical patterns in different languages we can perform cross-language lexical pattern clustering, which groups semantically similar patterns in multiple languages into a pattern cluster, as described in Section 5.3.2.

4.3 Recognizing Paraphrased Lexical Patterns in the same Language

Even when lexical patterns are stemmed, two relationally similar entity pairs often share only a small number of identical lexical patterns because a relation can be expressed in several ways in a natural language (e.g., "X acquired Y" and "X bought Y" are semantically similar). For example, the pair *(Google, YouTube)* has a high relational similarity with the pair *(Microsoft, Powerset)*, but the relation in the first pair might be expressed as "Google has acquired YouTube for $1.6 billion." while the relation in the second pair might be expressed by a sentence such as "Microsoft bought Powerset for $100M plus". Therefore, the lexical patterns that express these relations do not match, if we simply compare the surface forms of the lexical patterns. This leads to the data sparseness problem concerning matched lexical patterns between two entity pairs. That is, even in two entity pairs with high relational similarity, the number of orthographically matched lexical patterns between them is very small. To alleviate this problem, we must recognize lexical patterns that are semantically similar. For example, if we recognize that the two patterns "X acquired Y" and "X bought Y" are semantically similar, we can consider them as equal when we compute the similarity between two entity pairs (Google, YouTube) and (Microsoft, Powerset).

To recognize semantically similar patterns, we rely on the *Distributional Hypothesis* [104]. The Distributional Hypothesis states that, words that occur in the same contexts tend to have similar meanings [104]. That is, if the set of words that the word w_1 co-occurs with is similar to that of the word w_2, then there is a high probability that w_1 and w_2 have similar meanings. We extend this hypothesis to lexical pattern level: lexical patterns that co-occur with the same contexts tend to have similar meanings, to recognize similar lexical patterns. Specifically, we assume that lexical patterns that co-occur with similar sets of entity pairs have similar meanings. This method has been successfully used in previous studies to recognize semantically similar lexical patterns [105] or to improve precision of relational similarity measuring algorithms [30, 20].

Under the assumption that lexical patterns that co-occur with similar sets of entity pairs have similar meanings, we can define a similarity measure between two lexical patterns p and q, using their entity pair frequency (or PMI) vectors $\mathbf{\Phi}(p)$ and $\mathbf{\Phi}(q)$:

$$\text{sim}_{\text{VSM}}(p, q) = \cos(\mathbf{\Phi}(p), \mathbf{\Phi}(q)) = \frac{\sum_i (f(w_i, p) \cdot f(w_i, q))}{\sqrt{\sum_i f^2(w_i, p)} \sqrt{\sum_i f^2(w_i, q)}} \tag{4.13}$$

We can define the similarity between two lexical patterns using the cosine because we infer from the distributional hypothesis [104]: patterns that share many entity pairs that they co-occur with are semantically similar. We call this similarity as sim_{VSM} because it uses the vector space model to compute the similarity. For example, the two patterns "X acquires Y" and "X buys Y" often share a large entity pair set such as {*(Google, YouTube)*, *(Microsoft, Powerset)* ...}. Consequently, the cosine between their entity pair frequency (or PMI) vectors is large.

We then use a sequential pattern clustering algorithm proposed by Bollegala et al. [20] to group similar patterns into pattern clusters. We call the lexical pattern clustering algorithm the "monolingual lexical pattern clustering" (**MLPC**) algorithm. Algorithm 2 shows the pattern clustering procedure in this clustering method. For each pattern, the algorithm finds the cluster whose centroid has maximum cosine similarity with the pattern (line 6–12). If this similarity is above a pattern clustering similarity threshold θ_1 then the pattern is added to the cluster (line 13, 14), otherwise, the pattern forms a new singleton cluster itself (line 15–17). The appropriate value for the pattern clustering similarity threshold θ_1 will be determined by experiments.

To filter out patterns that are specific to an entity pair, such as "now offici: X * San Francisco * Y", and to reduce the time for pattern clustering, we omit rare patterns (i.e., patterns that appear only once) when clustering patterns.

The pattern clustering method in Algorithm 2 is a lightweight clustering algorithm, in the sense that it has less amortized time complexity than other clustering algorithms. For example, we can use the SLINK [106] or CLINK [107] algorithms to cluster lexical patterns into pattern hierarchies, without specifying the number of lexical pattern clusters beforehand. We can then use these hierarchies to achieve a set of lexical pattern clusters. However, these algorithm require $O(n^2)$ or $O(n^3)$ time complexity (n is the number of lexical patterns to be clustered), which is hard to perform with a very large number of lexical patterns. On the other hand, the procedure in Algorithm 2 has the time complexity of $O(n\log n + |K|n)$, in which $|K|$ is the size of the final output cluster set. The first term, $n\log n$ is due to the sort operation in line 2 of Algorithm 2. The sort step is to choose patterns with high frequencies to process in an early stage of the clustering algorithm, to form lexical pattern clusters that contain high confident lexical patterns. However, according to our recent research [108], lexical patterns with high numbers of occurrences might not be representative lexical patterns for a relation. Therefore, the sorting step is optional in the algorithm and can be omitted. The second term of the time complexity is

Algorithm 2 Monolingual Lexical Pattern Clustering (MLPC)

Input: pattern set $\wp$, similarity threshold $\theta_1 > 0$
Output: pattern cluster set $\mathbf{K}$

1: $\mathbf{K} \leftarrow \{\}$
2: sort($\wp$)
3: **for each** pattern $p \in \wp$ **do**
4: $maxClus \leftarrow$ NULL
5: $maxSim \leftarrow -1$
6: **for each** pattern cluster $c \in \mathbf{K}$ **do**
7: $cpSim \leftarrow \text{sim}_{\text{VSM}}(p, \text{centroid}(c))$
8: **if** $cpSim > maxSim$ **then**
9: $maxSim \leftarrow cpSim$
10: $maxClus \leftarrow c$
11: **end if**
12: **end for**
13: **if** $maxSim \geq \theta_1$ **then**
14: $maxClus$.append(p)
15: **else**
16: $newClus \leftarrow \{p\}$
17: $\mathbf{K} \leftarrow \mathbf{K} \cup \{newClus\}$
18: **end if**
19: **end for**
20: **return K**

the total number of operations required by the for-loop from line 3 to line 19. Normally, $|K|$ is very small compared to the number of lexical patterns n, because a cluster contains a large number of lexical patterns. Consequently, the time complexity of this algorithm is much smaller than $O(n^2)$. In case when $|K| \simeq O(\log n)$, the time complexity of the algorithm will be reduced to $O(n \log n)$. Therefore, using this algorithm, we can perform clustering in a reasonable time even with very large number of lexical patterns.

4.4 Index Model for Monolingual Latent Relational Search

After the pattern clustering step, we can build an index that contains information about entities, lexical pattern clusters and relations between them as shown in Figure 4.2. Figure 4.2 is the logical (conceptual) index model of latent relational search. The model is an undirected graph, with two different types of nodes, representing entities and lexical pattern clusters. Each ellipse node represents an entity in the index. Each lexical pattern cluster (rectangle node) contains semantically similar lexical patterns. An edge must connect an entity with a lexical pattern cluster. If an entity pair (A, B) holds a semantic relation, there is a path of length two from A to B and vice versa, in which all edges are of the same type (i.e., the dash-types of the edges in Figure 4.2 are the same). The path can be represented as $\langle (A, P), (P, B) \rangle$, in which A, B are ellipse nodes, P is a rectangle node, representing a lexical pattern cluster. (A, P) and (P, B) are edges of the same dash-type. We call a path of length two from A to B that satisfies this condition as a "*relational path*" of the entity pair (A, B). This path indicates that one of the semantic relations between A and B can be represented by one or more lexical patterns in the pattern cluster P.

We denote the graph representing the index as $\mathcal{G} = <\mathcal{V}_G, \mathcal{E}_G>$, in which $\mathcal{V}_G$ is the set of all vertices and $\mathcal{E}_G$ is the set of all edges. Moreover, we denote the set of all lexical

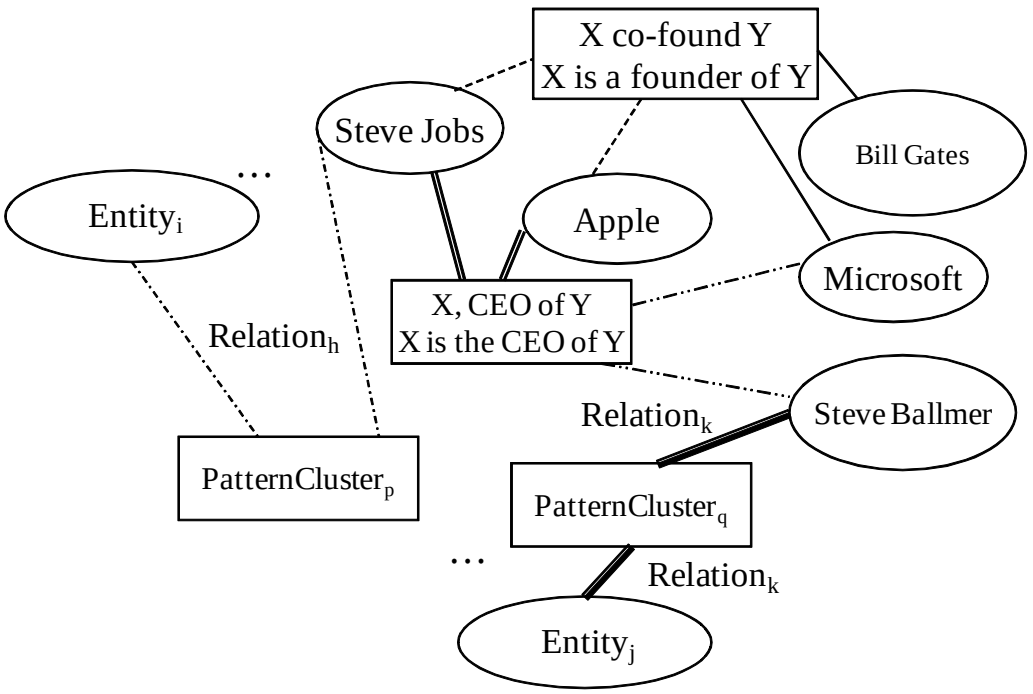


Fig. 4.2. The index model for latent relational search. Each entity is represented by an ellipse, each lexical pattern cluster is represented by a rectangle. Two entities in a relation are connected by the same dash-type lines through one or more lexical pattern clusters.

pattern clusters as $\mathbb{P}_c$ and the set of all entities in the index as $\mathbb{E}$. We then have the following conditions held:

$$\mathcal{V}_G = \mathbb{P}_c \cup \mathbb{E} \tag{4.14}$$

$$((A, P) \in \mathcal{E}_G) \wedge (A \in \mathbb{E}) \Rightarrow P \in \mathbb{P}_c \tag{4.15}$$

$$((P, A) \in \mathcal{E}_G) \wedge (P \in \mathbb{P}_c) \Rightarrow A \in \mathbb{E} \tag{4.16}$$

The first condition (in Equation 4.14) states that the set of vertices in the graph $\mathcal{G}$ is made up of two disjoint sets of vertices: the set $\mathbb{P}_e$ representing the set of all lexical pattern clusters and the set $\mathbb{E}$ representing all entities in the index. The conditions in Equation 4.15 and Equation 4.16 state that each edge in the graph $\mathcal{G}$ must connect a node in $\mathbb{E}$ with a node in $\mathbb{P}_c$ (i.e., the graph is a bipartite graph between two disjoint set $\mathbb{E}$ and $\mathbb{P}_c$).

Each dash-type is can be represented as an integer (label). Therefore, each edge of the same dash-type in $\mathcal{E}_G$ is assigned a same label. Suppose that the function $\mathrm{dt}(e)$ returns the dash-type of the edge $e \in \mathcal{E}_G$. The set of all relational paths between two entities A and B ($A, B \in \mathbb{E}$) is then defined as follows:

$$\mathrm{relpath}(A, B) = \{\langle (A, P), (P, B)\rangle \mid (A, P), (P, B) \in \mathcal{E}_G \wedge \mathrm{dt}((A, P)) = \mathrm{dt}((P, B))\} \tag{4.17}$$

The set of all lexical pattern clusters in the set of relational paths between A and B can be represented as follows:

$$\mathrm{rp}(A, B) = \{P \in \mathbb{P}_c \quad | \quad \langle (A, P), (P, B)\rangle \in \mathrm{relpath}(A, B)\} \tag{4.18}$$

Note that there might be several relational paths of a pair (A, B) because a semantic relation can be stated by different paraphrases and an entity pair might hold different semantic relations. For example, in Figure 4.2, *Steve Jobs* is linked to *Apple* by two lexical pattern clusters: "X co-found Y" and "X, CEO of Y". Therefore, there are two different relational paths from *Steve Jobs* to *Apple*.

An important property of relational paths is that, there is a relational path from an entity A to an entity B that passes through a lexical pattern cluster P even if there is only one lexical pattern $p \in P$ that co-occurs with the pair (A, B). The rest of lexical patterns in P (except p) might not co-occur with the pair (A, B) in the text data. However, because we recognize that these lexical patterns are semantically similar with p, we can assume that these lexical patterns also co-occur with (A, B), as p does. By following this assumption, we can alleviate the data sparseness problem concerning exactly matched lexical patterns between two entity pairs. For example, suppose that there is a pair (A', B') that co-occurs with the lexical pattern $q \in P$, but does not co-occur with the lexical pattern p. Because (A', B') co-occurs with $q \in P$, there is a relational path from A' to B' that passes through P. In this case, even q does not co-occur with the pair (A, B), the pairs (A, B) and (A', B') share the lexical pattern cluster P in one of their relational paths. Intuitively, the lexical patterns p and q are considered as equal (and are represented by the same lexical pattern cluster P), when comparing the semantic relations in (A, B) and in (A', B'). Consequently, the difference in surface forms of these lexical patterns is absorbed by using the lexical pattern cluster P.

A physical (actual) index for latent relational search must represent all of the information in the logical (conceptual) index, including entities, lexical patterns, lexical pattern clusters and relational paths between them. We must map these concepts into actual physical storage to create the physical index for a latent relational search engine. This is an implementation issue, which will be discussed in Section 4.9.

4.5 Entity Filtering Function

As described in Equation 4.7, the first step to achieve a ranked result list is to implement the entity filtering function $\mathbf{F}_e(q, D)$. This function returns a binary value which indicates the entity D is relevant to the query $q = \{(A, B), (C, ?)\}$ or not. Theoretically, after applying the entity filtering function on all entities in the entity set $\mathbb{E}$, we obtain a candidate answer entity set $\mathbb{S}(q) \subseteq \mathbb{E}$, as shown in Equation 4.3. However, in practice, applying this function to *all* entities in the set $\mathbb{E}$ is infeasible because the size of the entity set $\mathbb{E}$ might be very large. For example, if the set $\mathbb{E}$ contains millions of entities, then even for a very simple function, the calculation time for all entities in $\mathbb{E}$ would become much larger than the time of a normal user's search session. This implies that applying any function $\mathbf{F}_e$ to all entities in the set $\mathbb{E}$ is an impractical method. Consequently, we must investigate a method to restrict the set of entities before applying the entity filtering function $\mathbf{F}_e$.

To achieve this goal, we propose a heuristic to prune the set of potential candidate entities. The heuristic requires an inverted index from lexical patterns to entity pairs. We must actually store this inverted index because the proposed method requires the entity pair frequency (or PMI) vector $\mathbf{\Phi}$, as shown in Equation 4.10 in the pattern clustering step. Using the entity pair frequency vector $\mathbf{\Phi}$ and the lexical pattern frequency vector $\mathbf{\Psi}$ (in Equation 4.11), we prune the candidate set as follows. Assume that the source entity pair in the query $q = \{(A, B), (C, ?)\}$ is s ($s = (A, B)$). We first enumerate all patterns with which the source pair s co-occurs (i.e., the set $\mathbf{P}(s)$, as defined in Equation 4.8). For each pattern p with frequency above ten in this set, we retrieve all entity pairs which co-occur with p (i.e., the set $\mathbf{W}(p)$, as defined in Equation 4.9). For each retrieved entity

pair, if the pair has the form of (C, X) and has the frequency greater than or equal to five, we add the entity X into the potential candidate set $\Re$:

$$\Re(q) = \Re(\{s, (C, ?)\}) = \bigcup_{p \in \mathbf{P}(s) \wedge \text{freq}(p) \geq 10} \{X | (C, X) \in \mathbf{W}(p) \wedge \text{freq}((C, X)) \geq 5\} \quad (4.19)$$

By this method we can ensure that each candidate pair $c = (C, X)$ (with $X \in \Re$) shares at least one lexical pattern with the source pair s. Moreover, this condition also helps to limit the number of candidate pairs and speed up the candidate retrieving process. It is worth noting that it is not required to calculate any relational similarity to achieve the potential candidate set $\Re$. Instead, the calculation process only requires the index from entity pairs to lexical patterns and the inverted index from lexical patterns to entity pairs to retrieve entity pairs of the form (C, X) that share at least one lexical patterns with the source pair. Algorithm 3 shows the an implementation of the function $\Re(q)$ in Equation 4.19 to retrieve the potential candidate set $\Re(q)$ for a latent relational search query $q = \{(A, B), (C, ?)\}$. In Algorithm 3, the functions GetEntityPair and GetPattern

Algorithm 3 Retrieving a potential candidate set for the monolingual query q

Input: A latent relational search query $q = \{(A, B), (C, ?)\}$
Output: A potential candidate answer set $\Re(q)$

```
1: ℜ ← {}
2: /* Get all lexical patterns that co-occur with (A, B) */
3: P((A, B)) ← GetEntityPair(A, B).patterns()
4: for each pattern p ∈ P((A, B)) do
5:    /* freq returns the total frequency of an object in the index */
6:    if freq(p) ≥ 10 then
7:       W(p) ← GetPattern(p).entityPairs()
8:       for each entity pair w ∈ W(p) do
9:          if freq(w) ≥ 5 and w has form of (C, X) then
10:            ℜ ← ℜ ∪ {X}
11:         end if
12:      end for
13:   end if
14: end for
15: return ℜ
```

look up the index to retrieve the corresponding entity pair and lexical pattern, respectively. If the index is implemented by a hash table, the time complexity of a lookup operation is $O(1)$. Therefore, the time complexity of Algorithm 3 is determined by the for-loop in line 4–14. Suppose that the average number of lexical patterns that co-occur with an entity pair is n_p and the average number of entity pairs that co-occur with a lexical pattern is n_e, we have the following formula regarding the average time complexity of Algorithm 3:

$$\text{T}(\Re) = O(n_p \times n_e) \quad (4.20)$$

Normally, the number of lexical patterns that co-occur with an entity pair is limited by a constant, irrespective of the number of entity pairs in the index. This is because the number of lexical patterns that co-occur with an entity pair is determined by the number of different semantic relations (held in the entity pair) and the number of paraphrases that these relations are referred to. Similarly, the number of entity pairs that co-occur with a lexical pattern is also limited by a constant, irrespective of the number of lexical patterns

in the index. This implies that, the time complexity $T(\Re)$ is a constant, irrespective of the number of lexical patterns and entity pairs. Although this constant might be very large, when the size of the index goes large, the time complexity will not depend on the number of lexical patterns and entity pairs in the index. It is a result of the fact that, the algorithm does not require any global information regarding the entire set of entity pairs or the entire set of lexical patterns. It only requires the information concerning some specific entity pairs and lexical patterns. Consequently, when the index size is larger than a certain level, then the time complexity $T(\Re)$ will be a constant (or will increase very slow, compared to the corpus size). Consequently, the proposed algorithm is effective for very large corpora and indexes.

The candidate set $\Re$ might still contain a huge number of candidate entity pairs. Because sharing at least one lexical pattern does not ensure that the candidate pair is an analogy of the source pair, we must use a relational similarity measure to filter out inappropriate pairs and rank the result set. Therefore, after pruning the potential candidate answer set, we calculate the relational similarity between the source entity pair and each candidate target entity pair. We retain only candidate pairs whose the relational similarities with the source entity pair are above an entity filtering threshold σ in the final candidate set $\mathbb{S}$.

Therefore, the entity filtering function is a binary function which takes a query q (from the set of all latent relational search queries $\mathbb{Q}$) and an entity $D \in \Re(q)$ and returns a binary value as follows:

$$\mathbf{F}_e : \mathbb{Q} \times \Re(q) \rightarrow \{0, 1\} \tag{4.21}$$

$$\mathbf{F}_e(q, D) = \mathbf{F}_e(\{(A, B), (C, ?)\}, D) = \begin{cases} 1 & \text{if} \quad \text{RelSim}((A, B), (C, D)) \geq \sigma \\ 0 & \text{otherwise} \end{cases} \tag{4.22}$$

The entity filtering function $\mathbf{F}_e$ in Equation 4.21 is similar to the function in Equation 4.1. However, there are two important differences. First, the domain has been reduced from $\mathbb{Q} \times \mathbb{E}$ to $\mathbb{Q} \times \Re(q)$. This is a very important reduction because the candidate set has been reduced from the entire entity set to a small subset $\Re(q)$, which is defined in Equation 4.19. Second, the "relevant" concept has been clarified by the condition: $\text{RelSim}((A, B), (C, D)) \geq \sigma$.

In this implementation of the entity filtering function, we only consider a *small subset* of the original entity set $\mathbb{E}$ and therefore, we can achieve high speed in entity filtering.

4.6 Measuring the Relational Similarity between Two Entity Pairs

The index model in Figure 4.2 suggests a method for measuring the relational similarity between two entity pairs based on the number of identical lexical pattern clusters in the relational paths of the two pairs. Specifically, given the two entity pairs (A, B) and (C, D), we can first enumerate all relational paths connecting A with B and connecting C with D. Suppose that the set of all lexical pattern clusters in the relational paths between A and B is $\text{rp}(A, B)$, as defined in Equation 4.18. If we ignore the weights of the edges connecting an entity and a lexical pattern cluster, then the relational similarity between (A, B) and (C, D) is the degree of overlap between $\text{rp}(A, B)$ and $\text{rp}(C, D)$. This can be measured by a metric such as the Jaccard index:

$$\text{RelSim}((A, B), (C, D)) = \frac{|\text{rp}(A, B) \cap \text{rp}(C, D)|}{|\text{rp}(A, B) \cup \text{rp}(C, D)|} \tag{4.23}$$

However, in practice we can not ignore the weights of the edges between an entity and a lexical pattern cluster because they represent the strength of the association between the entity and the lexical pattern. To incorporate the weights of the edges into the similarity measure, we can use the algorithm proposed by Bollegala et al. in [30, 20]. These algorithms are optimized to precisely calculate the relational similarity between two entity pairs. However, these algorithms require heavy pre-processing steps as they must learn and store a large matrix. In addition, they require a vector-matrix multiplication to compute the relational similarity for each candidate pair, which might result in long query processing time. Therefore, in this work, we propose another algorithm to calculate the relational similarity based on the lexical pattern clusters obtained in the pattern clustering process. The algorithm is similar to those proposed by Bollegala et al. [30, 20] in the sense that it exploits the pattern clusters to alleviate the data sparseness problem regarding the exactly matched lexical patterns. Furthermore, it allows to compute the relational similarity between two entity pairs in high speed and it does not require large pre-processing time. The algorithm is shown in Algorithm 4.

In Algorithm 4, to calculate the relational similarity between two entity pairs s and c, we use their lexical pattern frequency (or PMI) vectors $\mathbf{\Psi}(s)$ and $\mathbf{\Psi}(c)$, as defined in Equation 4.11. We define the relational similarity between s and c using a modified version of cosine similarity of their pattern frequency vectors. The modified inner product of $\mathbf{\Psi}(s)$ and $\mathbf{\Psi}(c)$ is defined as follows: for a pattern $p \in \mathbf{P}(s) \cap \mathbf{P}(c)$ (the definition of $\mathbf{P}$ is shown in Equation 4.8), we add $\mathrm{f}(s,p) \cdot \mathrm{f}(c,p)$ (or $\mathrm{pmi}(s,p) \cdot \mathrm{pmi}(c,p)$ in case we are using PMI) to the inner product as normal (line 9). To identify the lexical patterns that are shared between the source pair and the target pair for supporting sentence retrieval, we report the common lexical patterns we found in the relational similarity calculation process by adding two pairs into the semantically similar co-occurrence set cL (in line 11, 12). Note that, we must record the lexical pattern together with the entity pair because supporting sentences can only be retrieved if we know which lexical pattern and which entity pair are in question. For a pattern $p \in \mathbf{P}(c)$ but $p \notin \mathbf{P}(s)$, we look for pattern $q \in \mathbf{P}(s) \backslash \mathbf{P}(c)$ that is in the same pattern cluster with p. If there are many patterns that satisfy this condition, we choose the one with largest frequency (line 17-22). Because the pattern q is in the same lexical pattern cluster with p, there is a high probability that q is semantically similar with p. Consequently, we add $\mathrm{f}(s,q) \cdot \mathrm{f}(c,p)$ to the inner product value (line 24). We also mark the chosen q to prevent it from participating to the next pattern comparison steps (line 25). Moreover, in this case, the lexical patterns are not shared between two pairs. For each pair, we must record the lexical pattern to retrieve supporting sentences separately (line 26, 27). Because p and q are semantically similar, we can retrieve semantically similar supporting sentences if we build the semantically similar co-occurrence set cL as shown in line 26, 27.

The proposed relational similarity measuring algorithm (Algorithm 4) requires only pattern clustering and does not require any vector-matrix multiplication. Therefore, it is fast enough to allow we investigate a large number of candidate answers. Moreover, it considers the strength of the association between an entity pair and each lexical pattern with which the entity pair co-occurs. This gives the same effect with the algorithm proposed by Bollegala et al. [30, 20], in the sense that it considers the association between an entity pair and a lexical pattern cluster.

The upper bound for the time complexity of Algorithm 4 can be estimated as follows. The body of the for-loop at line 7 is executed n_p times (n_p is the average number of lexical patterns that co-occur with an entity pair, as was used in Equation 4.20). The time complexity of the body of this for loop is in turn determined by the time complexity of the for loop at line 17, which is also n_p. Therefore, the overall time complexity of the

Algorithm 4 RelSim(s, c) : Calculate the relational similarity between two entity pairs

Input: two entity pairs s and c
Output: the relational similarity between s and c
side effect: the set of semantically similar co-occurrences cL is filled

```
/* Initialize the inner product to 0 */
ρ ← 0
/* Initialize the set of used patterns */
℘ ← {}
/* Clear the semantically similar co-occurrence set, which is a global variable */
cL.clear()
for pattern p ∈ P(c) do
  if p ∈ P(s) then
    ρ ← ρ + f(s,p)f(c,p)
    ℘ ← ℘ ∪ {p}
    cL.append(<s,p>) /* for supporting sentence retrieval */
    cL.append(<c,p>)
  else
    Ω ← the cluster that contains p
    max ← −1
    q ← null
    for pattern p_j ∈ (P(s)\P(c))\℘ do
      if (p_j ∈ Ω) ∧ (f(s,p_j) > max) then
        max ← f(s,p_j)
        q ← p_j
      end if
    end for
    if max > 0 then
      ρ ← ρ + f(s,q)f(c,p)
      ℘ ← ℘ ∪ {q}
      cL.append(<s,q>)
      cL.append(<c,p>)
    end if
  end if
end for
return ρ/(|Ψ(s)| · |Ψ(c)|)
```

algorithm is:

$$T(\text{RelSim}) = O(n_p^2) \tag{4.24}$$

As we have analyzed while evaluating the time complexity of the potential candidate retrieval process (in Section 4.5), n_p is often independent from the corpus size, provided that the corpus is large enough to cover almost all semantic relations between entity pairs. Therefore, as the corpus size goes above a certain level, the time complexity T(RelSim) will be a constant (or will increase very slow when we increase the corpus size).

The proposed relational similarity measuring algorithm (Algorithm 4) is similar to the algorithm in previous research [30, 20] in the sense that it exploits lexical pattern clusters information while calculating the similarity. However, in the algorithm proposed by Bollegala et al. [30, 20], they must accumulate the occurrences of an entity pair in a lexical

pattern cluster to create the final feature vector for an entity pair, as follows. The i^{th} element of the feature vector for the entity pair (A, B) is given by

$$\sum_{p \in c_i} \mathrm{f}(A, B, p) \tag{4.25}$$

in which c_i is the i^{th} pattern cluster and p is a lexical pattern in the cluster; $\mathrm{f}(A, B, p)$ is the number of co-occurrences between the pair (A, B) and the lexical pattern $p \in c_i$. Consequently, the algorithm by Bollegala et al. requires re-calculating the feature vector values for each entity pair (A, B) whenever the lexical pattern clusters change. On the other hand, in Algorithm 4, we do not require the feature vector values to be calculated, but we directly use the number of co-occurrences between an entity pair (e.g., $s = (A, B)$) and a lexical pattern p to derive the relational similarity (i.e., we use $\mathrm{f}(s, p)$, instead of $\sum_{p \in c_i} \mathrm{f}(s, p)$). We use the information regarding lexical pattern clusters when we do not find a match between two lexical patterns of the two input pairs (line 14 and 18 of Algorithm 4). This calculation method reduces the time complexity in the process of relational similarity measuring. Moreover, as the algorithm does not require re-calculating the feature vector values even when the lexical pattern clusters have changed, it is appropriate to use with incremental pattern clustering algorithms.

4.7 Entity Ranking Function

Using the entity filtering function, we achieve a candidate entity set for ranking $\mathbb{S}(q)$ (in Equation 4.3). The next step in processing a latent relational search query is ranking these candidate entities to achieve a ranked result list. We use the relational similarity measure proposed in Section 4.6 to rank the candidate set. Specifically, for a query $q = \{(A, B), (C, ?)\}$, we perform the candidate filtering process for the original query and the reversed query $q' = \{(B, A), (?, C)\}$. We define the relevance score $\mathrm{Z}(q, D)$ (or $\mathrm{Rel}((A, B), (C, D))$) of a candidate entity D for the query $q = \{(A, B), (C, D)\}$ as follow:

$$\mathrm{Z}(q, D) = \mathrm{Rel}((A, B), (C, D)) = \mathrm{RelSim}((A, B), (C, D)) + \frac{1}{2}\mathrm{RelSim}((B, A), (D, C)) \tag{4.26}$$

(note that the relevance score might be written as $\mathrm{Rel}((A, B), (C, D))$ when we use the entities A, B, C to express the query q). We utilize the relational similarity $\mathrm{RelSim}((B, A), (D, C))$ in the definition of the relevance score $\mathrm{Rel}((A, B), (C, D))$ because we can assume that the relational similarity between (B, A) and (D, C) is not too different from the relational similarity between (A, B) and (C, D), as described in Section 2.2. We set the weight of the relational similarity of the reversed entity pair $(\mathrm{RelSim}((B, A), (D, C)))$ to $1/2$ because we prefer that the candidate appears in the original query rather than the reversed query. Using reversed queries when processing latent relational search queries improves the performance of latent relational search, as shown in [9].

We rank the candidate entities in the candidate set $\mathbb{S}(q)$ by descending order of the score $\mathrm{Z}(q, D)$. That is, the final result list is achieved by sorting the candidate set $\mathbb{S}(q)$ using the score $\mathrm{Z}(q, D)$ as the sort key. By this method, the rank of each entity is indirectly calculated. Therefore, it is not required to calculate the value of the entity ranking function for each entity in the set $\mathbb{S}(q)$. However, we still want an entity ranking function to reveal the semantic of the ranking process. When implementing this model of latent relational search, one is not required to calculate the value of the entity ranking function. To clarify the definition of the entity ranking function, we use some additional definitions, as follows.

Definition 4.7.1. *We denote the set of all entities in the candidate set* $\mathbb{S}(q)$ *that have the relevance scores higher than or equal to the score of the entity* D *as* $\text{HigherThan}(q, D)$*:*

$$\text{HigherThan}(q, D) = \{D_i \in \mathbb{S}(q) \quad | \quad \text{Z}(q, D_i) \geq \text{Z}(q, D) \wedge D_i \neq D\} \tag{4.27}$$

Theorem 4.7.1. *If there exists at least a candidate entity then:*

$$0 \leq |\text{HigherThan}(q, D)| \leq |\mathbb{S}(q)| - 1$$

Proof. Because HigherThan is a set, its size is always greater than or equal to zero. On the other hand, $|\text{HigherThan}(q, D)|$ can not exceed $|\mathbb{S}(q)|-1$ because the set $\text{HigherThan}(q, D)$ does not include the entity D itself, while $D \in \mathbb{S}(q)$. □

Theorem 4.7.2. *For* $D, D' \in \mathbb{S}(q)$*, if* $\text{Z}(q, D) > \text{Z}(q, D')$ *then:*

$$\text{HigherThan}(q, D) \subset \text{HigherThan}(q, D')$$

Proof. If an entity $X \in \text{HigherThan}(q, D)$ then

$$\text{Z}(q, X) \geq \text{Z}(q, D) > \text{Z}(q, D')$$

This means, $X \in \text{HigherThan}(q, D')$. Therefore,

$$\text{HigherThan}(q, D) \subseteq \text{HigherThan}(q, D')$$

However, because $\text{Z}(q, D) > \text{Z}(q, D')$, the entity D is in the set $\text{HigherThan}(q, D')$, while D is not included in the set $\text{HigherThan}(q, D)$. Therefore,

$$\text{HigherThan}(q, D) \neq \text{HigherThan}(q, D')$$

Consequently,

$$\text{HigherThan}(q, D) \subset \text{HigherThan}(q, D')$$

□

We can then define the entity ranking function as follows.

Definition 4.7.2. *The rank* $\mathbf{R}_e(q, D)$ *of an entity* D *in the result set of the query* q *is:*

$$\mathbf{R}_e(q, D) = 1 + |\text{HigherThan}(q, D)| \tag{4.28}$$

We will show that the entity ranking function $\mathbf{R}_e(q, D)$ in Equation 4.28 satisfies all conditions in Equations 4.4 – 4.6.

Theorem 4.7.3. *The entity ranking function* $\mathbf{R}_e(q, D)$ *defined in Equation 4.28 satisfies the following conditions*

$$\mathbf{R}_e \in \mathbb{N}^{\mathbb{Q} \times \mathbb{E}} \tag{4.29}$$

$$1 \leq \mathbf{R}_e(q, D) \leq |\mathbb{S}(q)|, \quad \forall q, D \tag{4.30}$$

$$\begin{aligned} &\forall D_i, D_j \in \mathbb{S}(q) : \\ &\text{Rel}((A, B), (C, D_i)) > \text{Rel}((A, B), (C, D_j)) \Rightarrow \mathbf{R}_e(q, D_i) < \mathbf{R}_e(q, D_j) \end{aligned} \tag{4.31}$$

Proof. The first condition (the condition in Equation 4.29) states that the entity ranking function is a mapping from the set $\mathbb{Q} \times \mathbb{E}$ to the set of natural number $\mathbb{N}$. Because $\mathbf{R}_e$ takes a query q and an entity D as its parameters and outputs a positive integer (the rank of the entity D), this condition is satisfied.

The second condition states that, the value of $\mathbf{R}_e$ is always in the range from 1 to the size of the candidate set $\mathbb{S}(q)$. This is a direct result of Theorem 4.7.1: because

$$0 \leq |\text{HigherThan}(q, D)| \leq |\mathbb{S}(q)| - 1 \quad \text{(Theorem 4.7.1)}$$

we have:

$$1 \leq \mathbf{R}_e(q, D) = 1 + |\text{HigherThan}(q, D)| \leq |\mathbb{S}(q)|$$

The third condition states that if the relevance score of D_i is higher than that of D_j, then the rank of D_i must be smaller than the rank of D_j. Because $\text{Rel}((A, B), (C, D)) = \text{Z}(q, D)$, we must prove that:

$$\text{Z}(q, D_i) > \text{Z}(q, D_j) \Rightarrow \mathbf{R}_e(q, D_i) < \mathbf{R}_e(q, D_j)$$

This is equivalent to

$$\text{Z}(q, D_i) > \text{Z}(q, D_j) \Rightarrow |\text{HigherThan}(q, D_i)| < |\text{HigherThan}(q, D_j)|$$

This is a direct result of Theorem 4.7.2. □

Finally, the ranked result list of the query q can be constructed using the entity filtering function $\mathbf{F}_e(q, D)$ and the entity ranking function $\mathbf{R}_e(q, D)$ as shown in Algorithm 5. In

Algorithm 5 Building the ranked result list for a query q

Input: A query q
Output: A ranked list of entities, which are answers for q

```
/* Calculate the set of candidates for ranking */
𝕊(q) ← {D ∈ 𝔼 | F_e(q, D) = 1}
/* Array for storing the answers in descending order of score */
arr ← array[1..|𝕊(q)|] of entities
/* Initialize all elements of the array to null */
for i ← 1 to |𝕊(q)| do
    arr[i] ← null
end for
for each entity D ∈ 𝕊(q) do
    /* Calculate the value of the entity ranking function for the entity D */
    r_D ← R_e(q, D)
    /* Put D into the correct position in the array */
    for k ← r_D to |𝕊(q)| do
        if arr[k] = null then
            break
        end if
    end for
    arr[k] ← D
end for
L ← arr.toList()
return L
```

Algorithm 5, we first construct an array, which stores the answer entities in correct order. This can be done by calculating the rank of each entity (line 11 of Algorithm 5) and then directly put the entity into the array, according the the rank. However, because there might be several entities that have the same rank (because they have the same score $\mathrm{Z}(q, D)$), we must choose consecutive positions in the array to put them (line 13–18). The final array is guaranteed to be filled with all entities in the set $\mathbb{S}(q)$ and each position is filled exactly once. This is because the definition of the ranking function (Equation 4.28) guarantees that if an entity D_i has larger score than D_j then the rank of D_i must be smaller than D_j. Moreover, if two entities have the same score, then they will have the same rank. Entities with the same rank can be inserted in the final ranked list of answers in an arbitrary order. In Algorithm 5, the indices of entities with the same rank is determined according to the order of the scan operation in the for loop at line 9. We can prove that the ranking procedure in Algorithm 5 satisfies the conditions in Equations 2.3 – 2.4.

Theorem 4.7.4. *The ranking procedure in Algorithm 5 returns a ranked list of answer entities L (for the query $q = \{(A, B), (C, ?)\}$) that satisfies the following conditions*

$$\mathrm{RelSim}((A, B), (C, D_i)) > 0, \quad \forall D_i \in L$$

$$\mathrm{Rel}((A, B), (C, D_i)) \geq \mathrm{Rel}((A, B), (C, D_j)), \quad \forall 1 \leq i \leq j \leq |L|$$

Proof. The first condition states that, all entities D in the list L have the similarity $\mathrm{RelSim}((A, B), (C, D))$ higher than zero. This is satisfied because an entity $D \in L$ always has $\mathbf{F}_e(q, D) \geq \sigma$ (Equation 4.22) and σ is set to a value higher than zero.

The second condition states that, the list is sorted in descending order of the relevance score $\mathrm{Rel}((A, B), (C, D))$. To prove this, consider two indices i and j in the list L such that $i \leq j$. From Algorithm 5, we have:

$$\mathbf{R}_e(q, D_i) \leq \mathbf{R}_e(q, D_j)$$

This implies that

$$|\mathrm{HigherThan}(q, D_i)| \leq |\mathrm{HigherThan}(q, D_j)| \tag{4.32}$$

From the definition of HigherThan, this implies that $D_i \in \mathrm{HigherThan}(q, D_j)$. To see this, suppose that $D_i \notin \mathrm{HigherThan}(q, D_j)$. Therefore, $\mathrm{Z}(q, D_i) < \mathrm{Z}(q, D_j)$ and hence:

$$\mathrm{HigherThan}(q, D_i) \supseteq \mathrm{HigherThan}(q, D_j)$$

$$D_j \in \mathrm{HigherThan}(q, D_i)$$

However, because $D_j \notin \mathrm{HigherThan}(q, D_j)$, we have:

$$\mathrm{HigherThan}(q, D_i) \supset \mathrm{HigherThan}(q, D_j)$$

This conflicts with Equation 4.32. Therefore, $D_i \in \mathrm{HigherThan}(q, D_j)$. Consequently,

$$\mathrm{Z}(q, D_i) \geq \mathrm{Z}(q, D_j)$$

This implies that

$$\mathrm{Rel}((A, B), (C, D_i)) \geq \mathrm{Rel}((A, B), (C, D_j))$$

□

Although the procedure in Algorithm 5 is a purely theoretical procedure (when we have the value of the entity ranking function $\mathbf{R}_e$), it is important for understanding how the proposed ranking method satisfies the conditions in Equations 2.3 – 2.4. In actual implementations, it is not required to implement Algorithm 5, as well as explicitly calculate the value $\mathbf{R}_e$ for each candidate entity. One can simply calculate the relevance score $\mathrm{Z}(q, D)$ for each entity D and then sort the result list in descending order of the relevance score to achieve the same effect.

4.8 Supporting Sentence Retrieval

The final step in processing a latent relational search query is to retrieve a set of supporting sentences for each candidate entity pair. To retrieve supporting sentences for a candidate pair, we rely on the semantically similar co-occurrence set cL in Algorithm 4. The set cL is determined when two entity pairs for the input of the algorithm are given. Suppose that the source pair is $s = (A, B)$ and the candidate pair is $c = (C, D)$. In this case, cL is a function of s and c, and we denote it as $cL(s, c)$. Each element of the set cL is a pair $<w, p>$ in which w is an entity pair and p is a lexical pattern. The entity pair w must be the input source entity pair (s) or the candidate target pair (c). From Algorithm 4, the lexical pattern p must co-occur with the entity pair w. To see this, we can check the condition when the `append` method of cL is invoked. At line 12 and 27 of Algorithm 4, when the statement cL.append($<c, p>$) is issued, we are in the for loop starting at line 7. Therefore, p is a loop variable that is drawn from the set $\mathbf{P}(c)$. Consequently, $p \in \mathbf{P}(c)$ (the lexical pattern set as defined in Equation 4.8), and therefore p co-occurs with c. At line 11 of Algorithm 4, the statement cL.append($<s, p>$) is invoked when we are in the **if** branch with the condition $p \in \mathbf{P}(s)$. This implies that p co-occurs with s in this case. Finally, at line 26, the when the statement cL.append($<s, q>$) is invoked, q must be assigned at line 20. In this case, q must be a member of the set $(\mathbf{P}(s) \backslash \mathbf{P}(c)) \backslash_{\wp}$. This implies that $q \in \mathbf{P}(s)$ and therefore, q co-occurs with $\mathbf{P}(s)$. There are not any other places in Algorithm 4 that modify the set cL. Therefore, for every pair $<w, p>$ in the set cL, the condition "w co-occurs with p" holds. Because w co-occurs with p, it must co-occur with p in some sentences extracted from the input text corpus. Suppose that the set of sentences in which w appears is $\mathrm{Sents}_{\mathrm{entity}}(w)$ and the set of sentences in which p appears is $\mathrm{Sents}_{\mathrm{pattern}}(p)$. We must record these sets of sentences in the index, when we store the information concerning each entity pair and each lexical pattern. We can then retrieve the set of sentences in which the entity pair w and the lexical pattern p co-occur by computing the intersection between the above two sets:

$$\mathrm{Sents}_{\mathrm{wp}}(<w, p>) = \mathrm{Sents}_{\mathrm{entity}}(w) \cap \mathrm{Sents}_{\mathrm{pattern}}(p)$$

However, the set $\mathrm{Sents}_{\mathrm{wp}}$ often contains similar sentences because each sentence in this set always refers to the semantic relation stated by p. Therefore, we take only the first sentence of the set $\mathrm{Sents}_{\mathrm{wp}}$ to the final supporting sentence set, and we denote the set that contains only this sentence as $\mathrm{Sents1}_{\mathrm{wp}}$ (the size of this set is one). Note that, the average time complexity for calculating $\mathrm{Sents1}_{\mathrm{wp}}$ is the same as the time complexity for calculating $\mathrm{Sents}_{\mathrm{wp}}$. To retrieve all supporting sentences for the candidate pair c, we can take the union of $\mathrm{Sents1}_{\mathrm{wp}}(<w, p>)$, for all pair $<w, p>$ in cL:

$$\mathrm{Sents}(s, c) = \bigcup_{<w,p> \in cL(s,c)} \mathrm{Sents1}_{\mathrm{wp}}(<w, p>)$$

In practice, we can limit the number of elements of the set $\mathrm{Sents}(s, c)$ to a constant (e.g., 50) to retrieve a subset of supporting sentences. Algorithm 6 shows the procedure to

retrieve a subset of supporting sentences for a candidate entity pair based on the above formulae. Because we limit the number of supporting sentences in the returned set to

Algorithm 6 Retrieving a subset of supporting sentences for a target entity pair (in monolingual case)

Input: Source entity pair $s = (A, B)$ and target pair $c = (C, D)$; semantically similar co-occurrence set cL

Output: A set of supporting sentences for the pair c

```
1:  /* Initialize the set of supporting sentences */
2:  Sents ← {}
3:  /* Retrieve the set of sentences in which s appears */
4:  Sents_entity_s ← retrieve from the index sent. set for s
5:  /* Retrieve the set of sentences in which c appears */
6:  Sents_entity_c ← retrieve from the index sent. set for c
7:  for each <w, p> in cL do
8:     Sents_pattern ← retrieve from the index sent. set for p
9:     if w is c then
10:       Sents_wp ← Sents_entity_c ∩ Sents_pattern
11:    else
12:       Sents_wp ← Sents_entity_s ∩ Sents_pattern
13:    end if
14:    /* Only use the first sentence */
15:    firstSent ← first_of(Sents_wp)
16:    Sents ← Sents ∪ {firstSent}
17:    if |Sents| ≥ 50 then
18:       break
19:    end if
20: end for
21: return Sents
```

50, the time complexity of Algorithm 6 is a constant times of the complexity of the set intersection operation in line 10 or 12 (we assume that the set of sentences for an entity pair or a pattern can be retrieved from the index in $O(1)$ time). If the average number of sentences in which an entity appears is s_e and the average number of sentences in which a lexical pattern appears is s_p then time complexity of Algorithm 6 is:

$$T(Sents) = O(\max(s_e, s_p))$$

The procedure in Algorithm 6 does not assess the importance of lexical patterns. Therefore, it might retrieve sentences which do not strongly reflect the target semantic relations. To retrieve a set of supporting sentences that are strongly relevant to the target semantic relations, we can use some criteria such as the Point-wise Mutual Information (PMI) between a lexical pattern and the input source entity pair to assess the degree of association between the pattern and the entity pair. We can then take only the patterns with strong associativities to retrieve a strong relevant sentence list. Finally, we can rank the sentence list by this degree of association because a sentence is often determined by the lexical patterns which it includes. However, the sentence ranking problem is beyond the scope of this thesis. In this work, we only retrieve a set of supporting sentences, without ranking them. We discuss the supporting sentence ranking problem as a future research direction in Chapter 7.

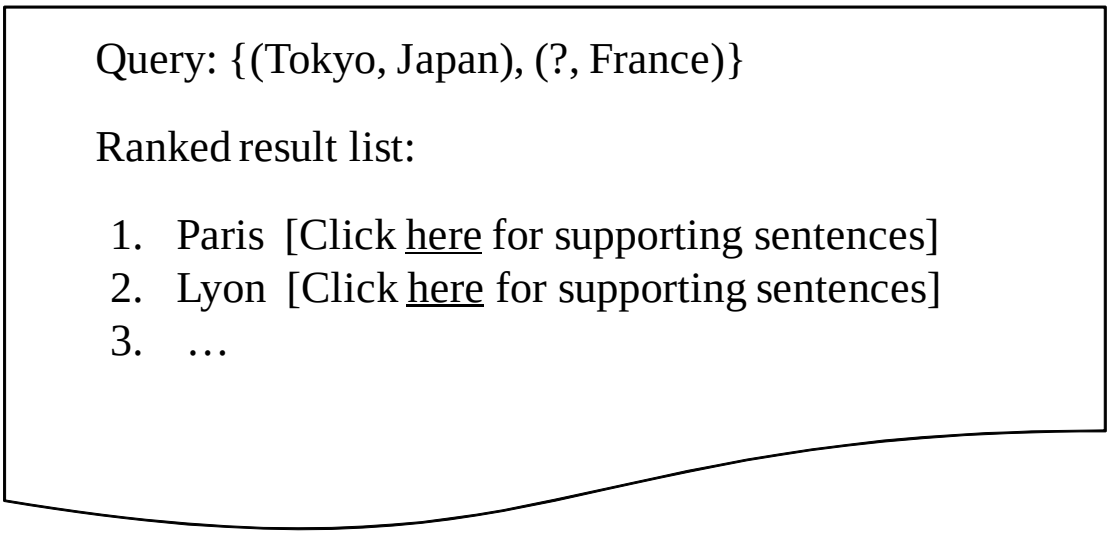


Fig. 4.3. An example user interface for the separation of the supporting sentence retrieval phase from the entity retrieval phase

When implementing a latent relational search engine, one can separate the supporting sentence retrieval problem from the candidate answer retrieval and ranking problem by providing a user interface which first displays a ranked list of answer entities together with hyperlinks (or buttons) to initiate the supporting sentence retrieval process, as shown in Figure 4.3. The separation helps to reduce the period from when the query is input to the time when the result list appears. Moreover, with the AJAX (Asynchronous JavaScript and XML) programming technologies, the list of supporting sentences can be shown in the same Web page with the entity list page. Therefore, a user can not conceive the differences between an implementation that uses the separation with one that does not.

From this reason, when evaluating the response time of a latent relational search engine, we only evaluate the period of time from when the query is input to when the answer entity list is returned, *without* supporting sentences. Moreover, when discussing the time complexity of the query processing algorithm, we only discuss the time complexity for retrieving and ranking the candidate answer entity list.

4.9 Implementation

In this section, we present a prototype implementation of the proposed model for monolingual latent relational search. We use this implementation for preliminary evaluation of the proposed model in Section 4.10. Because this implementation is mainly for creating a proof of concept for the proposed retrieval model, we exploit several techniques to reduce the implementation time, such as combination of programming languages (using the most appropriate programming language to implement each module and use XMLRPC for inter-module communication) or storing the index in random access memory (RAM). The components of the implemented system is shown in Figure 4.4.

In Figure 4.4, given the Web corpus (containing HTML pages), we first build an HTML to Plain Text Converter to extract text from each HTML page. The HTML to Plain Text Converter parses an HTML file and extracts text segments in each tag such as the <p>tag or the tag. It then concatenates these text segments to create final text contents

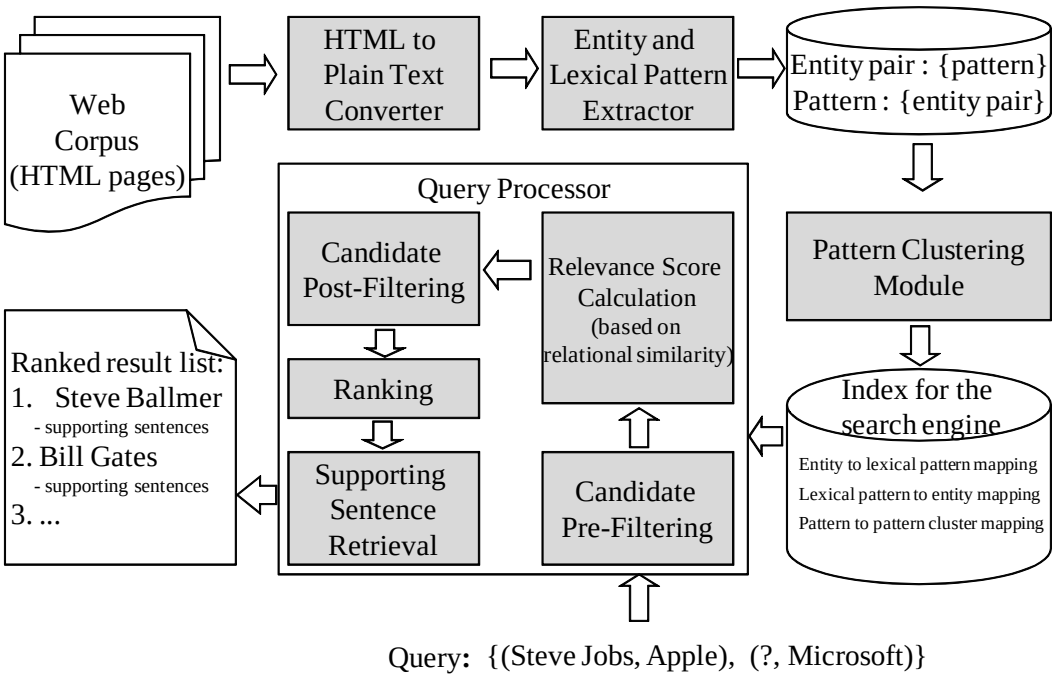


Fig. 4.4. Overview of the prototype monolingual latent relational search engine

of the file. The Converter is written in Ruby because we want to use the HPricot[*7] library for parsing an HTML file. The HPricot library, which can recover from several HTML markup errors, is robust for parsing HTML. From the plain text contents of each document, we use the Entity and Lexical Pattern Extractor to extract entity pairs and lexical patterns that might represent the semantic relations in each pair. The Entity and Lexical Pattern Extractor implements the algorithm in Section 4.2 for extracting entity pairs and lexical patterns. The Extractor is implemented in Python because we want to use the Python NLTK[*8] toolkit for stemming and generating lexical patterns. The Extractor invokes the HTML to Plain Text Converter by using remote procedure calls (XMLRPC). Moreover, the Extractor uses the Stanford Named Entity Recognizer (NER) and the MeCab POS Tagger to recognize named entities in each sentence. Because the Stanford NER is written in Java, while the Extractor is implemented in Python, we create an XMLRPC server wrapper for the Stanford NER and invoke it by remote procedure calls from Python. For Japanese sentences, we use the MeCab POS Tagger (which provides Python bindings) for tagging.

After extracting entity pairs and lexical patterns, we have an index that contains the hash from entity pairs to lexical patterns (that each pair co-occurs with), as well as from lexical patterns to entity pairs. At this step, we input these mappings into the Pattern Clustering Module. The Pattern Clustering Module implements the Monolingual Lexical Pattern Clustering (MLPC) algorithm (as shown in Algorithm 2). It is written in Python. After the clustering step, we obtain the final index that is the physical representation of the index model in Figure 4.2.

The last component of the system is the Query Processor (as shown in the square block

[*7] http://hpricot.com/
[*8] http://www.nltk.org/

in Figure 4.4). The Query Processor contains several sub-modules, which are all implemented in Python. The first sub-module is the Candidate Pre-Filtering module, which implements the procedure in Algorithm 3 to retrieve a potential candidate set. The second sub-module is for calculating relevance scores, which actually implements the function $\mathrm{Z}(q, D)$ as shown in Equation 4.26. This function requires the calculation of the relational similarity between the source pair and the candidate target pair. Next, the Candidate Post-Filtering module implements the entity filtering function $\mathbf{F}_e(q, D)$, as shown in Equation 4.22. Note that, at this step, we have calculated all relational similarities required by the entity filtering function $\mathbf{F}_e$. Therefore, the function can be executed in $O(1)$ time (only a comparison operation between two floating point numbers) for each candidate. The results of the filtering function and the relational similarity calculation process are then fed to the Ranking module, which simply performs a QuickSort operation to achieve the ranked result list. This operation effectively calculates $\mathbf{R}_e(q, D)$ for all candidate entities. Therefore, the average time complexity for this step is $O(|\mathbb{S}(q)|\log|\mathbb{S}(q)|)$ for *all* candidates (in which $\mathbb{S}(q)$ is defined in Equation 4.3). Finally, for each candidate entity pair, we retrieve supporting sentences by implementing the procedure in Algorithm 6. However, as explained in Section 4.8, we can separate the supporting sentence retrieval phase from the entity retrieval phase. Therefore, we do not account the time of the supporting sentence retrieval process into the query processing time. If the average time complexity of $\mathrm{Z}(q, D)$ is $T(Z)$ then the time complexity for processing a query is:

$$T(\text{QueryProcessing}) = O(T(\Re) + n_e n_p T(Z) + |\mathbb{S}|\log|\mathbb{S}|)$$

In which n_e is the average number of entities that co-occur with a lexical pattern and n_p is the average number of lexical patterns that co-occur with an entity pair. The first term is the time complexity of the potential candidate retrieval operation (as shown in Equation 4.20). The second term is the time to calculate the relational similarity for all candidate pairs (the upper bound for the number of candidate pairs is $n_e n_p$, the size of the set $\Re$). The third term is the time of the QuickSort operation in the ranking phase. The time $T(Z)$ is a constant times of $T(\text{RelSim})$ (Algorithm 4). $T(\text{RelSim})$ is $O(n_p^2)$, as shown in Equation 4.24. Therefore, the time complexity $T(\text{QueryProcessing})$ becomes:

$$T(\text{QueryProcessing}) = O(n_e n_p^3 + |\mathbb{S}|\log|\mathbb{S}|)$$

The upper bound for the size of the candidate set $\mathbb{S}$ is the size of the set $\Re$. The upper bound for the size of the set $\Re$ is $O(n_e n_p)$. Consequently, the time complexity for processing a query is:

$$T(\text{QueryProcessing}) = O(n_e n_p^3 + n_e n_p \log n_e + n_e n_p \log n_p) = O(n_e n_p (n_p^2 + \log n_e))$$

Because the time complexity is a cubic function of the average number of lexical patterns that co-occur with an entity pair (n_p), we can drastically reduce the time complexity by reducing n_p. To achieve this, we can drop rare patterns (e.g., patterns that appear only once) or improve the precision of the lexical pattern extraction process (to extract only patterns that strongly represent the semantic relations). Normally, n_p and n_e depend on the number of semantic relations that are held in the source entity pair and the candidate entity pair, and they are *independent* from the corpus size. Therefore, the query processing time is independent from the corpus size, when the corpus size is larger than a certain level (so that the corpus is large enough to cover almost all semantic relations between entities). Consequently, with a large corpus, the query processing time depends on the number of semantic relations that are held in the input source entity pair and the target pairs, irrespective to the size of the corpus.

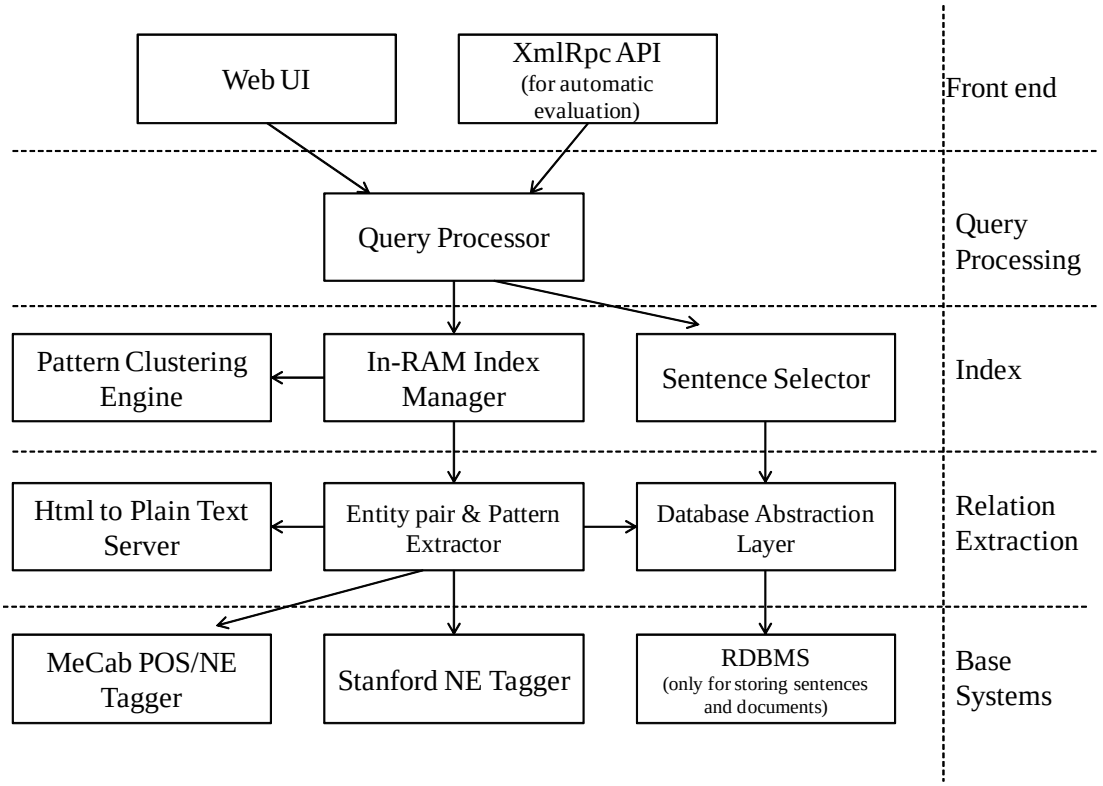


Fig. 4.5. The architecture of the prototype implementation of a latent relational search engine

Figure 4.5 presents the architecture of our prototype of the monolingual latent relational search engine. The prototype is implemented by a layered architecture with the In-RAM Index Manager and the Query Processor at the center. The In-RAM Index Manager triggers the index building process: it invokes the Entity and Lexical Pattern Extractor to analyze text corpora. The Query Processor implements all modules in the "Query Processor" block in Figure 4.4. At the lowest layer, the "Base Systems" layer, we use the Stanford Named Entity Recognizer*[9], which are trained to recognize three types of entities (organization, location and person) for recognizing named entities in sentences. We use the MeCab POS Tagger*[10] for Japanese word segmentation and POS tagging. The MeCab POS Tagger also recognizes named entities of several types. We extract all named entities that MeCab identifies as organization, location or person. Because the number of extracted sentences is large, we store these sentences using a MySQL database server.

Other indices, such as the index from entity pairs to lexical patterns or the inverted index from lexical patterns to entity pairs, are kept in RAM. Therefore, the prototype is only for experiment purpose, as it requires re-processing the entire corpus after a system restart. However, this prototype is an important system because we can use it to carry out several experiments for tuning the parameters in the proposed method. After we have learned the appropriate values for the parameters, we directly use them in a large scale implementation, as discussed in Chapter 6.

*[9] http://nlp.stanford.edu/software/CRF-NER.shtml
*[10] http://mecab.sourceforge.net/

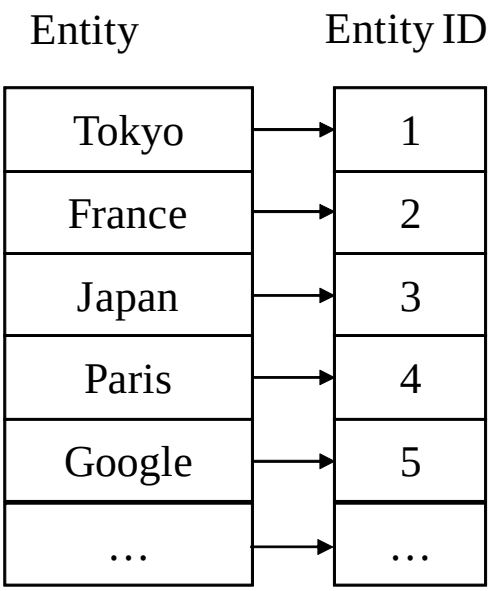


Fig. 4.6. The mapping from Entities to Entity IDs

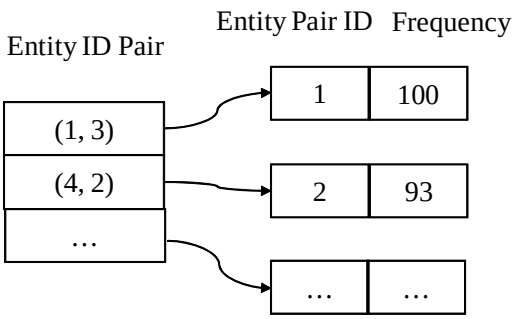


Fig. 4.7. The mapping from Entity pairs to Entity pair IDs

Figures 4.6 – 4.10 show some tables (mappings) that are included in the indices for the search engine. We first assign a unique ID for each entity, as shown in Figure 4.6. Using these IDs reduces the amount of memory to store the index. We then use entity ID pairs for storing information related to entity pairs and assign a unique ID for each entity pair, as shown in Figure 4.7. We record the frequency of each entity pair in the same table.

Similarly, we map each lexical pattern to a unique Pattern ID, as shown in Figure 4.8. We also record the frequency of each lexical pattern in this table.

The index from Entity Pairs to Lexical Patterns is shown in Figure 4.9. At this step, because we have mapped all entity pairs and lexical patterns into IDs, we can use only these IDs to store the index. We record the number of co-occurrences of each entity pair with each lexical pattern for easily calculating the relational similarity.

Finally, to support the inverted index lookup operations (such as in retrieving the potential candidate set $\Re$), we store the inverted index from Lexical Patterns to Entity Pairs, as shown in Figure 4.10. The inverted index has the same structure with the index (from Entity Pairs to Lexical Patterns). Because we use IDs for storing co-occurrence information, we can drastically reduce the amount of memory required by the index.

4.10 Evaluation

In this section, we evaluate the proposed model with monolingual latent relational search query sets in both English and Japanese. We first describe the query sets (entity and relation types) that we use for evaluation. We then describe the method for gathering Web corpora that contain documents related to the semantic relations in the query sets. Next, we present the parameter tuning process, in which we determine an appropriate value for the parameter θ_1 in the lexical pattern clustering algorithm (Algorithm 2). In Section 4.10.3, we evaluate the effect of using different values (frequency and PMI) as elements of feature vectors. To verify the effectiveness of the proposed lexical pattern extraction algorithm, we compare the performance of the proposed method with the method based on an existing lexical pattern extraction algorithm in Section 4.10.4. We show the evalua-

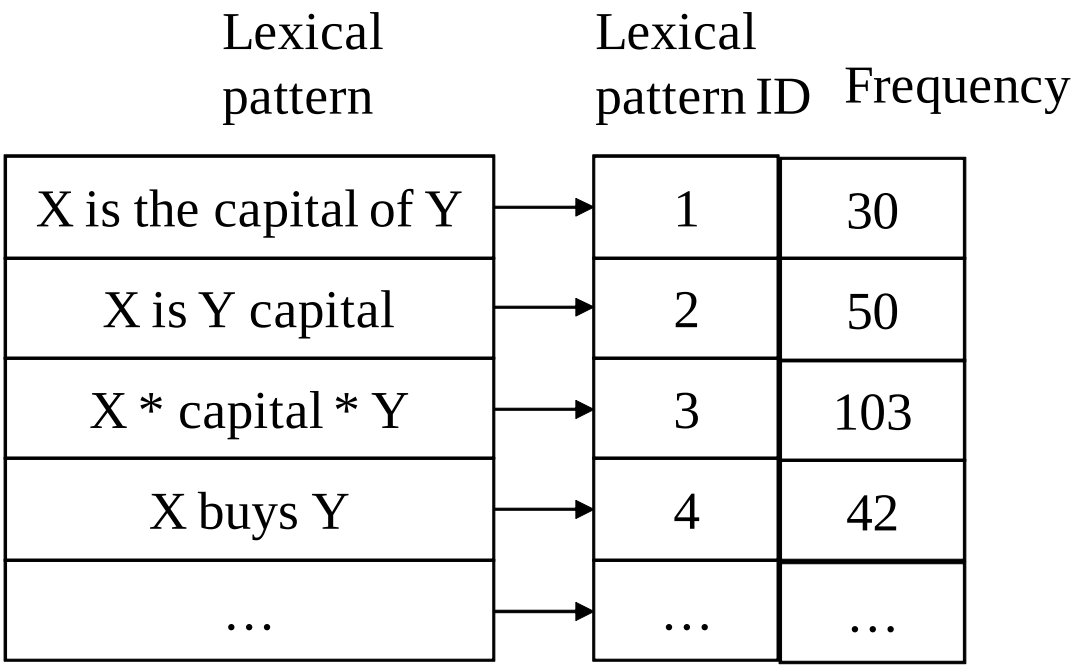


Fig. 4.8. The mapping from Lexical patterns to Pattern IDs

tion result on each query set and compare the performance with an existing monolingual latent relational search engine in Section 4.10.5 and Section 4.10.6, respectively. Finally, we evaluate the query processing time of the proposed search engine in Section 4.10.7.

4.10.1 Dataset

We use the relation types in Table 4.3 for evaluation of the proposed model on monolingual latent relational search. These relation types are frequently used for evaluating relation extraction systems [7, 109], relational similarity measuring algorithms [20, 6] and latent relational search systems [8].

To create latent relational search query sets concerning the semantic relations in Table 4.3, for each relation type, we first collect a set of seed entity pairs in English and another set of seed entity pairs in Japanese. We then enumerate all entity pairs of the same relation types in the same language. We combine two entity pairs into a tuple and then remove one of the four entities in the tuple to create a query. For example, from the tuple {(Ganymede, Jupiter), (Oberon, Uranus)}, we remove Uranus to form the English monolingual latent relational search query {(Ganymede, Jupiter), (Oberon, ?)}. For a majority number of relation types, we remove the entity so that each query has only one correct answer. For example, from the tuple {(Ganymede, Jupiter), (Oberon, Uranus)}, we do not remove the entity *Oberon* to form the query {(Ganymede, Jupiter), (?, Uranus)} because this query has multiple correct answers (the satellites of Uranus). Instead, we remove the entity *Uranus* to form the query {(Ganymede, Jupiter), (Oberon, ?)}, which has only one correct answer (Uranus). To evaluate the ability of retrieving and ranking answers of the proposed system on a query set where each query has multiple correct answers, we also create the query set *Acquirer-Acquiree*. Each query in the query set *Acquirer-Acquiree* has multiple correct answers. For example, from the tuple {(Google, YouTube), (Microsoft, Powerset)}, we remove *Powerset* to form the query {(Google, YouTube), (Microsoft, ?)}.

Table. 4.3. Relation types for evaluation (italic entities are actually in Japanese)

Relation type	Exists between	Example
BIRTHPLACE	A person and his place of birth	(Franz Kafka, Prague) (Albert Einstein, Ulm) (*Hamasaki Ayumi*, *Fukuoka*) (*Nakata Hidetoshi*, *Yamanashi*) ...
HEADQUARTERS	A company and its headquarters	(Google, Mountain View) (Apple, Cupertino) (*Toyota*, *Aichi*) (*Nitendo*, *Kyoto*) ...
CEO	A CEO and his company	(Larry Page, Google) (Michael Dell, Dell) (*Toyoda Akio*, *Toyota*) (*Wada Isamu*, *Sekisui Hausu*) ...
ACQUISITION	Two companies	(Google, YouTube) (Microsoft, Powerset) (*Panasonikku*, *Sanyo*) (*Rakuten*, *Infoseek*) ...
PRESIDENT	A person and the country whose president is this person	(Barack Obama, U.S) (Dmitry Medvedev, Russia) (*Sarukozi, Furansu*) (*I MyonBagu*, *Kankoku*) ...
PRIMEMINISTER	A person and the country whose PM is this person	(David Cameron, U.K) (Angela Merkel, Germany) (*Kan Naoto*, *Nihon*) (*On Kahou*, *Chuugoku*) ...
CAPITAL	A city and the country whose the capital is this city	(Paris, France) (Hanoi, Vietnam) (*Tokyo*, *Nihon*) (*Peiking*, *Chuugoku*) ...
SATELLITE	A moon and the planet which the moon orbits	(Ganymede, Jupiter) (Oberon, Uranus) (*Ganimede*, *Mokusei*) (*Oberon, Tennosei*) ...

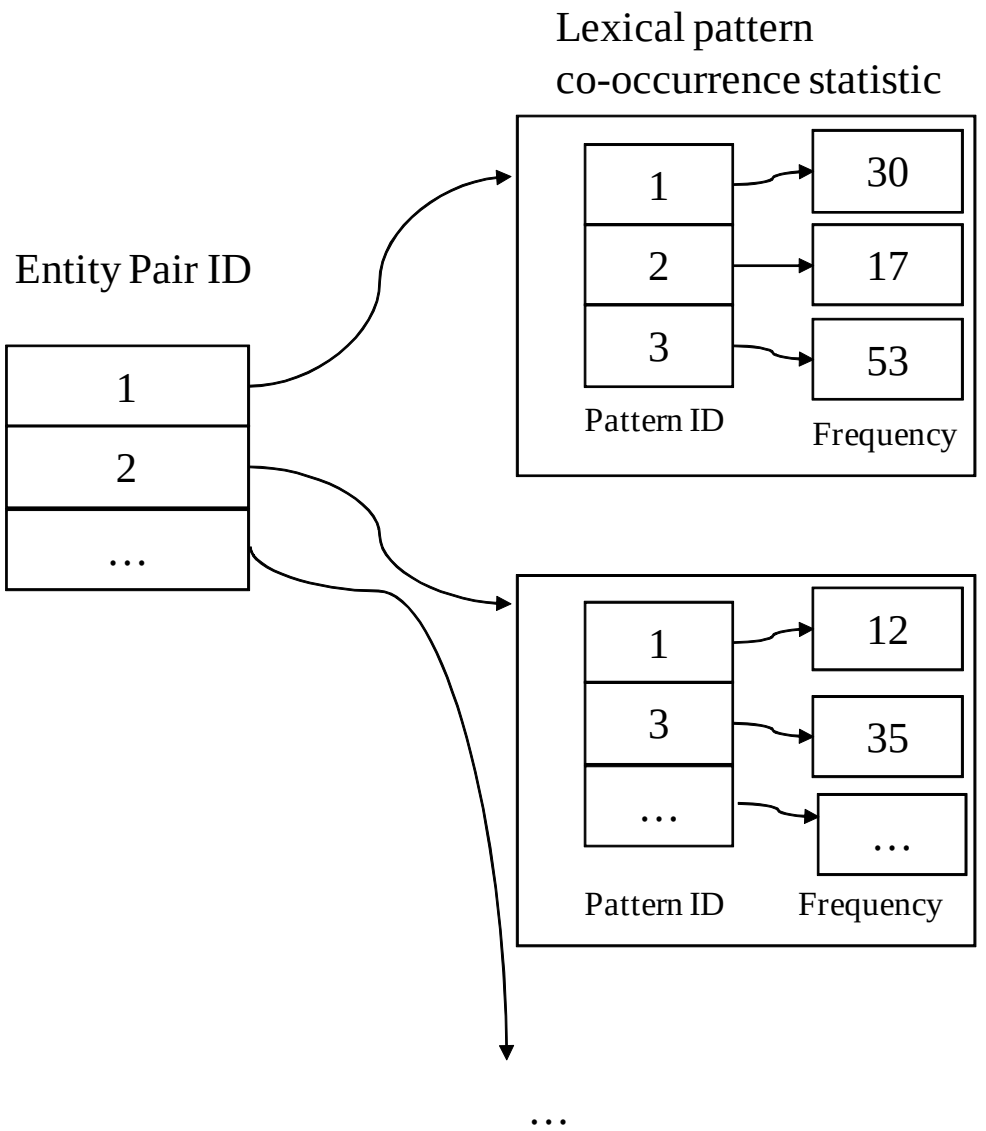


Fig. 4.9. The index from Entity Pairs to Lexical Patterns

This query has multiple correct answers (the companies that Microsoft acquired). For each relation type, we create a data set with more than 50 queries, which correspond to more than 50 information needs. A set of 50 queries is considered to be large enough for evaluating an information retrieval system [11]. Table 4.4 shows some example queries that we created to evaluate the system.

To obtain a Web corpus containing HTML pages that refer to the entities and semantic relations in Table 4.3, we use the following method. Using two entities in each pair and some keywords that describe the semantic relations of the two entities, we formulate some queries for retrieving relevant documents. For example, from the entity pair (Google, YouTube), we formulate the following queries to retrieve documents related to the acquisition of YouTube by Google: "Google buy YouTube", "Google ∗ buy ∗ YouTube", "Google ∗∗ buy ∗∗ YouTube", "Google bought YouTube", "Google ∗ bought ∗ YouTube", "Google ∗∗ bought ∗∗ YouTube", "Google acquired YouTube", "Google purchased YouTube", "ac-

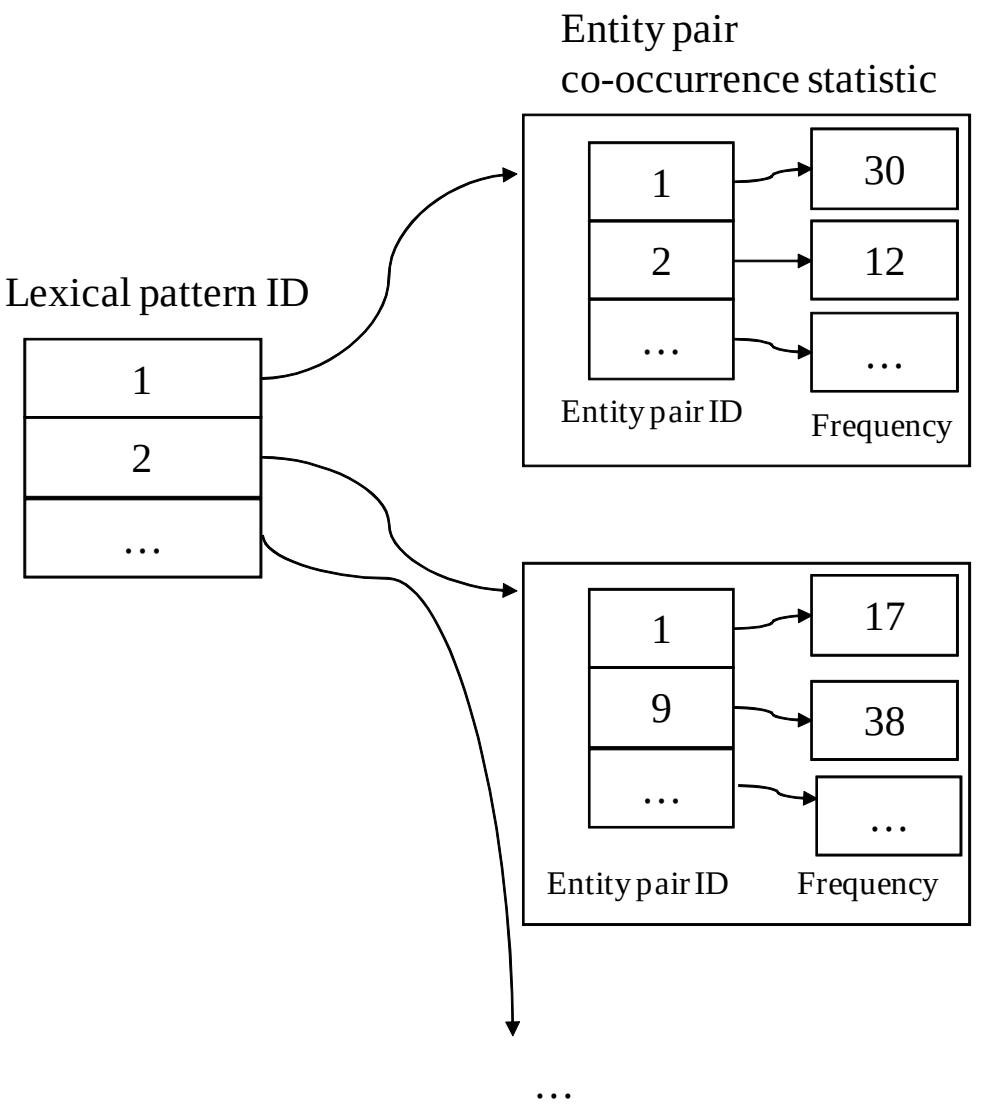


Fig. 4.10. The index from Lexical Patterns to Entity Pairs

quisition ∗ YouTube ∗ Google", ... These queries would retrieve many documents that contain several different paraphrases of the acquisition relation existing between Google and YouTube. Therefore, after the crawling process, we obtain a set of documents that are relevant to the relations in Table 4.3. However, we do not know exactly the number of relations in the corpus because, for example, a document that describes the acquisition of Powerset by Microsoft might also mention the acquisition of Emagic by Apple (even the pair (Apple, Emagic) might not be in Table 4.3). Note that for simplicity in the experiments, we use these queries to retrieve documents that are strongly related to the relation types in Table 4.3, but the proposed system can be run on any corpus, with different relation types. The Extractor in Figure 4.4 does not know in advance what types of relations are in the corpus. It is designed for working without any knowledge about the relations. It can work by blindly crawling the World Wide Web. We will present the performance of the proposed search engine with the entire Wikipedia data dump (which includes nu-

Table. 4.4. Example queries in the query sets for evaluation (italic entities are actually in Japanese). The relations are held in the pairs at the time of the experiments (e.g., Naoto Kan was the Prime Minister of Japan at that time)

Query set	Example queries
BIRTHPLACE (English)	{(Franz Kafka, Prague), (Andre Agassi, ?)} {(Charlie Chaplin, London), (Garry Kasparov, ?)}
BIRTHPLACE (Japanese)	{(*Asada Mao*, *Aichi*), (*Nakata Hidetoshi*, ?)} {(*Hirai Ken*, *Osaka*), (*Mikitani Hiroshi*, ?)}
HEADQUARTERS (English)	{(Google, Mountain View), (Microsoft, ?)} {(Citigroup, New York), (General Electric, ?)}
HEADQUARTERS (Japanese)	{(*Toyota*, *Aichi*), (*Hitachi seisakusho*, ?)} {(*Nintendo*, *Kyoto*), (*Sekisui Hausu*, ?)}
CEO (English)	{(Steve Ballmer, Microsoft), (?, Yahoo)} {(Philippe Dauman, Viacom), (?, Symantec)}
CEO (Japanese)	{(*Yamada Ryuji*, *NTT Docomo)*, (?, *Rakuten*)} {(*Ootsuba Fumio*, *Panasonikku*), (?, *Honda*)}
ACQUISITION (English)	{(Google, YouTube), (?, Powerset)} {(IBM, Cognos), (?, PeopleSoft)}
ACQUISITION (Japanese)	{(*Rakuten*, *Infoseek*), (?, *Nippon Housou*)} {(*Panasonikku*, *Sanyo*), (?, *Kishuseishi*)}
PRESIDENT (English)	{(Barack Obama, U.S), (?, France)} {(Hugo Chavez, Venezuela), (?, Kenya)}
PRESIDENT (Japanese)	{(*Sarudari*, *Pakisutan*), (?, *Ro-si-a*)} {(*Obama*, *Amerika*), (?, *Kankoku*)}
PRIMEMINISTER (English)	{(Australia, Julia Gillard), (Bulgaria, ?)} {(Greece, George Papandreou), (Finland, ?)}
PRIMEMINISTER (Japanese)	{(*Chuugoku*, *Onkaho*), (*Igirisu*, ?)} {(*Nihon*, *Kan Naoto*), (*Betonamu*, ?)}
CAPITAL (English)	{(Tokyo, Japan), (?, France)} {(Berne, Switzerland), (?, Germany)}
CAPITAL (Japanese)	{(*Tokyo*, *Nihon*), (?, *Betonamu*)} {(*Peiking*, *Chuugoku*), (?, *Kankoku*)}
SATELLITE (English)	{(Genymede, Jupiter, (Phobos, ?)} {(Enceladus, Saturn), (Nereid, ?)}
SATELLITE (Japanese)	{(*Ganimede*, *Mokusei*), (*Fobosu*, ?)} {(*Taitan*, *Dosei*), (*Oberon*, ?)}

merous relation types) later in Chapter 6. Using the query set, we query Google[*11] to retrieve the Top 100 URLs that are relevant to each query. From the URL set, we crawl the HTML page at each URL. We use Google to retrieve the relevant documents for evaluation because we want to know what relations are in the corpus in this experiment. After the crawling process, we obtain a training corpus and a test corpus. The training corpus has size of 1.8 GB, of which about 60% are English web pages and 40% are Japanese web pages. The test corpus has size of 1.6 GB, about one-half of which are English web pages, the rest are Japanese web pages. These sets of web pages contain a large number of entities and relations of many types (not only those in Table 4.3 because a web page

[*11] http://www.google.com

might describe many entities and relations and might contain non-related information such as text from advertisements). We use the training corpus for tuning parameters in the proposed method. After we have determined appropriate values for these parameters, we use the test corpus for evaluating the system. We use two different corpora for training and testing to avoid any bias from the parameter tuning process (e.g., the system is only optimized with the training corpus).

4.10.2 Parameter tuning

To determine the appropriate values of the free parameters (the pattern clustering similarity threshold θ_1 and candidate similarity threshold σ), we evaluate our system using four types of relations: the first three relation types in Table 4.3 and the *Acquirer-Acquiree* relation (in which each query has multiple correct answers). For this purpose, we use the training corpus to make an index for the search engine in this experiment. To avoid any interference of Japanese entity pairs and lexical patterns into the monolingual search concerning English entity pairs, we only analyze English documents from the corpus. From the text documents in the training corpus, the system extracted 113,742 entity pairs. The number of extracted lexical patterns is 2,069,121. As suggested by Bollegala et al. [6], we filter out very rare patterns, which frequently contain misspellings for strange symbols.

Effect of lexical pattern clustering

To evaluate the effect of the pattern clustering algorithm to the candidate retrieving and ranking process, we vary the lexical pattern clustering threshold θ_1 and measure precision, recall and F-score at each value of θ_1.

For query sets in which each query has only one correct result, such as $\{(\text{Person}_i, \text{Birthplace}_i), (\text{Person}_j, ?)\}$ $(i \neq j)$, we only evaluate the first result (the top 1 ranked result) of each query. For calculating the precision of a query set Q of this type, we count the number of queries where the system outputs at least one answer (not *"No answer"*). Suppose that the number of these queries is a and the total number of queries in the query set is $|Q|$. The number of queries in this query set that the system gave correct answer at the top 1 ranked result is c. We then define the precision and recall level of this query set as follows:

$$\text{Precision}(Q) = \frac{\text{Number of queries with correct answer at top 1}}{\text{Total number of queries with at least an answer}} = \frac{c}{a}$$

$$\text{Recall}(Q) = \frac{\text{Number of queries with correct answer at top 1}}{\text{Total number of queries}} = \frac{c}{|Q|}$$

The F-score of the query set Q is defined as follow:

$$\text{F-score}(Q) = 2 \cdot \frac{\text{Precision}(Q) \cdot \text{Recall}(Q)}{\text{Precision(Q)} + \text{Recall(Q)}}$$

For the Acquirer-Acquiree relation, we evaluate the system with queries of type $\{(\text{Acquirer}_i, \text{Acquiree}_i), (\text{Acquirer}_j, ?)\}$ $(i \neq j)$. Note that these queries have multiple correct answers (correct answers are in the set of companies that were acquired by Acquirer_j). We investigate the top 10 answers of each query (if the number of answers is smaller than 10, we consider only this number of answers). The precision of each query in this set is defined as the number of correct answers c divided by the total number of results that the system outputs a. The precision for a query set of this type is defined as the average of the precision of each query in the set. The recall for this test can not

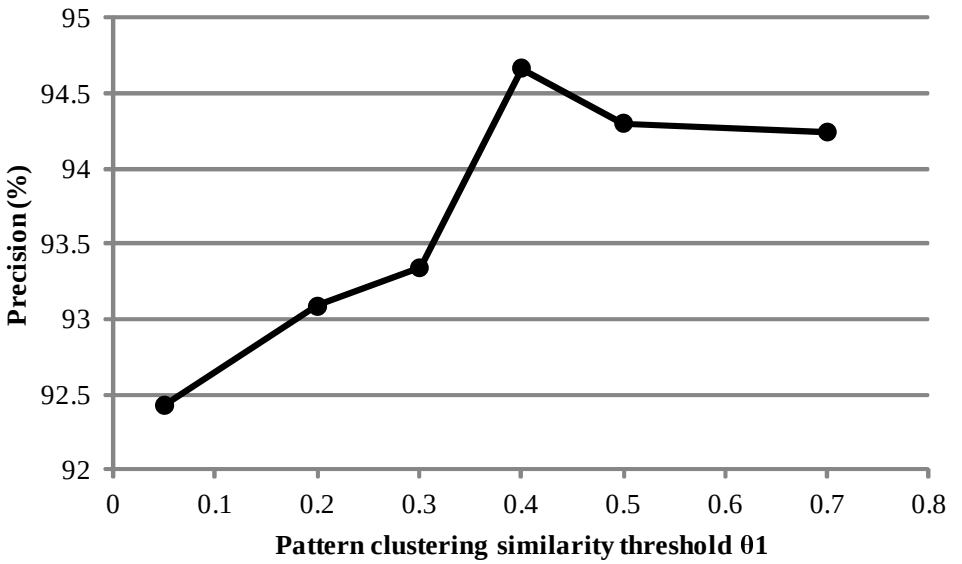


Fig. 4.11. Average of precision of four query sets while varying the pattern clustering similarity threshold θ_1 (at $\sigma = 0.05$)

be calculated because we can not determine the number of correct answers in the corpus and we only take the top 10 candidates.

Figure 4.11 shows the average of the precision of four query sets (BIRTHPLACE, HEADQUARTERS, CEO and Acquirer-Acquiree) at each value of θ_1 (when $\sigma = 0.05$). Figure 4.12 shows the average F-score of three test sets that we can calculate recall (BIRTHPLACE, HEADQUARTERS and CEO). When θ_1 is 0.4, we obtain the maximum value of both the average of precision values and the average F-score. We observe that at $\sigma = 0.05$, we achieve the best F-score. If σ is too low (e.g., 0.01) then there are many noisy candidate pairs which do not actually hold the desired semantic relations with the source pair in the query set with multiple correct answers. Moreover, the query processing time is also long because the candidate set is too large. On the other hand, when σ is too large (e.g., above 0.2), the recall is very small and hence the F-score drastically decreases. Therefore, we fix the parameter σ at 0.05 in all subsequent experiments.

4.10.3 PMI vs. Frequency as feature vector values

In this section, we compare the performance of the search engine when using PMI and the number of co-occurrences between entity pairs and lexical patterns as elements of the matrix $\mathbf{M}$ (in Table 4.2). It is enough to verify the difference for English query sets, as the behavior does not depend on the language of the queries.

To be aligned with previous work on latent relational search [8], we use Mean Reciprocal Rank (MRR) of each query set as the criterion for evaluation in this experiment. MRR reflects both recall and precision of a search engine and is frequently used for evaluation of high accuracy retrieval techniques [110, 111, 112, 8]. For a query set Q, if the rank of the first correct answer in a query $q \in Q$ is r_q, then the Mean Reciprocal Rank (MRR) of Q is:

$$\mathrm{MRR}(Q) = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{r_q} \tag{4.33}$$

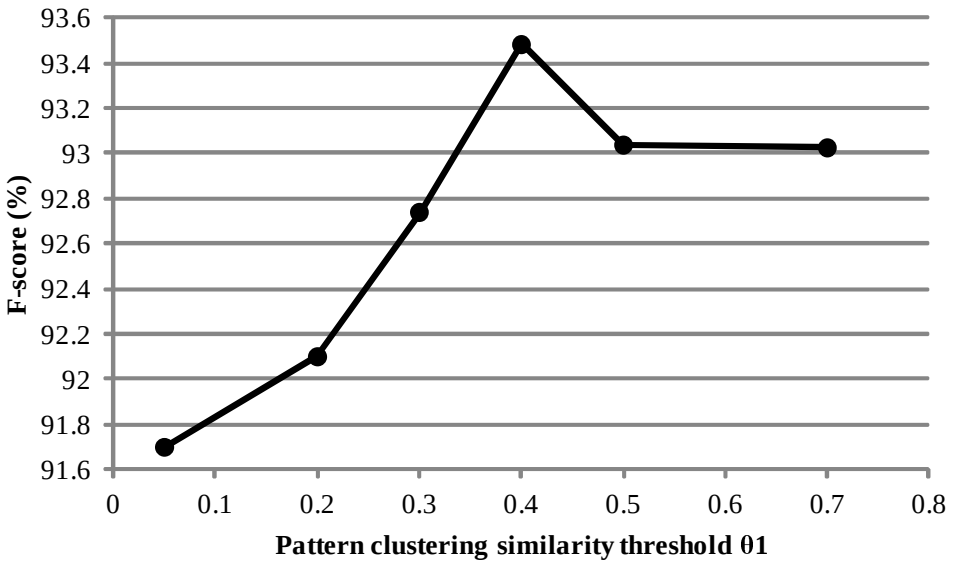


Fig. 4.12. Average F-score of three query sets corresponding to three relation types (BIRTHPLACE, HEADQUARTERS, CEO) while varying the pattern clustering similarity threshold θ_1 (at $\sigma = 0.05$)

Here, $|Q|$ denotes the number of queries in the query set Q. Therefore, the MRR is the average of the first correct answer's inverse ranks. When all queries have correct answers ranked at the top 1, the MRR achieves the maximum value of 1.0. The higher the MRR, the better the performance because the inverse ranks of correct answers are higher (that is, the correct answers are ranked near the top of the answer list).

We compare the mean reciprocal rank (MRR) of four monolingual query sets (corresponding to the first four relation types in Table 4.3: BIRTHPLACE, HEADQUARTERS, CEO and ACQUISITION) when the system uses frequencies and PMIs as feature vector values (in all of these query sets, each query has only one correct answer).

When using PMIs as feature values, we achieve an average MRR of 0.989 on four monolingual query sets, whereas, when using numbers of co-occurrences, this value is 0.963. Therefore, the value of MRR is only slightly different. However, when we evaluate the system with the Top 10 results of queries that have multiple correct answers, there is a significant difference between the two methods. We use the *Acquirer-Acquiree* query set which contains queries of the form $\{(A, B), (C, ?)\}$ of the acquisition relation for this purpose. Note that the Acquirer-Acquiree query set is different from the ACQUISITION query set: each query in this set has multiple correct answers. For example, the answers for the query {(Google, YouTube), (Microsoft, ?)} are companies that are acquired by Microsoft. For the Acquirer-Acquiree query set, the method based on numbers of co-occurrences achieves a precision of 81.34% in the Top 10 results, whereas, the method based on PMIs achieves 88.06%. This shows that using PMIs as feature vector elements improves the precision of queries with multiple correct answers.

4.10.4 The effectiveness of the proposed extraction algorithm

As described in Section 4.2, we make two modifications to the baseline pattern extraction algorithms [29, 20]: we stem the input sentence before extraction and we eliminate the

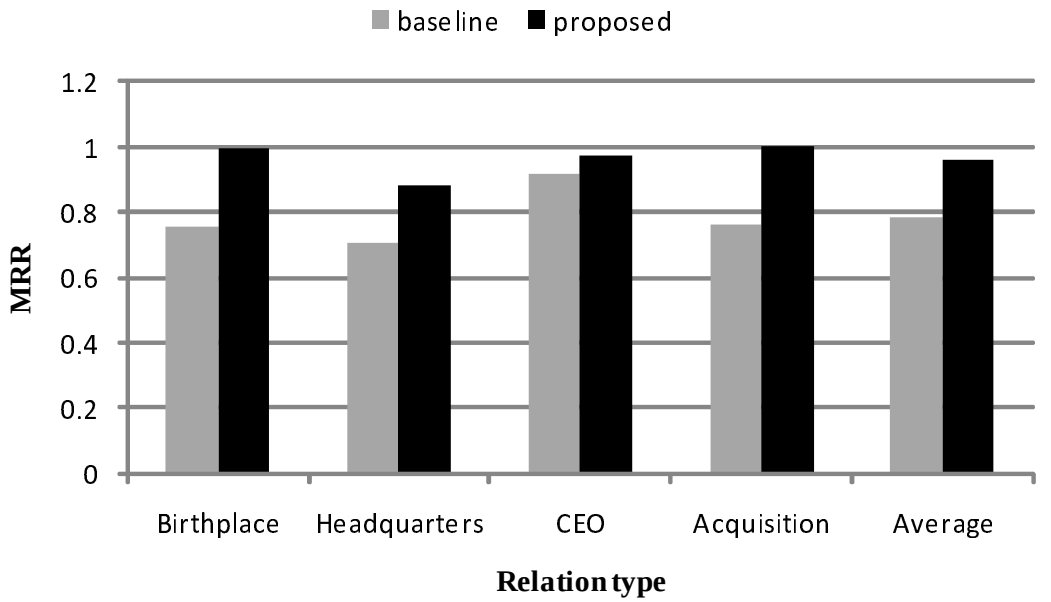


Fig. 4.13. Comparison between the performance of the search engine on four monolingual query sets while using the proposed lexical pattern extraction algorithm and the baseline algorithm

condition that a pattern must contain both X and Y (instead we add the prefix "$X*$" if the pattern contains Y, but does not contain X and the subfix "$*Y$" if the pattern contains X, but does not contain Y).

To evaluate the effect of these changes, we compare the performance of the search engine when we use our extraction algorithm and the baseline algorithm as described in [29, 20]. The baseline algorithm does not stem the input sentence and does require that a pattern must contain both X and Y. We compare the performance only on English monolingual latent relational search query sets because the previous algorithm is proposed for English pattern extraction. Moreover, we want to prevent any interference of Japanese lexical patterns, which can blur the difference between the proposed extraction algorithm and the baseline.

We compare the mean reciprocal rank (MRR) of four monolingual query sets (similar to those in the previous section) when the system runs with the proposed extraction algorithm and the baseline.

The comparison between MRR of the search engine with each pattern extraction algorithm is shown in Figure 4.13. For the BIRTHPLACE and ACQUISITION relation, the proposed extraction algorithm significantly outperforms the baseline algorithm. This is because English phrases that describe these relations contain various inflected forms of a word (e.g., "acquires", "acquired") and the text inside the gap between X and Y is complex (e.g., "X was born and risen up in Y", "X was born in 1948 in Y"). The baseline algorithm could not extract many common patterns in these cases. On the other hand, the proposed algorithm is able to extract many common patterns (e.g., "X was born * Y") in these cases. For the CEO relation, the difference is not significant. This is because the CEO relation is often referred by some patterns such as "X, CEO of Y", "X - the CEO of Y", which do not contain inflected forms of words and are not complex. The results prove that the proposed extraction algorithm works well for all types of relations, whereas, the

Table. 4.5. Performance of the proposed method on English monolingual query sets in which each query has only one correct answer (@N is the percentage of queries where the correct answer is in the Top N results)

Relation type	MRR	@1	@5	@10	@20
BIRTHPLACE	0.962	93.0	100.0	100.0	100.0
HEADQUARTERS	0.970	94.0	100.0	100.0	100.0
CEO	0.963	94.0	99.0	99.0	100.0
ACQUISITION	0.973	96.0	100.0	100.0	100.0
PRESIDENT	0.969	94.4	99.2	99.2	99.2
PRIMEMINISTER	1.000	100.0	100.0	100.0	100.0
CAPITAL	0.963	93.0	100.0	100.0	100.0
SATELLITE	0.935	88.0	100.0	100.0	100.0
Average	0.967	94.1	99.8	99.8	99.9

baseline only achieves high performance for relations in which the lexical patterns are not complex.

4.10.5 Performance on each query set

We fix the free parameters θ_1 and σ at the values that gave the best performance in the parameter tuning phase (we set θ_1 to 0.4 and σ to 0.05) and use a completely different corpus (the corpus for testing) to evaluate the performance of the proposed system. We use a different corpus in order to verify the parameter tuning process and prevent the bias to the corpus that is used for parameter tuning. The corpus for testing performance also contains Web pages regarding the same relation types with the corpus in the parameter tuning phase (the relation types are shown in Table 4.3), but the relation instances are completely different from those in the first experiment (i.e., the entity pairs are different). Because PMIs yield better result than Frequencies, we use PMIs as feature vector values in this experiment.

Following Kato et al. [8], we also evaluate the MRR and the percentage of queries where the system retrieved the correct answers at the Top 1, Top 5, Top 10 and Top 20 ranked results. For easily to compare the performance with the method in Kato et al., we only evaluate the system with query sets in which each query has only one correct answer (as Kato et al. did in their work [8]).

We evaluate the system with eight English monolingual query sets and eight Japanese monolingual query sets, as shown in Table 4.4. Table 4.5 and Table 4.6 show the evaluation result of the system. In each table, MRR is the Mean Reciprocal Rank of each query set. @1, @5, @10, @20 are the percentages of queries with correct answers in the Top 1, Top 5, Top 10 and Top 20 ranked results, respectively. Table 4.7 presents the average result of English and Japanese monolingual query sets. We achieve high MRR on all English monolingual query sets. Moreover, on these query sets, the system retrieves the correct answers at the Top 1 ranked result for 94.1% of the queries. We also achieve high MRR on the majority of Japanese monolingual query sets. The MRR of the Japanese monolingual query set concerning headquarters of companies (HEADQUARTERS) is lowest (0.620). This is because in Japanese, there are many different paraphrases to state the HEADQUARTERS relation (for example, "*X ha Y ni honsha wo oku*", "*X (honsha Y)*", etc. Moreover, the company-headquarters relation is not frequently mentioned in Japanese sentences. Therefore, the lexical pattern extraction process could not extract enough lexical patterns to exactly measure the relational similarity. Similarly, the PRES-

Table. 4.6. Performance of the proposed method on Japanese monolingual query sets in which each query has only one correct answer (@N is the percentage of queries where the correct answer is in the Top N results)

Relation type	MRR	@1	@5	@10	@20
BIRTHPLACE	0.840	80.0	88.0	88.0	88.0
HEADQUARTERS	0.620	60.0	64. 0	64.0	64.0
CEO	1.000	100.0	100.0	100.0	100.0
ACQUISITION	0.960	92.0	100.0	100.0	100.0
PRESIDENT	0.680	68.0	68.0	68.0	68.0
PRIMEMINISTER	1.000	100.0	100.0	100.0	100.0
CAPITAL	1.000	100	100.0	100.0	100.0
SATELLITE	1.000	100.0	100.0	100.0	100.0
Average	0.888	87.5	90.0	90.0	90.0

Table. 4.7. Average performance of the proposed method on monolingual query sets in which each query has only one correct answer (@N is the percentage of queries where the correct answer is in the Top N results)

Relation type	MRR	@1	@5	@10	@20
BIRTHPLACE	0.901	86.5	94.0	94.0	94.0
HEADQUARTERS	0.795	77.0	82.0	82.0	82.0
CEO	0.982	97.0	99.5	99.5	100.0
ACQUISITION	0.967	94.0	100.0	100.0	100.0
PRESIDENT	0.825	81.2	83.6	83.6	83.6
PRIMEMINISTER	1.000	100.0	100.0	100.0	100.0
CAPITAL	0.982	96.5	100.0	100.0	100.0
SATELLITE	0.968	94.0	100.0	100.0	100.0
Average	0.927	90.8	94.9	94.9	95.0

IDENT relation also has multiple paraphrases in Japanese. For example, the phrase for referring to the President of China or Vietnam is written as "*kokka shuseki*", but the phrase for U.S or France president is "president of" ... These lexical patterns are very difficult to be clustered into the same cluster because they often do not share the entity pair set. Consequently, the performance of the system on this monolingual query set is low. On other Japanese monolingual query sets, we also achieve high performance as in English query sets.

Table 4.8 shows some example queries and results that the system outputs in the evaluation process. In some cases, the system outputs a reasonable answer, but it is not a correct answer. For example, for the query {(Germany, Angela Merkel), (Japan, ?)}, the system outputs "Junichiro Koizumi" as the first answer. This is because the frequency of the entity pair (Japan, Junichiro Koizumi) is very high and therefore the system extracts a large number of lexical patterns concerning this pair. However, this answer is only correct in the past, when Junichiro Koizumi was the Prime Minister of Japan. To alleviate this problem, we must also recognize the change of semantic relations across time. Detecting the time that an article is written is an active research topic and is beyond the scope of this thesis. It would be an important future research direction of latent relational search, in which we can integrate temporal aspects of semantic relations into the relational similarity calculation process.

Table. 4.8. Example queries and results

Query	Result	Correct?	Common patterns
{(Franz Kafka, Prague), (Albert Einstein, ?)}	Ulm	Yes	X * born in Y; X wa born in Y; X wa born in Y in; X *, born * Y ...
{(Yahoo, SunnyVale), (Apple, ?)}	Cupertino	Yes	X * headquart in Y, calif.; X * locat in Y; at X headquart in Y ...
{(Michael Dell, Dell), (?, Microsoft)}	Steve Ballmer	Yes	X, CEO of Y; X *, chairman * Y; X, chairman and CEO * Y ...
{(Microsoft, Hotmail), (?, YouTube)}	Google	Yes	X acquir * Y, X bought Y, X's acquisit of Y, X announc * Y, X * plan to * Y ...
{(Germany, Angela Merkel), (Japan, ?)}	Junichiro Koizumi	No	X prime minist Y, X prime minist is Y, X * prime minist * Y ...

Another issue that causes errors is that the extraction algorithm might ignore negative or speculative aspects of a relation. For example, from the sentence "Microsoft withdraws the proposal to acquire Yahoo.", the proposed extraction algorithm also extracts the pattern "X $*$ acquire Yahoo". This pattern might cause the entity "Yahoo" to be appeared in the results for the query {(Google, YouTube), (Microsoft, ?)} (Yahoo is an incorrect result, regarding the acquisition relation). Another example is that from the sentence "Kyoto is an ancient capital of Japan", the extraction algorithm also extracts the pattern "X * capital of Y", which might be mistakenly recognized as the pattern for representing the relation between a (current) capital and a country. However, if there are other entities which are consistently mentioned in positive sentences (such as "Microsoft acquires Powerset" or "Tokyo is the capital of Japan") then the relevance scores for those entities will be higher than the scores for the entities in the negative or speculative sentences. This indicates that, the system will rank the correct entities above the mistakenly recognized entities in the result list. Furthermore, it might be acceptable to have those mistakenly recognized entities in the result list because they can be considered as "correct" answers in some extent. This also improves the variety of the search results, which might be of interest to search engine users (e.g., while searching for the current capital of a country, a user might also be interested in the ancient capitals). We will discuss a future research direction for recognizing negative and speculative sentences to improve the precision of latent relational search in Section 7.2.3.

4.10.6 Comparison with previous method

Table 4.9 shows the comparison between the performance of the proposed method with the method of Kato et al. [8]. The MRR of the proposed method on Japanese monolingual query sets outperforms the previous method (Kato et al.) by a wide margin (0.888 vs. 0.545).

The proposed method also outperforms the previous method in other criteria such as @1, @5, @10 and @20. Especially, the proposed method ranks the correct answer in the

Table. 4.9. Comparison between the proposed method with previous method on monolingual query sets (@N is the percentage of queries where the correct answer is in the Top N results).

Method	MRR	@1	@5	@10	@20
Kato et al. [8] (Japanese)	0.545	43.3	68.3	72.3	76.0
Proposed method (Japanese)	0.888	87.5	90.0	90.0	90.0
Proposed method (English)	0.967	94.1	99.8	99.8	99.9
Proposed method (average J+E)	0.927	90.8	94.9	94.9	95.0

Top 1 for 87.5% of Japanese monolingual queries. For English monolingual queries, the proposed method achieves an MRR of 0.967 and it ranks correct answers in the top 1 result for 94.1% of the queries. The detailed performance for MRR, @1, @5, @10 and @20 of the proposed method are shown in Table 4.5, Table 4.6 and Table 4.7.

We obtained high MRR and percentage of queries with correct answer in Top 1 because the proposed lexical pattern extraction algorithm works well. Entity pairs with similar semantic relations might have slightly different lexical patterns in the gaps between two entities in the pairs (e.g., "*Barrack Obama* is the 44th and current president of the *U.S*" and "*Nicolas Sarkozy* is the current president of *France*"). In this situation, the method **SP** (based on exactly matched lexico-syntactic patterns) by Kato et al. [8] as described in Section 3.4 gives a small relational similarity because it considers only the exactly matched lexical patterns in the gap between two entities. On the other hand, the proposed lexical pattern extraction algorithm gives a high relational similarity for these pairs because it generates many patterns that are matched in two pairs (e.g., *X * president * Y, X * president of * Y, X * current president * Y, X * current president of * Y*). The proposed algorithm also eliminates too general patterns such as *"X is * Y"* or *"X is the * Y"*, ... (because the algorithm filters out patterns that contain no content word). Moreover, while the method **SP** considers only lexical patterns inside the gap between two entities, the proposed method also considers lexical patterns surrounding these entities. This results in a more accurate semantic relation representation, and hence a higher precision in retrieving and ranking candidates.

In addition, the proposed method represents the semantic relations between two entities in an entity pair more precisely than the method **TC** (based on term co-occurrences) by Kato et al. [8]. As described in Section 3.4, the method **TC** does not maintain the order of words in the context between two entities. Therefore, it might extract inappropriate words for representing the semantic relations. On the other hand, the proposed method maintains the order of words in the context, thereby alleviating this problem.

Finally, the method **CNJ** (which is a combination (conjunction) between **TC** and **SP**) in [8] tries to combine the strengths of **TC** and **SP**. Specifically, the final rank of an entity D_i against a query $q = \{(A, B), (C, ?)\}$ in the method **CNJ** is defined as follows:

$$\text{Rank}_{CNJ}(q, D_i) = w_1\text{Rank}_{TC}(q, D_i)+w_2\text{Rank}_{SP}(q, D_i)+w_3\text{Rank}_{TC}(q, D_i)\text{Rank}_{SP}(q, D_i) \tag{4.34}$$

in which w_1, w_2, w_3 are coefficients and are heuristically set to 0.90, 0.50 and 0.80, respectively. That is, the rank in the **CNJ** method is a heuristic combination of the ranks in **TC** and **SP**. There might be several methods for combining the ranks in **TC** and **SP**, such as using a linear combination of the ranks in **TC** and **SP** or using the product of those ranks. Each combination in turn has it own parameters, which we must determine to optimize the performance. Consequently, it is not simple to experimentally optimize those parameters for several types of relations.

To this end, the proposed lexical extraction method has an advantage that it directly combines both of the strengths of the method **SP** and the method **TC** in Kato et al. [8] (i.e., it precisely captures the semantic relations and it allows flexible pattern matching), without requiring to heuristically determine any parameters. This combination is not trivial because it must solve the two conflicting properties of the two methods **TC** and **SP**. That is, **TC** allows retrieving a wide range of answers with low precision and **SP** allows us to precisely retrieve answers but with low recall.

Furthermore, by relying on the extraction method proposed in this work to measure the relational similarity, we only need to calculate the relational similarity (and hence the rank) for each candidate answer only once, instead of separately calculating Rank_{TC} and Rank_{SP}. This indicates that proposed method is more appropriate for achieving a practical query processing time.

4.10.7 Scalability and query processing time

The proposed system can scale to the Web because of the following reasons. First, the extraction algorithm scans through each document only once. It also requires only one pass through all documents in the corpus. Moreover, the extraction algorithm can be executed in parallel for a corpus (each machine processes a subset of documents, as describe latter in Chapter 6). This makes it feasible for processing a huge corpus such as the Web.

Second, the amortized time complexity of the pattern clustering algorithm is $O(n \log n)$, where n is the number of lexical patterns to be clustered. This time complexity is acceptable because it slowly grows with the number of lexical patterns in the index. Other clustering algorithms, such as k-means clustering or agglomerative hierarchical clustering take much more time.

Third, the algorithm for calculating the relational similarity between two entity pairs (Algorithm 4) runs in constant time with respect to both the number of entity pairs and the number of lexical patterns in the index (it is independent from the number of entity pairs; it depends on the average number of lexical patterns for each entity pair, but this number does not depend on the total number of lexical patterns). Therefore, the time complexity of the candidate retrieving and ranking process depends only on the number of candidates, which in turn is limited by the filtering condition as shown in Equation 4.19.

To evaluate the retrieval speed of the proposed method, we measure the time for processing a query as follow. We get the timestamp (t1) at the beginning of the entity retrieval process for a query and another timestamp (t2) at the end of the process. We then assume that the time for processing this query is equal to (t2-t1) (we ignore any overhead incurred by function calls to get the timestamps). As explained in Section 4.8, we can separate the supporting sentence retrieval process from the entity retrieval process. Therefore, we do not account the time for retrieving sentences into the query processing time. For each relation type, we randomly take 100 queries in both Japanese and English to measure the average query processing time. The pseudo code for the time measuring experiment is shown in Figure 4.14.

The result of the above experiment is shown in Table 4.10. The average query processing time is 0.64, but the standard deviation is large (1.14). This is because for a rare number of queries, the query processing time is very large (about 16 seconds). The rest of the queries have very small query processing time (e.g, 0.6). The queries that take long query processing times are query related to very popular entity pairs. These pairs have a large number of extracted lexical patterns, in which many patterns can be seen in different types of semantic relations (e.g., the pattern "X is one of the * Y" can be seen in "X is one of the city of Y", "X is one of the CEO of Y", ...). This leads to a large candidate set containing a huge number of entity pairs which we must compute the relational similarities.

```
query_set = randomly take 100 queries;
sum_time = 0.0;
for each query q in query_set do
    t1 = gettimeofday();
    results = process_query( q );
    t2 = gettimeofday();
    t = (t2 - t1);
    sum_time += t;
end do
avg_time = sum_time / 100.0;
```

Fig. 4.14. Pseudo-code for measuring the query processing time

Table. 4.10. Average query processing time for each relation type (the standard deviation is the value for a sample of 100 queries in both English and Japanese)

Relation type	Query processing time (s)
BIRTHPLACE	0.12 ± 0.12
HEADQUARTERS	0.51 ± 1.55
CEO	0.83 ± 1.54
ACQUISITION	0.54 ± 0.25
PRESIDENT	0.79 ± 1.51
PRIMEMINISTER	0.56 ± 0.23
CAPITAL	1.33 ± 1.56
SATELLITE	0.49 ± 0.21
Average	0.64 ± 1.14

However, the number of these queries is very small (one or two for each relation type). Therefore, on average the query processing time is very small in this implementation. This is because we store all important information for the candidate retrieval process in RAM. We do not need to look up secondary storage when processing a query in this prototype implementation.

Chapter 5

Retrieval Model for Cross-Lingual Latent Relational Search

5.1 Retrieval Model

The general retrieval model for cross-lingual latent relational search is the same with the model for monolingual latent relational search in Equation 4.7:

$$\text{Rank}(q, D) = \mathbf{F}_e(q, D) \times \mathbf{R}_e(q, D)$$

In the equation, $\mathbf{F}_e(q, D)$ is the entity filtering function and $\mathbf{R}_e(q, D)$ is the ranking function, as described in Section 4.1 of Chapter 4. However, the methods to calculate the entity filtering function $\mathbf{F}_e$ and the entity ranking function $\mathbf{R}_e$ are different with those of monolingual latent relational search. This is because in monolingual latent relational search, it is only required to compare lexical patterns in the same language, where as, in cross-lingual latent relational search, lexical patterns in different languages must be compared.

The method for representing the semantic relations between two entities in an entity pair is the same as in monolingual latent relational search. That is, the semantic relations are represented by lexical patterns of the context surrounding the entity pair. For example, from the sentence "Steve Jobs, a co-founder of Apple, is the CEO of the company now.", we extract the entity pair (Steve Jobs, Apple) and lexical patterns such as "X, a co-founder of Y", "X, a co-founder of Y, is the CEO", ... In these patterns, we replace the original entities by the symbols X and Y to make the lexical patterns independent from the entity pair.

In many cases, Japanese language also uses an identical orthography to represent an entity name. For example, in many sentences, Japanese write the name of the Google Inc. as "Google", instead of the Katakana expression グーグル. We propose an indexing method that exploits this phenomenon to capture the semantic similarity between lexical patterns in different languages. Because in several Japanese sentences, "グーグル" is also written as "Google", "ユーチューブ" is also written as "YouTube", the entity pair (Google, YouTube) co-occurs with both English and Japanese lexical patterns. Therefore, we can group all lexical patterns of the pair (Google, YouTube) in Japanese with those in English, and consider them to express the same set of semantic relations between Google and YouTube. We call this method "multi-lingual entity pair and pattern indexing", as described in Section 4.2. The above process results in the matrix $\mathbf{M}$ as shown in Table 4.2 of Chapter 4. Despite the fact that many lexical patterns in two different languages might be grouped together after the multi-lingual indexing process, there might be many noisy patterns (that do not express the same relation) including in these groups. Therefore,

we improve the quality of our cross-language relational similarity measuring algorithm by trying to find parallel pattern pairs in the two languages. We use machine translation to achieve this goal. The proposed method only uses machine translation to translate very simple lexical patterns, not long and complex sentences. Translating short lexical patterns can be done with a good precision in a high speed.

Next, we merge any two rows that correspond to two parallel patterns in **M** and we also merge any two columns that correspond to two parallel entity pairs to create a matrix **A**. We then apply Singular Value Decomposition to the matrix **A** as can be seen in Latent Relational Analysis (LRA) [29]. Latent Relational Analysis tries to reduce the dimension of each row and column vector to a small number (e.g., 300) to eliminate noisy co-occurrences and to compress semantically similar dimensions into one.

We propose a two-phase lexical pattern clustering algorithm to 1) capture the semantic similarity between paraphrased lexical patterns in the same language and 2) capture the semantic similarity between similar or translated patterns across languages. The second phase helps us to transfer semantic relations across languages. Therefore, the retrieval model for cross-lingual latent relational search is basically the same with the retrieval model for monolingual latent relational search, except that the clustering algorithm is different. Specifically, we add another phase (the second phase) into the clustering algorithm to recognize paraphrased lexical patterns across languages. With only this extension, the retrieval model in the previous chapter can be used for processing cross-lingual latent relational search queries. The similarity between two lexical patterns for clustering is measured in the dimensionally reduced vector space after LRA. We then use the result of the clustering algorithm to measure the relational similarity between the source entity pair and each candidate target pair. The source pair and the target pair can be in two different languages. Therefore, we call this process the "cross-language relational similarity measuring" process. We rank the candidate answer set using the relational similarity scores that we obtained by the cross-language relational similarity measuring algorithm to achieve a ranked result list for a query.

In following sections, we will describe in detail each step of the proposed method.

5.2 Entity Pair and Lexical Pattern Translation

Although we can extract potentially similar lexical patterns in different languages with multi-lingual entity pair and lexical pattern indexing, we might also extract noisy patterns that do not describe the semantic relation between two entities in an entity pair. To make the lexical pattern clustering process more accurate, we propose an entity pair and lexical pattern translation method that can find parallel patterns and entity pairs with high precision. Figure 5.1 shows the overview of the proposed method for entity pair and lexical pattern translation.

To find parallel entity pairs, such as (Japan, Mt. Fuji) and (日本, 富士山), we first translate all entities that are extracted. We use a Statistical Machine Translation (SMT) system to translate or transliterate an entity pair from a source language into a target language. Specifically, we experimentally use Google Translate[*1] and the machine translation system Moses [113] as the SMT system to show that the performance of the proposed method does not heavily depend on the underlying machine translation system. An SMT system is able to find the transliteration of an entity because it can find the alignment between the source entity and the target entity in its corpus. For example, using the SMT system, we can transliterate the entity "Google" in English into "グーグル" in Japanese

[*1] http://translate.google.com/

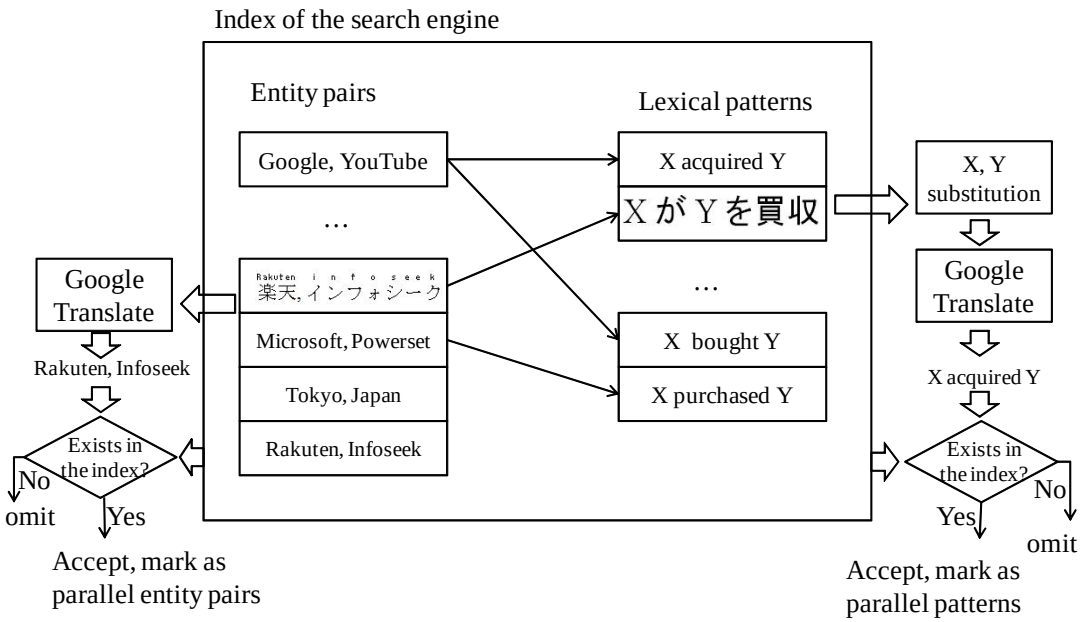


Fig. 5.1. Overview of the entity pair and lexical pattern translation method

and vice versa. Similarly, we can translate the entity "Japan" in English into "日本" in Japanese. Previous work on automated translation of semantic relations [96] shows that by translating only entity pairs, we can find parallel lexical pattern clusters with a reasonable precision. Therefore, the entity pair translation phase contributes a significant value to the final phase of transferring semantic relation across languages. After translating all entities, we can combine the translation results to find the translation of an entity pair. If an entity pair (A, B) is translated into (A', B') by the SMT system, we look up the target entity pair (A', B') in the index to verify the translation result. If we can actually find the pair (A', B') in the index, then we record that the entity pair (A', B') is a parallel entity pair of the entity pair (A, B).

We then merge two columns of the matrix $\mathbf{M}$ that correspond to these entity pairs, as shown in Table 5.2. The resulting column corresponds to both (A, B) and (A', B') (i.e., two entity pairs are now assigned a same column ID). The effect of merging parallel entity pairs is that two similar lexical patterns will have a higher similarity. For example, if we merge the first column and the second column of the matrix $\mathbf{M}$ in Table 5.1 then the cosine similarity of the second row (X が Y を買収, meaning "X acquires Y") and the third row (X bought Y) is increased.

Although recent work on automated semantic translation [96] has proposed the use of parallel entity pairs in the lexical pattern translation process, it does not optimize the entity pair translation method for latent relational search. Specifically, the method proposed by Davidov and Rappoport [96] uses a set of dictionaries for entity pair translation. However, in many applications of information retrieval and question answering such as latent relational search, the majority of target entities are named entities. Therefore, it is difficult to use dictionaries for translating these entities. Moreover, each dictionary in the set of dictionaries might give a different translation result, which corresponds to a sense of the word. The word sense preference of each dictionary system might be different from other systems. Therefore, it is difficult to aggregate all these results to find a consistent translation in case when multiple translation results are obtained. Finally,

Table. 5.1. The pattern vs. entity pair matrix **M**, as shown in Section 4.2

Pattern vs. Entity pair	(Google, YouTube)	(グーグル, ユーチューブ)†	(Microsoft, Powerset)	(日産, 横浜)‡	(Yahoo, Sunnyvale)
X acquires Y	80	0	70	0	0
X が Y を買収	6	75	2	0	0
X bought Y	67	0	60	0	0
X is headquartered in Y	0	0	0	0	105
XはYに本社を置く	0	0	0	90	0

Note: Each Japanese lexical pattern is the translation of the corresponding pattern above it.
† meaning (Google, YouTube) ‡ meaning (Nissan, Yokohama)

Table. 5.2. The pattern vs. entity pair matrix after translation and merging (**A**).

Pattern vs. Entity pair	(Google, YouTube) = (グーグル, ユーチューブ)	(Microsoft, Powerset)	(日産, 横浜)‡	(Yahoo, Sunnyvale)
X acquires Y = X が Y を買収	161	72	0	0
X bought Y	67	60	0	0
X is headquartered in Y = XはYに本社を置く	0	0	90	105

Note: Parallel patterns and entity pairs are marked with the equal (=) symbol.
‡ meaning (Nissan, Yokohama)

dictionaries often fail to translate named entities, which are important targets of latent relational search. Instead of using dictionaries, we use a Statistical Machine Translation system for translating entities from a source language into a target one. Because an SMT system finds the translation of an entity based on the alignments between parallel text, there is a high probability that it gives correct results when translating named entities. Consequently, the proposed method for finding parallel entity pairs is more optimized for latent relational search than the previous work by Davidov and Rappoport [96].

Because the semantic relations between two entities in an entity pair are represented by lexical patterns of the context surrounding the two entities, parallel lexical patterns play an important role in the process of transferring semantic relations across two languages. Especially, in this work, lexical patterns are proxies for representing semantic relations. Therefore, if we can precisely find parallel patterns in two languages, we can identify semantic relations across languages. Consequently, we propose a method to translate lexical patterns from a source language into a target language with high precision. It is worth noting that, previous work on automated translation of semantic relations [96] also uses lexical patterns to represent the semantic relations between entities. However, the previous work does not directly translate lexical patterns, but it translates entity pairs then infers the translation of lexical patterns. The advantage of the above method is that, it does not require a machine translation system. On the other hand, the disadvantage is that, the precision in the translation process might be low. Errors in lexical pattern translation will result in an enormous loss in precision of cross-lingual latent relational

search. Therefore, in cross-lingual latent relational search, directly translating lexical patterns is preferred.

To find the parallel pattern of a lexical pattern, we use the information concerning POS tags (or named entity tags) of the two variables X and Y in the pattern to replace each variable by a well-known entity. For example, if the lexical pattern is from an English document and the tag of X is ORGANIZATION then we replace X with the entity "Microsoft". We remember that the transliteration of "Microsoft" in Japanese is "*Maikurosofuto*" (the lexical pattern must actually be written in Japanese characters (Katakana) but for convenience, we write the pattern using the English alphabet) to use later. We replace X and Y with well-known entities because we want to increase the performance of the SMT system. If we input the original pattern with X and Y not replaced, the SMT system will have neither information about the named entity tags nor the alignments of X and Y in the target language. Therefore, we do not translate the entity markers X and Y in lexical patterns, but replace X and Y with well-known entities to let the SMT system easily find the corresponding entities in the target language. After substituting two variables X and Y with well-known entities A and B, if the pattern does not contain any wildcard operator "$*$", we input the pattern into the SMT system to get the translation result. We only translate patterns without any wildcard operator because the result of translating a non-complete pattern is not reliable. Moreover, we want to limit the number of patterns to translate to reduce the pre-processing time. Suppose that we have already known that A is translated into A' and B is translated into B' in the target language (we remember the translation of an entity before substituting it). Then we search for the string A' and B' in the translation result. If we can find both of these strings in the translation result, we replace them with the variables X and Y to obtain a translated lexical pattern, otherwise we assume that the translation process failed and we omit the result.

Lexical patterns in the index are usually written by human, not generated by machine translation systems. Consequently, if we can find the translated lexical pattern in the index, then there is a high probability that the SMT system has produced a correct result (which is used by human). Therefore, if we can find the translated pattern in our index, we record that the two patterns are parallel and merge two rows in the matrix $\mathbf{M}$ that correspond to these lexical patterns, as shown in the last row of Table 5.2. From the matrix $\mathbf{M}$ in Table 5.1, after merging both parallel entity pairs and lexical patterns, we obtain a matrix $\mathbf{A}$ as shown in Table 5.2.

5.3 Measuring Relational Similarity across Languages

5.3.1 Multi-lingual Latent Relational Analysis

The dimensions of the row and column vectors of the matrix $\mathbf{A}$ are very large because they are the numbers of different lexical patterns and entity pairs. Moreover, the extraction algorithm in Section 4.2 might extract several noisy co-occurrences between entity pairs and lexical patterns. In addition, many lexical patterns actually have the same meaning but they have different surface forms because there are several ways to state a semantic relation in a natural language (e.g., "X is the CEO of Y" and "X, the CEO of Y"). Therefore, it is difficult to precisely measure the semantic similarity between two entity pairs or two lexical patterns across languages if we directly use column vectors or row vectors of the matrix $\mathbf{A}$. Latent Semantic Analysis [114] and Latent Relational Analysis [29] have successfully used Singular Value Decomposition (SVD) to reduce the number of dimensions of these vectors and to compress semantically similar dimensions into one.

The idea of multilingual term-document indexing has been exploited in multi-lingual Latent Semantic Analysis [81] and cross-language sentiment classification [94]. Therefore, we propose *multi-lingual Latent Relational Analysis*, to measure the similarity between entity pairs and lexical patterns across languages.

For the $m \times n$ matrix $\mathbf{A}$, SVD decomposes $\mathbf{A}$ into three matrices $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{5.1}$$

where $\mathbf{U}$ is an $m \times m$ matrix, $\mathbf{V}$ is an $n \times n$ matrix in column orthonormal form and $\mathbf{\Sigma}$ is a rectangular diagonal $m \times n$ matrix of *singular values* [29, 11]. We can re-arrange the column vectors of $\mathbf{U}$ and $\mathbf{V}$ such that the elements in the main diagonal of $\mathbf{\Sigma}$, which contains singular values, are sorted from large to small (i.e., the top left element has the largest value). The ranks of $\mathbf{A}$ and $\mathbf{\Sigma}$ are equal:

$$\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{\Sigma}) = r \tag{5.2}$$

For example, if $\mathbf{A}$ is the following matrix (rank 3):

$$A = \begin{pmatrix} 0 & \frac{3\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ 0 & -\frac{3\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\sqrt{2} & 0 & 0 \\ \sqrt{2} & 0 & 0 \end{pmatrix}$$

Then we can decompose $\mathbf{A}$ into three matrices $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}$ as follows:

$$\underbrace{\begin{pmatrix} 0 & \frac{3\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ 0 & -\frac{3\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\sqrt{2} & 0 & 0 \\ \sqrt{2} & 0 & 0 \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} \frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} & 0 \\ 0 & \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} \\ 0 & -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} \end{pmatrix}}_{U} \underbrace{\begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}}_{\Sigma} \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}}_{V^T}$$

We can easily check that the columns of $\mathbf{U}$ are orthonormal. Moreover, the columns of $\mathbf{V}$ are also orthonormal.

If $\mathbf{\Sigma}_k$ ($k < r$) is the diagonal matrix created from the top k singular values from $\mathbf{\Sigma}$ and $\mathbf{U}_k$, $\mathbf{V}_k$ are the matrices formed by selecting the first k columns of $\mathbf{U}$ and $\mathbf{V}$, then $\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$ is the matrix of rank k that best approximates the matrix $\mathbf{A}$ (i.e., the Frobenius norm of $(\mathbf{A} - \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T)$ is minimized) [11]. That is, if $\mathbb{M}_k$ is the set of all $m \times n$ matrices with rank k then:

$$||\mathbf{A} - \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T||_F \quad \leq \quad ||\mathbf{A} - \mathbf{B}||_F \qquad \forall \mathbf{B} \in \mathbb{M}_k$$

(in which $||\mathbf{A}||_F$ is the Frobenius norm of the matrix $\mathbf{A}$).
For example, if we take the two largest singular values from the matrix $\mathbf{\Sigma}$ above, then we have the following matrix $\mathbf{A}_2$, which is a rank 2 matrix that best approximates the original matrix $\mathbf{A}$ (rank 3):

$$\mathbf{A}_2 = \underbrace{\begin{pmatrix} \frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & 0 \\ 0 & \frac{\sqrt{2}}{2} \\ 0 & -\frac{\sqrt{2}}{2} \end{pmatrix}}_{U_k} \underbrace{\begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix}}_{\Sigma_k} \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}}_{V_k^T} = \begin{pmatrix} 0 & \frac{3\sqrt{2}}{2} & 0 \\ 0 & -\frac{3\sqrt{2}}{2} & 0 \\ -\sqrt{2} & 0 & 0 \\ \sqrt{2} & 0 & 0 \end{pmatrix}$$

We can directly solve the problem of measuring the relational similarity between two entity pairs (corresponding to two columns in $\mathbf{A}$) by calculating the cosine similarity between two corresponding columns in the low rank matrix $\mathbf{\Sigma}_k \mathbf{V}_k^T$, as in LRA [29]. LRA compresses many semantically similar lexical patterns into one dimension. Therefore, it yields the most precise result in measuring the relational similarity for monolingual case [30]. However, in multi-lingual LRA, the number of common lexical patterns between two entity pairs in two different languages is not large. This implies that we might not achieve a good performance if we use only LRA. Therefore, we propose a novel two-phase lexical pattern clustering algorithm (to be described in the next section) to precisely group semantically similar lexical patterns (across languages) into pattern clusters. The clustering algorithm helps us to transfer a semantic relation of an entity pair across two languages because we can assume that lexical patterns of different languages in the same cluster express the same semantic relation. We then use the result of the clustering algorithm to measure the relational similarity between two entity pairs across languages. We use LRA for calculating the semantic similarity between two lexical patterns (corresponding to two rows in the matrix $\mathbf{A}$). The cosine similarity between two lexical patterns p_i and p_j in the dimensionally reduced space is the cosine of two corresponding rows i and j of the low rank matrix $\mathbf{H}$ defined below [29].

$$\mathbf{H} = \mathbf{U}_k \mathbf{\Sigma}_k \tag{5.3}$$

(because $\mathbf{U}_k$ has the size of $m \times k$, $\mathbf{H}$ is also an $m \times k$ matrix, which represents m lexical patterns in the reduced space of k dimensions, $k < r, n$). We denote $\mathbf{\Phi_{LRA}}(p_i)$ as the transpose of the row vector corresponding to the pattern p_i:

$$\mathbf{\Phi_{LRA}}(p_i) = (\mathbf{H}_{i1}, \mathbf{H}_{i2}, \ldots, \mathbf{H}_{ik})^T \tag{5.4}$$

The cosine similarity between two patterns p_i and p_j in the dimensionally reduced space is then defined as:

$$\mathrm{sim}_{\mathrm{LRA}}(p_i, p_j) = \mathrm{cosine}(\mathbf{\Phi_{LRA}}(p_i), \mathbf{\Phi_{LRA}}(p_j)) \tag{5.5}$$

5.3.2 Lexical pattern clustering algorithm

There might be several paraphrases that describe the same semantic relation in a language. For example, the lexical pattern "X acquired Y" is semantically similar to the pattern "X bought Y", but the surface forms of the patterns are completely different. Likewise, the lexical pattern "*X ga Y wo baishu shita*" (meaning "X acquired Y") in Japanese is also semantically similar to the pattern "X purchased Y" in English. Therefore, the number of parallel patterns that we can find by translation in previous step is often not large enough for retrieving a candidate set for the majority of queries. Moreover, because of this sparseness, the relational similarity between two entity pairs in different languages will be too small that we can not differentiate between two pairs that share only noisy patterns and two pairs that actually hold similar semantic relations. This is the main reason that LRA might not work well in cross-language latent relational search. Previous research [20] on monolingual relational similarity measurement has shown that we can improve the precision while measuring the relational similarity between two entity pairs by clustering semantically similar patterns into clusters and consider all patterns in a cluster as equal. We have used this idea in the previous chapter (Chapter 4) while calculating the similarity between two entity pairs in the same language. Consequently, we propose a two-phase lexical pattern clustering algorithm to 1) capture the semantic

Algorithm 7 Hybrid Lexical Pattern Clustering (HLPC) of lexical patterns

Input: pattern set $\wp$, threshold $\theta_1 > \theta_2 > 0$
Output: cluster set **K**

```
1: K ← {}
2: /* First phase (MLPC, as shown in Algorithm 2) */
3: sort( ℘ )
4: for pattern p ∈ ℘ do
5:    maxClus ← NULL
6:    maxSim ← −1
7:    for each pattern cluster c ∈ K do
8:       cpSim ← sim(p, centroid(c))
9:       if cpSim > maxSim then
10:         maxSim ← cpSim
11:         maxClus ← c
12:      end if
13:   end for
14:   if maxSim ≥ θ1 then
15:      maxClus.append(p)
16:   else
17:      newClus ← {p}
18:      K ← K ∪ {newClus}
19:   end if
20: end for
21: /* Second phase */
22: for each pattern p ∈ ℘ do
23:   if hasParallel(p) then
24:      for cluster c ∈ K do
25:         if sim(p, centroid(c)) ≥ θ2 then
26:            c.append(p)
27:            c.append(paralellOf(p))
28:         end if
29:      end for
30:   end if
31: end for
32: return K
```

similarity between paraphrased lexical patterns in the same language and 2) capture the semantic similarity between similar or translated patterns across languages.

The clustering algorithm is shown in Algorithm 7. In the first phase of our clustering algorithm, we want to capture the semantic similarity between paraphrased lexical patterns in the same language. Therefore, we use the lexical pattern clustering algorithm MLPC in Chapter 4 (which is a version of the clustering algorithm proposed by Bollegala et al. [20, 30]) in this phase. The MLPC (Monolingual Lexical Pattern Clustering) algorithm was shown in Algorithm 2 in the previous chapter, but we rewrite the algorithm in the first phase of Algorithm 7 (lines 3–20) for easy to follow. First, it sorts the pattern set in the order of frequency from high to low to process high frequency patterns first [20]. For each pattern, the algorithm finds the cluster whose centroid has maximum similarity with the pattern (lines 5–13). We try two methods for calculating the similarity between two lexical patterns, sim_{LRA} or sim_{VSM} as defined in Equation 5.5 and Equa-

tion 4.13. When we use sim_{VSM}, we do not need to perform LRA (and hence SVD) so the pre-processing time is fast. We denote the method that uses sim_{VSM} for calculating the similarity as **HLPC** (hybrid lexical pattern clustering) and the method that uses sim_{LRA} as **HLPC+LRA**. If the similarity is above a pattern clustering similarity threshold θ_1 then the pattern is added to the cluster, otherwise, the pattern forms a new singleton cluster itself (lines 14–19). Therefore, this algorithm is a hard clustering algorithm (i.e., each pattern can be in only one cluster). Although there are many clustering algorithms which can be used in the first phase (e.g., the SLINK [106], CLINK [107] algorithms) we choose the lexical patten clustering algorithm by Bollegala et al. [20, 30] because the algorithm has the time complexity of $O(n\log n + n|K|)$, where n is the number of input lexical patterns, and $|K|$ is the number of output clusters. As described in Section 4.3, normally, $|K| \ll n$, therefore, the amortized time complexity of the algorithm is much smaller than $O(n^2)$ or $O(n^3)$, which are required by hierarchical clustering algorithms. This allows us to perform the clustering process in high speed to reduce pre-processing time.

We need to set θ_1 to a high value to reduce the number of large clusters that might express many different semantic relations. However, we observe that when two semantically similar lexical patterns p and q are in the same language, their semantic similarity is normally higher than when they are in two different languages (e.g., p is in Japanese and q is in English). This happens even when the pattern p (in Japanese) has a parallel pattern p' in English. This is because patterns that have parallel partners are associated with a large number of entity pairs in different languages, as the result of multi-lingual entity indexing, which merges entity pairs of two parallel patterns. For example, after merging, the merged pattern $\{p, p'\}$ now co-occurs with both English and Japanese entity pairs, whereas, q co-occurs with only English entity pairs. This implies that the cosine similarity between two row vectors corresponding to these patterns in the matrix $\mathbf{A}$ is low. Therefore, in the second phase, we use a soft clustering algorithm with a lower pattern clustering similarity threshold θ_2 to associate parallel patterns to the pattern clusters that we obtained in the first phase. That is, we consider only patterns that have some parallel partners (e.g., p and p' in the above example) for clustering in the second phase (line 23 of Algorithm 7), and we allow each of these patterns to be associated with many pattern clusters. If the similarity between a pattern that has some parallel partners and the centroid of a pattern cluster is above θ_2, we add the pattern and its parallel partners to the cluster (lines 24–28). We need to associate as many parallel patterns as possible to these clusters to increase the recall as well as the precision for cross-language queries. A soft clustering algorithm in this phase accomplishes this goal, because a pattern and its parallel partners are allowed to appear in multiple clusters.

Moreover, even we set the similarity threshold θ_2 lower than θ_1, it is still large enough to filter out errors in the lexical pattern translation process. For example, if we incorrectly recognize a lexical pattern p' in English as the parallel pattern of a pattern p in Japanese, then the (merged) entity pairs that are associated with p' are completely different from those of p. This makes the similarity between the merged pattern $\{p, p'\}$ and an English pattern q (which is a similar pattern of p') low. Consequently, the merged pattern $\{p, p'\}$ is not grouped into the English pattern cluster that contains q, thereby alleviating errors in the pattern translation phase.

The second phase is an important step in our algorithm because it captures the semantic similarity between patterns across languages. Even when two patterns in two different languages share only a small number of entity pairs so that they failed to be in a cluster in the first phase (because the similarity is much lower than θ_1), they can be grouped into a same cluster in the second phase, because of the low similarity threshold value. Therefore, the second phase is mainly to capture the similarity between paraphrased lexical patterns

Algorithm 8 Retrieving a potential candidate set for a cross-lingual query q

Input: A cross-lingual latent relational search query $q = \{(A, B), (C, ?)\}$
Output: A potential candidate answer set $\Re(q)$

```
1: /* Initialize the potential candidate answer set */
2: ℜ ← {}
3: /* Initialize the potential similar lexical pattern set */
4: 𝔾 ← {}
5: /* Get all lexical patterns that co-occur with (A, B) */
6: P((A, B)) ← GetEntityPair(A, B).patterns()
7: 𝔾 ← 𝔾 ∪ P((A, B))
8: for each pattern p ∈ P((A, B)) do
9:    Ω ← the set of pattern clusters that contain p
10:   /* Add all lexical patterns from these pattern clusters */
11:   for each pattern cluster K ∈ Ω do
12:      𝔾 ← 𝔾 ∪ K
13:   end for
14: end for
15: for each pattern r ∈ 𝔾 do
16:   W(r) ← GetPattern(r).entityPairs()
17:   for each entity pair w ∈ W(r) do
18:      if freq(w) ≥ 5 and w has form of (C, X) then
19:         ℜ ← ℜ ∪ {X}
20:      end if
21:   end for
22: end for
23: return ℜ
```

across languages.

5.4 Retrieving and Ranking Answers

5.4.1 Retrieving a potential candidate set

Given the query $\{(A,\ B),\ (C,\ ?)\}$, we denote the source pair as s ($s = (A, B)$) and a candidate target pair as c ($c = (C, X)$). To process a query, we first identify the language of the source pair and of the entity C. If the source pair s is in a different language with the key entity C, then we will process a cross-language query. Otherwise, we will process a monolingual query. Identifying the language of the source pair and the key entity is not a big problem in our system because we can count the number of Japanese characters in each entity. However, in some European languages, it might be difficult to recognize the language of each entity by this method. In such case, we can find the set of documents that contain the entity in question and determine the language of the entity based on the major language of these documents. Identifying language of a document can be done with high precision [100, 102]. To retrieve a potential candidate answer set for a cross-lingual query, we use the procedure as shown in Algorithm 8. The algorithm is similar to Algorithm 3, but it is adapted to be able to retrieve candidate set when the source pair and the target pair do not share any lexical pattern. First, we add all patterns with which the source pair ($s = (A, B)$) co-occurs (i.e., the set $\mathbf{P}((A, B))$, as defined in Equation 4.8) to a new lexical pattern set $\mathbb{G}$, which represents the set of potentially similar

lexical patterns between the source pair and the target pair (lines 6, 7). We then add all patterns that are in the same pattern cluster with at least one pattern in $\mathbf{P}((A, B))$ to $\mathbb{G}$ (lines 8–14). Adding patterns in the same cluster with a pattern in $\mathbf{P}((A, B))$ is an important step for processing a cross-language query, especially when the source pair co-occurs only with patterns in the source language (the language of the source pair). This is because a pattern $p \in \mathbf{P}((A, B))$ might have some parallel patterns in the target language (the language of the key entity C) or might be in the same cluster with some semantically similar patterns in the target language (in the pattern clustering step, we added all parallel patterns of p into a cluster that contains the pattern p). The rest of the algorithm is identical with that of Algorithm 3 in Chapter 4. For each lexical pattern r in $\mathbb{G}$, we enumerate all entity pairs that appeared with r (i.e., the set $\mathbf{W}(r)$) and append all entity pairs of the form (C, X) into the candidate set $\Re$. By this method we can ensure that each candidate pair $c = (C, X)$ has at least one lexical pattern in the same cluster with some lexical patterns of the source pair $s = (A, B)$. This condition also helps to limit the number of candidate pairs and speed up the candidate retrieving process. At this step, we obtained a potential candidate set $\Re(q)$ for the cross-lingual query q, similar to the set $\Re$ in Equation 4.19 in Chapter 4 for a monolingual query.

5.4.2 Cross-lingual relational similarity measuring

To rank the result list, we must first calculate the relational similarity between two entity pairs s ($s = (A, B)$) and c ($c = (C, X)$). However, we can not use the same algorithm as for monolingual relational similarity measuring (Algorithm 4), because in cross-lingual relational similarity measuring, we must use information regarding parallel lexical patterns. We define the relational similarity $\mathrm{RelSim}(s, c)$ between s and c using a modified version of cosine similarity of their pattern frequency (or PMI) vectors $\mathbf{\Psi}(s)$ and $\mathbf{\Psi}(c)$ (as defined in Equation 4.11) by considering two patterns that are in the same cluster as equal. That is, we allow a value of a dimension in $\mathbf{\Psi}(s)$ to be multiplied with a value of a dimension in $\mathbf{\Psi}(c)$ and added to the inner product if the two dimensions represent lexical patterns of the same pattern cluster. Algorithm 9 shows the procedure to calculate $\mathrm{RelSim(s, c)}$, the relational similarity between the source pair s and a candidate pair c when the source and the target entity pair are in different languages.

The modified inner product of $\mathbf{\Psi}(s)$ and $\mathbf{\Psi}(c)$ is defined as follows: for a pattern $p \in \mathbf{P}(s) \cap \mathbf{P}(c)$, we add $\mathrm{f}(s, p) \cdot \mathrm{f}(c, p)$ (the corresponding dimensions in the vectors $\mathbf{\Psi}(s)$ and $\mathbf{\Psi}(c)$) to the inner product as normal (lines 8, 9). We mark p as a pattern that has been already used for calculating the similarity by adding p to the set of used patterns $\mathbf{T}$ (line 10). Moreover, we add the pairs $<s, p>$ and $<c, p>$ into the semantically similar co-occurrence set cL to be used later in the supporting sentence retrieval phase.

For a pattern $p \in \mathbf{P}(c)$ but $p \notin \mathbf{P}(s)$, we first look for a parallel pattern p' of p, such that $p' \in \mathbf{P}(s)$ and it is not a used pattern (lines 14, 15). We denote the set of such p' as Ω. All patterns in Ω can be considered semantically identical with p, because they are parallel patterns of p in different languages. If we can not find any parallel pattern p' that satisfies the above condition, then we look for a pattern $q \in \mathbf{P}(s) \backslash \mathbf{P}(c)$ that is in the same pattern cluster with p or in the same cluster with a parallel pattern p' of p. We add these patterns into Ω (lines 16, 17). Because q is in the same cluster with p (or one of its parallel partners p'), it is expected to be semantically similar with p. Therefore, Ω now contains lexical patterns that are expected to be semantically similar with p. If there are many patterns q in Ω, we choose the one with the largest co-occurrences with the source pair s, because we want the inner product to be as large as possible (lines 19–26). Finally, we add $\mathrm{f}(s, q) \cdot \mathrm{f}(c, p)$ to the inner product value (line 28). We also mark the chosen q to prevent it from participating to the next pattern comparison steps (we do this by adding q

Algorithm 9 Cross-lingual RelSim(s, c)

Input: two entity pairs s and c in different languages
Output: the relational similarity between s and c
side effect: the set of semantically similar co-occurrences cL is filled

```
1: /* Initialize the inner product to 0 */
2: ρ ← 0
3: /* Initialize the set of used patterns */
4: T ← {}
5: /* Clear the semantically similar co-occurrence set, which is a global variable */
6: cL.clear()
7: for each pattern p ∈ P(c) do
8:   if p ∈ P(s) then
9:     ρ ← ρ + f(s,p)f(c,p)
10:    T ← T ∪ {p}
11:    cL.append(<s,p>) /* for supporting sentence retrieval */
12:    cL.append(<c,p>)
13:  else
14:    ℘ ← {p' | IsParallel(p,p')}
15:    Ω ← (P(s) ∩ ℘)\T
16:    if Ω = {} then
17:      Ω ← ⋃{K ∈ Clusters | K ∩ (℘ ∪ {p}) ≠ {}  }
18:    end if
19:    max ← −1
20:    q ← null
21:    for each pattern p_j ∈ (P(s)\P(c))\T do
22:      if (p_j ∈ Ω) ∧ (f(s,p_j) > max) then
23:        max ← f(s,p_j)
24:        q ← p_j
25:      end if
26:    end for
27:    if max > 0 then
28:      ρ ← ρ + f(s,q)f(c,p)
29:      T ← T ∪ {q}
30:      cL.append(<s,q>)
31:      cL.append(<c,p>)
32:    end if
33:  end if
34: end for
35: return ρ/(|Ψ(s)| · |Ψ(c)|)
```

to the set $\mathbf{T}$ in Algorithm 9, line 29). Moreover, we also add the pairs $<s, q>$ and $<c, p>$ into the semantically similar co-occurrence set cL to be used later in the sentence retrieval phase (line 30, 31).

5.4.3 Entity filtering function

The entity filtering function ($\mathbf{F}_e(q, D)$) for a cross-lingual latent relational search query q is similar to the function for monolingual queries (as shown in Equation 4.21 and Equation 4.22). Therefore, the entity filtering function is a binary function which takes a

query q (from the set of all cross-lingual latent relational search queries $\mathbb{Q}$) and an entity $D \in \Re(q)$ and returns a binary value as follows:

$$\mathbf{F}_e : \mathbb{Q} \times \Re(q) \rightarrow \{0, 1\} \tag{5.6}$$

$$\mathbf{F}_e(q, D) = \mathbf{F}_e(\{(A, B), (C, ?)\}, D) = \begin{cases} 1 & \text{if RelSim}((A, B), (C, D)) \geq \sigma \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

The forms of these functions are identical with those in Equation 4.21 and Equation 4.22. However, note that the method for calculating the potential candidate set $\Re(q)$ is different: it is not calculated by Algorithm 3, but instead is calculated by Algorithm 8. Moreover, the entity filtering similarity threshold σ is also different: it should be smaller than the value of σ for monolingual queries (which was 0.05).

5.4.4 Ranking the candidate list

We define the relevance score $Z(q, D)$ (or $\text{Rel}((A, B), (C, D))$) of an entity D against a cross-lingual latent relational search query $q = \{(A, B), (C, ?)\}$ as identical to the score in monolingual latent relational search (as shown in Equation 4.26). However, note that the relational similarity RelSim is not calculated by Algorithm 4, but instead is calculated by Algorithm 9 (for cross-lingual relational similarity measuring).

We sort the candidate list in descending order of the relevance score to obtain the final result list. Therefore, the algorithm to build a ranked result list for a cross-lingual latent relational search query is identical with that of a monolingual latent relational search query (as shown in Algorithm 5). All the properties of the ranking algorithm for monolingual latent relational search are also correct for cross-lingual latent relational search.

We use the same algorithm as Algorithm 6 to retrieve a supporting sentence set. Note that the semantically similar co-occurrence set mainly contains pairs of type $<s, p>$ and $<c, q'>$ (in which p is a lexical pattern in the source language and q' is a lexical pattern in the target language). Consequently, the retrieved supporting sentences for the entity pair s are mainly in the source language and those for the pair c are mainly in the target language.

5.5 Implementation

We implemented a prototype cross-lingual latent relational search engine by extending the monolingual latent relational search engine in Section 4.9. The components of the system are shown in Figure 5.2. Almost all components of the cross-lingual latent relational search engine are the same as those of the monolingual latent relational search engine, as shown in Figure 4.4. The two modules that are in Figure 5.2 but are not in Figure 4.4 are: the Entity and Pattern Translator and the Cross-lingual Latent Relational Analysis module. The Entity and Pattern Translator is for translating entity pairs and lexical patterns from a source language into a target language. The Translator is an abstraction of underlying machine translation systems that we use in our experiments. It provides a same interface for querying different machine translation systems (we use Google Translate and the open source machine translation software Moses in our experiments). To avoid redundant traffic to machine translation systems, we cache all translation results in our database. Therefore, we keep the translation results in a table in our database. If we could not find a translation of a lexical pattern or an entity in the database, then we issue a query to the underlying translation system to get the result. We then store the translation result into the database, irrespective of the result is a good result or not (i.e.,

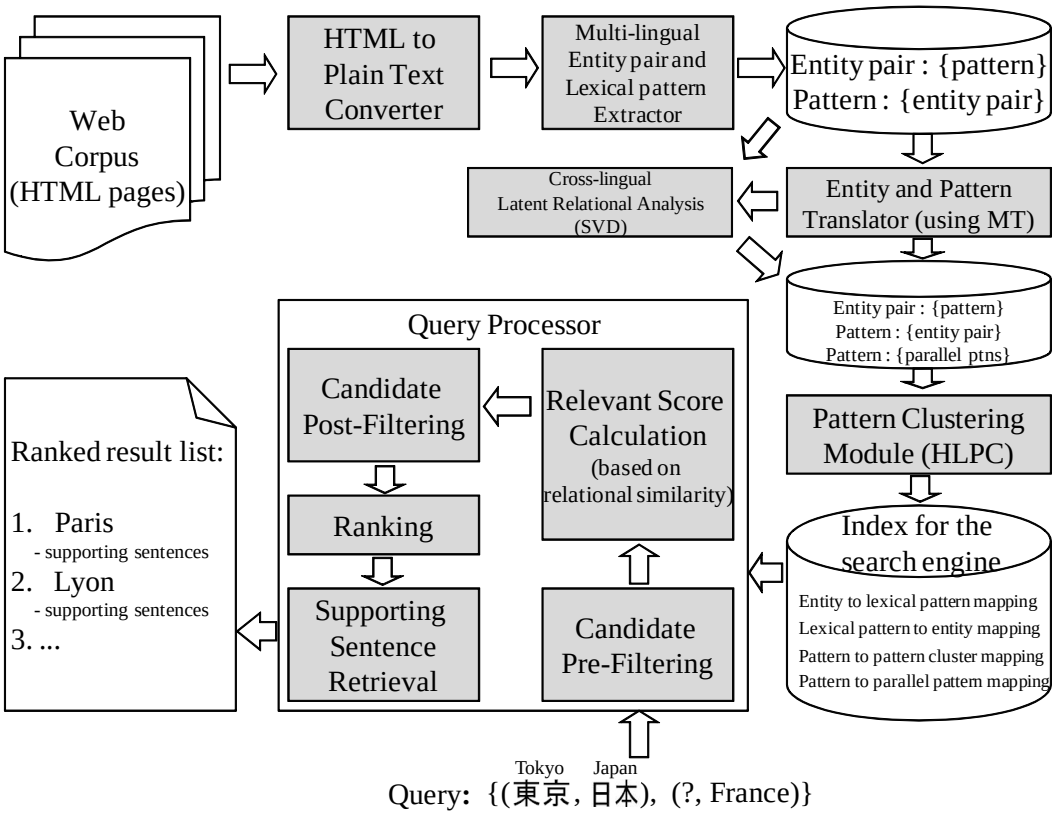


Fig. 5.2. The components of the cross-lingual latent relational search engine

it is passed the verification "Exists in the index?" in Figure 5.1 or not). This is the only additional index (other than those in the monolingual latent relational search engine) that we need to store to process cross-lingual latent relational search queries.

The Cross-lingual Latent Relational Analysis module performs the Singular Value Decomposition (SVD) operation on a large matrix representing the co-occurrences between entity pairs and lexical patterns. We use SVDLIBC[*2] to perform Singular Value Decomposition for Latent Relational Analysis. Specifically, we create an XMLRPC server wrapper for SVDLIBC and we send matrix data from the prototype system (written in Python) to the server and issue commands to instruct the server to decompose the matrix. We then get the decomposition result back from the server to the prototype system.

Other modules in Figure 5.2 are similar to those in the monolingual latent relational search engine as shown in Figure 4.4. However, the Pattern Clustering Module performs the Hybrid Lexical Pattern Clustering (HLPC) algorithm, as shown in Algorithm 7, instead of the Monolingual Lexical Pattern Clustering (MLPC) in Algorithm 2.

Similar to the monolingual prototype system, this cross-lingual latent relational search engine keeps almost all indices (except the extracted sentences and the translation results) in RAM. It is used for learning parameters and for comparing the performance with other implementations of latent relational search.

[*2] http://tedlab.mit.edu/~dr/SVDLIBC/

5.6 Evaluation

In this section, we describe numerous experiments to evaluate the proposed method on cross-lingual latent relational search query sets. We first determine an appropriate value for the clustering similarity thresholds (θ_1 and θ_2) in the Hybrid Lexical Pattern Clustering algorithm (HLPC) by evaluating the proposed system with a corpus for parameter tuning (the *training corpus*, as described in Section 4.10.1). Using another corpus as the test corpus (as described in Section 4.10.1), we compare the performance of the proposed method with three baseline methods: the method based only on LRA [29], the method with only the first phase clustering (in Section 4.3) and the method with the first phase clustering and query, document translation. We then report the best performance that we achieved with the test corpus in Section 5.6.5. We evaluate the average query processing time for cross-lingual query sets to show that the time is not different from the time for processing a monolingual query. We also compare the performance of the proposed method when using two different Statistical Machine Translation systems to show that we can achieve a reasonable performance with different SMT systems in Section 5.6.8.

5.6.1 Relation types and query sets

The proposed method is extensively experimented with eight relation types as shown in Table 4.3 in Chapter 4. These relation types are frequently used in previous research to evaluate relational similarity measuring algorithm [20], monolingual latent relational search engines [8] or relation extraction systems [5, 109].

We use the same training corpus and test corpus with those of the monolingual evaluation experiments to achieve the index for the search engine. As described in Section 4.10.1, the training corpus has size of 1.8 GB, of which about 60% are English web pages and 40% are Japanese web pages. The test dataset has size of 1.6 GB, about one-half of which are English web pages, the rest are Japanese web pages. These sets of web pages contain a large number of entities and relations of many types (not only those in Table 4.3 because a web page might describe many entities and relations and might contain non-related information such as text from advertisements).

We use the same method as described in Section 4.10.1 to create 16 query sets to evaluate the system, eight query sets are English-to-Japanese query sets, the other eight sets are Japanese-to-English. Each query set corresponds to a relation type in Table 4.3 and contains exactly 50 queries. A set of 50 queries is considered to be sufficient to evaluate the performance of an information retrieval system [11]. Each query has only one correct answer. For example, we create the query {(?, YouTube), (*Panasonikku*, *Sanyo*)} for the ACQUISITION relation and {(Ganymede, Jupiter), (*Oberon*, ?)} for the SATELLITE relation (entities that are written in *italic* are actually written in Japanese writing system, we use the English alphabets here for convenience). The criteria for evaluation is the Mean Reciprocal Rank (MRR) of each query set, as explained in Equation 4.33.

For convenience, if we use sim_{VSM} (Equation 4.13) in Algorithm 7 to calculate the similarity between two lexical patterns then we call the method as **HLPC** (hybrid lexical pattern clustering). If we use sim_{LRA} (Equation 5.5) then we call the method as **HLPC+LRA**. We use PMI values as elements of the matrix $\mathbf{A}$ because we found that PMI yields better performance than Frequency (the number of co-occurrences between an entity pair and a lexical pattern) in latent relational search, as described in Section 4.10.3.

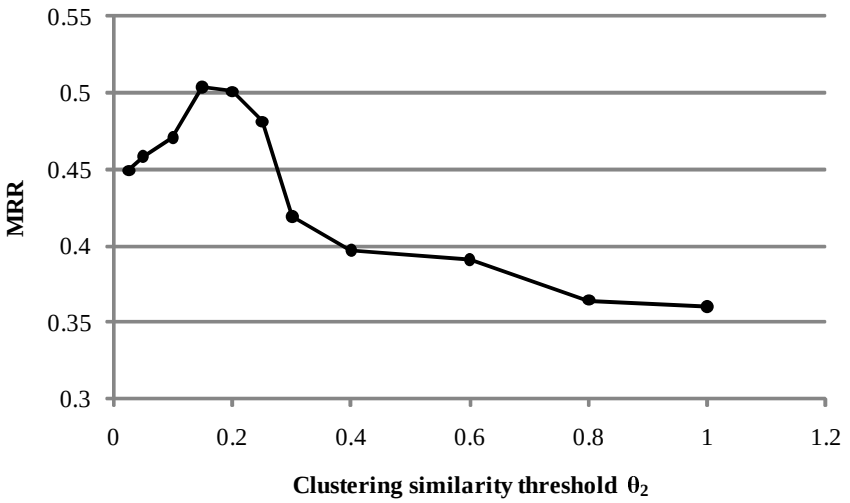


Fig. 5.3. The relation between MRR and θ_2 of the method **HLPC** (at $\theta_1 = 0.4$)

5.6.2 Parameter tuning

We run the proposed extraction algorithm on the training corpus (1.8 GB of Web pages) to build an index for the system. The resulting index contains 5,241,627 lexical patterns and 236,923 entity pairs. Bollegala et al. [6] suggest that we must filter very rare patterns (e.g., patterns that appear only once). This is because rare patterns are normally noisy patterns, which frequently contain strange symbols or misspellings. Therefore, we only use 1,878,463 patterns that appear more than two times to build the matrix **A** for clustering (however, when we calculate the cosine similarity between two entity pairs, we take into account all patterns). Of which, only 149,835 patterns that do not contain the wildcard character ("∗") are considered for translation. We use Google Translate[*3] to translate these lexical patterns from English into Japanese and vice versa.

After the pattern translation process, we found 6812 patterns that have parallel patterns. Therefore, the ratio of reliable translation is only 4.55% and only 0.13% of the total number of patterns are translated. Only 4862 entity pairs (2.05%) are translated (i.e., have parallel entity pairs). These very small ratios indicate that if we had relied only on machine translation, then we would not be able to achieve a reasonable recall level. We use SVDLIBC[*4] to perform Singular Value Decomposition (SVD) of the matrix **A**, as described in Section 5.5. We set the value of k (the number of singular values to be calculated in Section 5.3.1) to 300, as suggested by Dumais [81] and Turney [29]. Because the time complexity of the SVD operation is $O((m+n)k^2)$ ($m \times n$ is the size of the matrix **A**) [115], we can perform the operation in several hours on an Intel Core i7, 3GHz, 24GB RAM machine. The total time complexity of the clustering algorithm is $O(m\log m + m|\mathbf{K}|)$, as described in Section 5.3.2. With this complexity, we were able to perform all pre-processing steps in less than one day on the same machine.

In the Section 4.10.2 on monolingual latent relational search, we perform only the first phase of the clustering algorithm described in Section 5.3.2 to capture the semantic

[*3] http://translate.google.com
[*4] http://tedlab.mit.edu/~dr/SVDLIBC/

similarity between paraphrased lexical patterns in the same language (i.e, no second phase and no LRA). In this setting, we found that the appropriate value for the pattern clustering similarity threshold is 0.4, as shown in Section 4.10.2. Consequently, we set the value of the parameter θ_1 in our clustering algorithm to 0.4 if we use the method **HLPC** (the method that does not require LRA, only sim_{VSM}). We then vary the parameter θ_2 to determine an appropriate value. We use four relation types in the first four rows of Table 4.3 for training purpose. In the test phase, we will use all of the eight relation types in Table 4.3, to avoid the bias to those relations that are optimized in the training phase. Therefore, we use only eight query sets (four English-to-Japanese and four Japanese-to-English query sets) that correspond to the first four relation types in Table 4.3 in this experiment and evaluate the average MRR value of these query sets. Figure 5.3 shows the experiment result. At $\theta_2 = 0.15$, we obtain the best value of MRR. Consequently, in all following experiments, we set θ_2 to 0.15.

For two semantically similar lexical patterns p and q, $\text{sim}_{\text{LRA}}(p, q)$ is often larger than $\text{sim}_{\text{VSM}}(p, q)$ because LRA compresses semantically similar dimensions into one and reduces noisy dimensions. Therefore, we can not assume that the appropriate value for θ_1 in the method **HLPC** (which was set to 0.4) is also appropriate for the method **HLPC+LRA**. Consequently, we vary the value of θ_1 in the method **HLPC+LRA** to find an appropriate value. At $\theta_1 = 0.8$ we achieve the best performance for the method **HLPC+LRA**. This value is much larger than the appropriate value in the method **HLPC** (which was 0.4). Therefore, in all experiments related to the method **HLPC+LRA**, we set θ_1 to 0.8.

5.6.3 Effect of the second-phase clustering algorithm

We investigate the effect of the second phase in the proposed clustering algorithm by comparing the performance of the search engine with and without the second phase clustering (i.e., soft-clustering of parallel patterns). We use the test corpus (1.6 GB of Web pages) to create an index for the search engine in this experiment. We use the **HLPC** method (without LRA) in this experiment because we want to eliminate the effect of LRA to clearly reveal the impact of the second phase in the clustering algorithm. We evaluate the performance of eight query sets corresponding to the first four relation types in Table 4.3. In these eight query sets, four query sets are Japanese-to-English query sets, the rest four query sets are English-to-Japanese cross-lingual query sets.

With the second phase clustering (i.e., the **HLPC** method), we obtain an average MRR of 0.430, while without this phase (i.e., we only rely on pattern translation and the first phase clustering), the MRR is only 0.186. Figure 5.4 shows the comparison between the percentage of queries with correct answer in Top 1, Top 5, Top 10 and Top 20 results when the search engine runs with and without the second phase clustering. Without the second phase clustering, the percentage of queries with correct answer in the Top 1 is only about a half of the percentage when we use the second-phase clustering (16.8%, compared to 31.0%). Moreover, even when we look down the result list to the Top 20, the percentage without second-phase clustering increases very slow. On the other hand, with the second-phase clustering, the percentage increases drastically. This is because without the soft clustering step, only candidate pairs that have at least a parallel pattern with the source pair can be retrieved, therefore the number of candidates is not large. Moreover, because the relational similarity between a cross-language pair could not be precisely measured without the second phase clustering, the percentage of queries with correct answer in the Top 1 is also lower. With the second phase clustering, we can group similar patterns in different languages to precisely measure the relational similarity of a cross-language entity pair. Consequently, we can retrieve many candidates and precisely rank the result list.

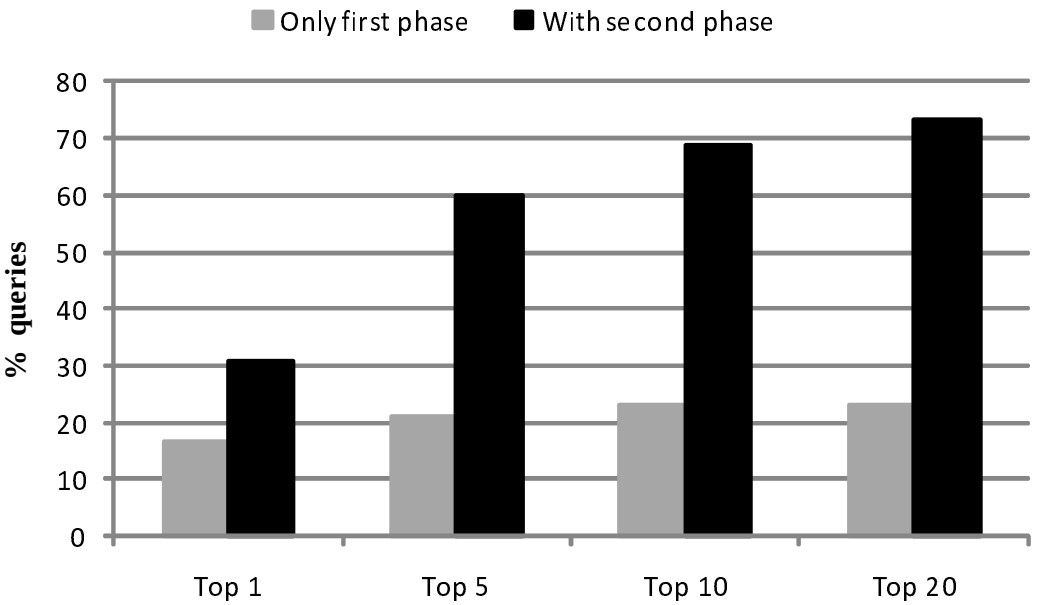


Fig. 5.4. The percentage of queries in which correct answers are in Top N results when the search engine runs with and without the econd-phase clustering

The result shows that the proposed two-phase clustering algorithm successfully captures the semantic similarity between lexical patterns across two languages.

5.6.4 Comparison with baseline methods for Cross-lingual queries

We compare the performance of the two proposed methods (**HLPC** and **HLPC+LRA**) with that of baseline methods using the test corpus (the 1.6GB corpus). Three baseline methods for comparison are as follows.

- **LPC**: This method uses only the first phase of the lexical pattern clustering algorithm (the first phase in Algorithm 7, or the procedure shown in Algorithm 2). This is the method in Chapter 4 for monolingual latent relational search (however, **LPC** finds parallel patterns by lexical pattern translation).
- **Trans+LPC**: This method first translates all documents in the corpus into English. Then it translates all entities in the query into English and performs monolingual latent relational search. The result is translated back to the target language, if needed.
- **LRA**: This method does not use clustering, instead it directly calculates the cosine similarity between two entity pairs using the dimensionally reduced vector space after LRA (i.e., the matrix $\mathbf{\Sigma}_k \mathbf{V}_k^T$).

The comparison is performed on eight query sets (four English-to-Japanese and four Japanese-to-English query sets) similar to those in the previous section. The average MRR of these eight query sets is shown in Figure 5.5. The proposed methods (**HLPC** and **HLPC+LRA**) outperform the **LPC** method by a wide margin. We verify this difference by using a paired t-test in which the samples are 400 queries derived from eight query sets above (each query set contains 50 queries). For a query q, we use the value

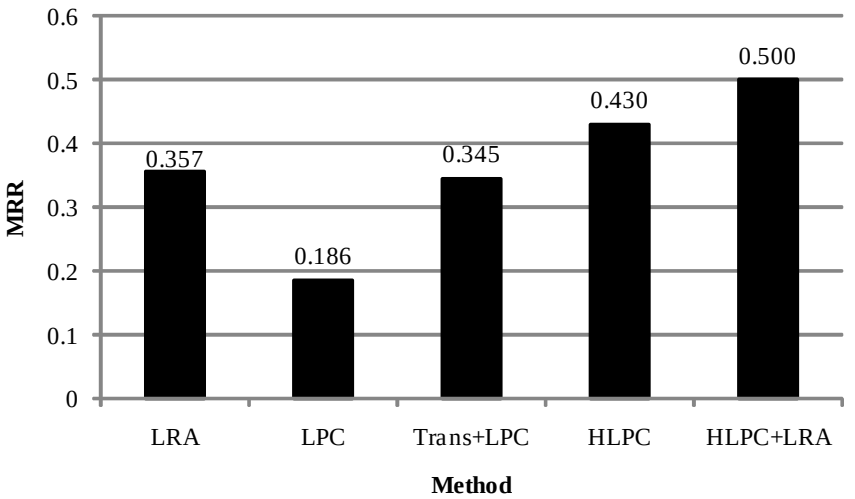


Fig. 5.5. Comparison between the MRR of the proposed methods (**HLPC** and **HLPC+LRA**) and baseline methods on cross-lingual query sets

of $\frac{1}{r_q}$ in the t-test, where r_q is the rank of the first correct answer (therefore, the mean of these values is the MRR of the query set). Because we use the same 400 queries to evaluate the five methods, we can use a paired t-test in this case. A set size of 400 is large enough for a paired t-test to verify the difference between two means of two sample sets. We found that the difference between the performance of the method **LPC** and **HLPC** is statistically significant (at the significance level 0.01) under the paired t-test. This proves that the proposed second-phase clustering (i.e., soft clustering of parallel patterns) successfully captures the semantic similarity between paraphrased lexical pattern across languages.

We also found that the difference between the performance of **Trans+LPC** and **LRA** is not statistically significant under the paired t-test (the value of the t-Statistic is 0.466, corresponding to an one-tail p-value of 0.32, which is much larger than the significance level of 0.05). This indicates that document translation combined with the clustering algorithm proposed by Bollegala et al. [30] can achieve a comparable performance to multi-lingual LRA [29]. The values of MRR of the two proposed methods, **HLPC** and **HLPC+LRA** are 0.430 and 0.500, respectively. Under the same paired t-test settings, **HLPC** and **HLPC+LRA** significantly outperform **LRA** (the p-value for the **HLPC** vs. **LRA** test is 0.003, whereas, the p-value for the **HLPC+LRA** vs. **LRA** test is 2.5×10^{-8}, which indicates that the differences are statistically significant at the significance level $\alpha = 0.01$). Moreover, **HLPC** and **HLPC+LRA** significantly outperform **Trans+LPC** (at the significance level of 0.01). Finally, **HLPC+LRA** significantly outperforms **HLPC** (the one-tail p-value is 0.0001). This demonstrates that LRA significantly improves the performance of the system (with the cost of the SVD operation on a large matrix).

5.6.5 Performance on each query set

We use the test corpus (the 1.6GB corpus) to evaluate the performance of the system on 16 query sets (of eight relation types as describe in Table 4.3; eight of them are English-to-Japanese query sets, the rest are Japanese-to-English). The evaluation result is shown in Figure 5.6. We achieve high MRR on the CAPITAL, PRIMEMINISTER

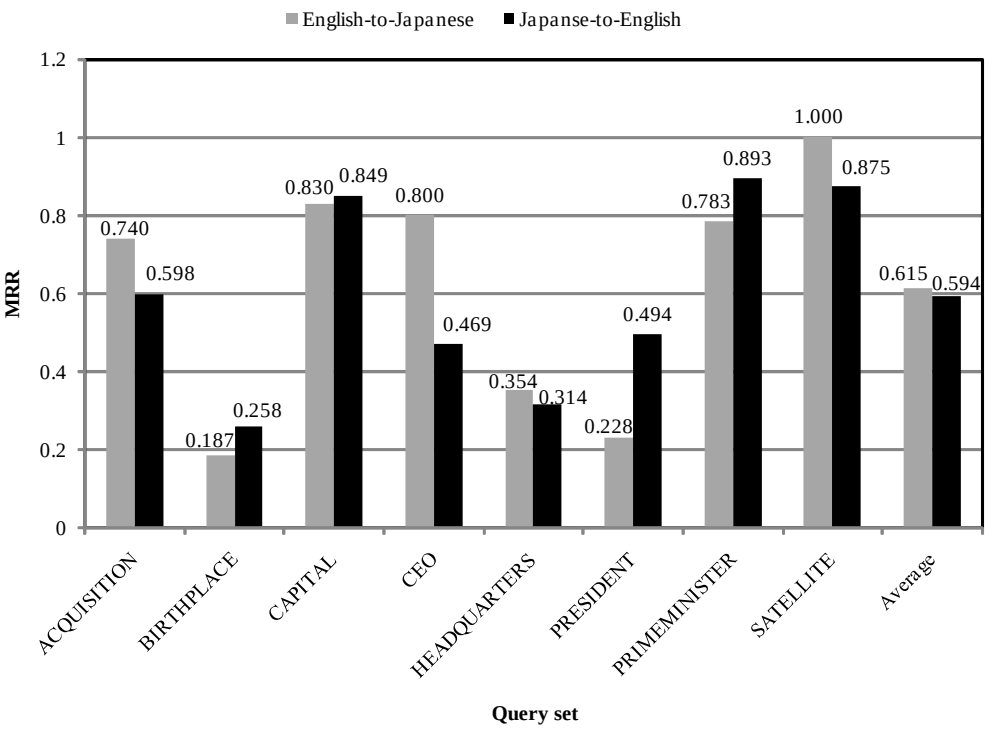


Fig. 5.6. Performance of the proposed method (**HLPC+LRA**) on cross-language latent relational search queries of eight relation types

and SATELLITE relation. This is because lexical patterns that represent the CAPITAL, PRIMEMINISTER and SATELLITE relation are simple for translation from English into Japanese and vice versa. Especially the MRR for the SATELLITE relation is very high because the number of entities in this relation is not large and all entities in this relation are very popular. Therefore, for globally mentioned relations (i.e., relations that are popular across many countries and languages such as the SATELLITE relation), the proposed method achieves high performance. For the CEO relation, although the entities might not be popular but in Japanese sometime the phrase for describing the CEO relation is "CEO", which is identical with the phrase in English. Therefore, we can achieve high performance on the CEO relation in English-to-Japanese query set because we can easily retrieve candidates which have the common lexical pattern "CEO". However the MRR of the Japanese-to-English CEO query set is not high because there are many paraphrases to describe the CEO relation in Japanese (such as "*X ga Y no daihyo torishimari yaku*", "*X ga Y no shachou*", ...). Moreover, the relation between a CEO and a company might be a local information (i.e., only mentioned in a specific language), if the company is not famous. Similarly, the BIRTHPLACE and HEADQUARTERS relations have many different lexical patterns in Japanese and it is very difficult to exactly translate these patterns into English. More importantly, these relations might be local information and it is difficult to recognize the relational similarity between these relations across languages. Therefore, the performance on these query sets is low.

On average, we achieve an MRR of 0.605 for 16 query sets of eight relation types, as described above. Figure 5.7 shows some example queries and results that the search engine retrieved.

Query	Answer	Excerpt of the supporting sentence list	Lexical patterns
{(Charlie Chaplin, London), (Matsushita Konosuke 松下幸之助, ?)}	Wakayama 和歌山	- On April 15, 1889, Charlie Chaplin *was born in* London, England to Charles Chaplin, Sr., and Hannah Hill. - 明治 27 年 11 月 27 日、松下幸之助は和歌山*に生まれた*。 (trans: Konosuke Matsushita was born in Wakayama on Nov. 27, Meiji 27th year)	X wa born in Y X は Y に生まれ X は Y 出身 X * born in Y on
Rakuten Infoseek {(楽天,インフォシーク), (?, Powerset)}	Microsoft	-Confirmed: Microsoft *buys* Powerset natural-language search. -楽天がインフォシーク*を買収*したことによって... (trans: By acquiring Infoseek, Rakuten, ...) - By *acquiring* Powerset, Microsoft may launch a competitive ... - インフォシーク*買収*が楽天にとって吉と出るか凶と出るか。(trans: Is it good or bad for Rakuten to acquire Infoseek?)	X acquir Y X が Y を買収 X to buy Y X * Y の買収

Fig. 5.7. Example queries and results of the proposed search engine (the small string above a Japanese entity is the transliteration of the entity).

Table. 5.3. Average cross-lingual latent relational search query processing time for each query set (the standard deviation is the value for a sample of 50 queries)

Query set	English-to-Japanese time (s)	Japanese-to-English time (s)	Average (s)
BIRTHPLACE	0.03 ± 0.02	0.20 ± 0.15	0.12 ± 0.14
HEADQUARTERS	0.03 ± 0.02	0.66 ± 2.30	0.35 ± 1.65
CEO	0.06 ± 0.02	1.11 ± 2.37	0.59 ± 1.75
ACQUISITION	0.10 ± 0.07	0.70 ± 1.85	0.40 ± 0.33
PRESIDENT	0.16 ± 0.11	0.70 ± 2.16	0.43 ± 1.55
PRIMEMINISTER	0.26 ± 0.09	1.58 ± 3.23	0.92 ± 2.37
CAPITAL	0.43 ± 0.16	3.21 ± 4.35	1.82 ± 3.37
SATELLITE	0.16 ± 0.06	0.70 ± 0.20	0.43 ± 0.31
Average	0.15 ± 0.15	1.11 ± 2.51	0.63 ± 1.84

5.6.6 Query processing time

In this section, we evaluate the average query processing time of 16 query sets in Section 5.6.5. Each query set in Section 5.6.5 has 50 queries. Therefore, we first calculate the average query processing time and the standard deviation for each query set. We then report the average query processing time for all query sets. The experiment procedure is similar to that in Figure 4.14, except that the number of queries in each query set is 50. Table 5.3 shows the average query processing time and the standard deviation for each query set. The average query processing time of the Japanese-to-English CAPITAL, PRIMEMINISTER and CEO cross-lingual query set is significantly higher than that of the corresponding monolingual query set. Especially, the CAPITAL query set takes significantly higher time than other query sets. The entity pairs in the CAPITAL query set might participate in numerous relations other than the capital relation (for example, the entity pair (Tokyo, Japan) also holds the relation "X is the largest city in Y". Consequently, there are numerous lexical patterns that co-occur with entity pairs in this query set. This leads to a large query processing time. For other query sets, such as the CEO or the PRIMEMINISTER query set, there are more lexical patterns in Japanese than in English. For example the CEO relation in Japanese might be stated by Japanese lexical patterns such as "*X ha Y no shacho*", "*X ha Y no saiko sekinin sha*" or "*X ha Y no CEO*". Where as, in English, the relation is often described using the pattern "X is the CEO of Y". The PRIMEMINISTER relation in Japanese is also stated by several different

Table. 5.4. Comparison between the proposed methods and existing methods (TopN is the percentage of queries with correct answer in the Top N ranked results)

Method	MRR	Top1	Top5	Top10	Top20
Kato et al. [8](JJ)	0.545	43.3	68.3	72.3	76.0
Monolingual (EE)	0.967	94.1	99.8	99.8	99.9
Monolingual (JJ)	0.888	87.5	90.0	90.0	90.0
HLPC+LRA-EE	0.971	94.9	99.9	100	100
HLPC+LRA-JJ	0.889	87.0	91.0	91.0	91.0
HLPC-Cross	0.515	37.6	70.1	78.4	82.9
HLPC+LRA-Cross (Freq)	0.579	44.6	74.6	83.6	88.4
HLPC+LRA-Cross (PMI)	0.605	49.8	74.5	78.5	82.0

lexical patterns such as "*X ha Y no shori daijin*" or "*X ha Y no shusho*" .... This is also the reason why the average query processing time of Japanese-to-English query sets is longer than that of English-to-Japanese query sets. There are some query sets have large standard deviation, as can also be seen in the experiment for monolingual queries. This is because there are a very small number of queries in which the system takes a long time to process (e.g., 16 seconds for a query in the CAPITAL query set). However, the number of such queries is very small (e.g., one or two queries in each query set). Overall, the query processing time in cross-lingual latent relational search is not larger than that of monolingual latent relational search. It is worth noting that, on every query set, the average query processing time is less than ten seconds, which is acceptable for normal user search sessions.

5.6.7 Comparison with existing monolingual latent relational search methods

In Section 4.10.6, we compared the performance of the proposed method for monolingual latent relational search with that of an existing monolingual latent relational search engine [8]. Note that the method that we proposed for cross-lingual latent relational search can also be used for processing monolingual latent relational search queries. Therefore, in this section, we further compare the performance of the proposed method with eight Japanese-to-Japanese and eight English-to-English (monolingual) query sets corresponding to eight relation types in Table 4.3. The comparison result is shown in Table 5.4. The first row in the table shows the result reported in [8] on Japanese monolingual query sets (of many common relation types in Table 4.3). The second and third rows are the performance of the monolingual method on English and Japanese monolingual query sets, as shown in Chapter 4. The fourth and fifth rows are the performance of the **HLPC+LRA** method on the same monolingual query sets. The last three rows are the performance of the proposed methods on 16 cross-language query sets which are described in previous sections. The results of **HLPC-Cross** and **HLPC+LRA-Cross (Freq)** are results when we are using frequencies as feature vector values. The **HLPC+LRA-Cross (PMI)** row shows the performance of the proposed method in this paper, which uses PMI as feature vector values. We found that PMI yields better performance than Frequency (the number of co-occurrences between an entity pair and a lexical pattern) in latent relational search. The performance of the **HLPC+LRA** method on *cross-language* query sets is slightly higher than that of the method in [8] on *monolingual* query sets. The gap between **HLPC+LRA-EE** and **HLPC+LRA-Cross** can be explained by the gap between the difficulty of monolingual latent relational search and cross-language latent relational search.

5.6.8 Dependency on the underlying machine translation system

In this section, we study the effect of the Statistical Machine Translation (SMT) system used for translating lexical patterns and entity pairs on the performance of the proposed method. Specifically, we compare the performance of the method when using two different SMT systems. The first SMT system is Google Translate[*5], which we used in all previous experiments concerning cross-language queries. The second SMT system is the open source SMT system Moses [113], which is frequently used as a baseline for evaluating machine translation techniques. Moses does not provide default parallel corpus for Japanese-to-English translation. Therefore, we use the Japanese-English bilingual corpus in the Kyoto Free Translation Task (KFTT) [116] as training data to train the Moses system. This corpus is an aligned parallel corpus that contains Wikipedia articles related to Kyoto[*6]. The corpus includes about 440,000 parallel sentences which are made up of 12 million Japanese words and 11.5 million English words[*7]. With this very small amount of training data, the SMT system trained with this corpus only achieves a reasonable performance if the input sentence is related to cities or locations in Japan, especially Kyoto. We run all pre-processing phases to build an index for our search engine, as in previous experiments, except that the SMT system is now changed from Google Translate to the Moses system training with KFTT. Because we trained Moses for Japanese-to-English translation (i.e., we input Japanese lexical patterns and get English outputs), we only evaluate performance of eight Japanese-to-English query sets in the previous sections. Figure 5.8 shows the comparison between the performance of the search engine on these eight cross-lingual (Japanese-to-English) query sets when the search engine uses Google Translate and Moses as the underlying SMT system. From the figure, we can observe that the performance of the system using Moses on the CAPITAL query set is relatively good, despite the fact that the training data for the SMT system is very small. This is because the CAPITAL relation is frequently mentioned in the KFTT bilingual corpus (Kyoto is the former capital of Japan). We do not achieve a comparable performance on other query sets because the KFTT corpus contain little articles concerning these relations, so the precision of Moses on the task of lexical pattern translation is low, compared to Google. This result implies that, with enough training data (bilingual corpus) to train the underlying SMT system, we can achieve a reasonable performance.

[*5] http://translate.google.com/

[*6] The data is originally prepared by the National Institute for Information and Communication Technology of Japan (NICT)

[*7] http://www.phontron.com/kftt/

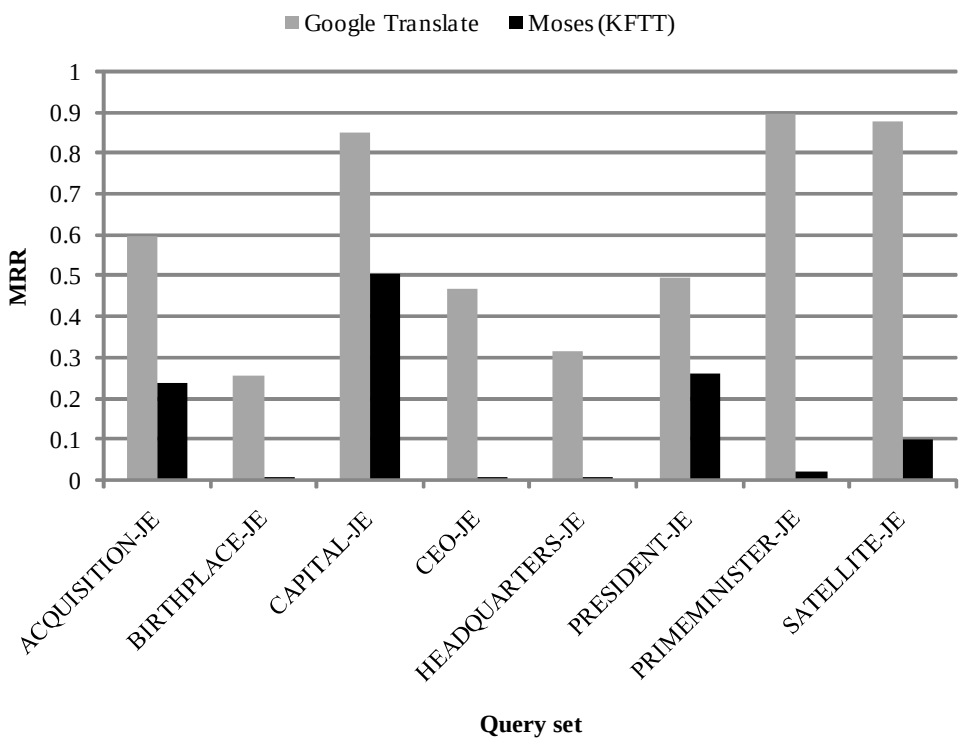


Fig. 5.8. Comparison between the performance of the search engine on Japanese-to-English cross-language query sets while using Google Translate and Moses as the underlying SMT system.

Chapter 6

Milresh: A Large-Scale Latent Relational Search Engine Based on the Proposed Model

In this chapter, we present Milresh, a large scale latent relational search engine that is based on the proposed model. While the prototype systems (described in Section 4.9 of Chapter 4 and Section 5.5 of Chapter 5) are mainly for experimental purposes, such as for parameter tuning and performance evaluation, the Milresh system is a practical latent relational search engine. We design and implement Milresh to be able to index large corpora, such as an entire Wikipedia data dump or the ClueWeb09[*1] corpus. In particular, Milresh is built on top of recent technologies in data intensive parallel processing and distributed storage systems, which allow storing and retrieving billions of lexical patterns and entity pairs in high speed.

We first present the software architecture of Milresh, which we design for scaling to large corpora and large volume of queries. We then describe in detail the data schema and the function of each module in the system. We demonstrate that the proposed system could be used for answering sophisticated questions in the INEX 2008 Entity Ranking task [117]. Specifically, we use the entire English and Japanese Wikipedia as corpora (which totally include about seven million Wikipedia articles) for building an index for the search engine. We then use the queries and standard answers in the INEX 2008 Entity Ranking task to evaluate the inferred average precision (xinfAP) of the system. Finally, we compare the performance of Milresh with other entity ranking methods in Section 6.8.

6.1 System Architecture

Figure 6.1 shows the architecture of Milresh. We design Milresh as a database centric application, in which almost all components are relied on the database. This reflects the fact that for a search engine, the index is one of the most important components, and we store the index in the database. Moreover, we employ a Component-Based architecture, which allows each component to be independent from other components. The only constraint on components is that, each component must provide a set of pre-specified APIs. Once a component provides the required APIs, it can be varied in implementation and even be replaced during runtime.

Because we want to index large corpora in Milresh, we use a key-value store as the database. Different from relational database management systems (RDBMS), a key-value

[*1] http://lemurproject.org/clueweb09.php/

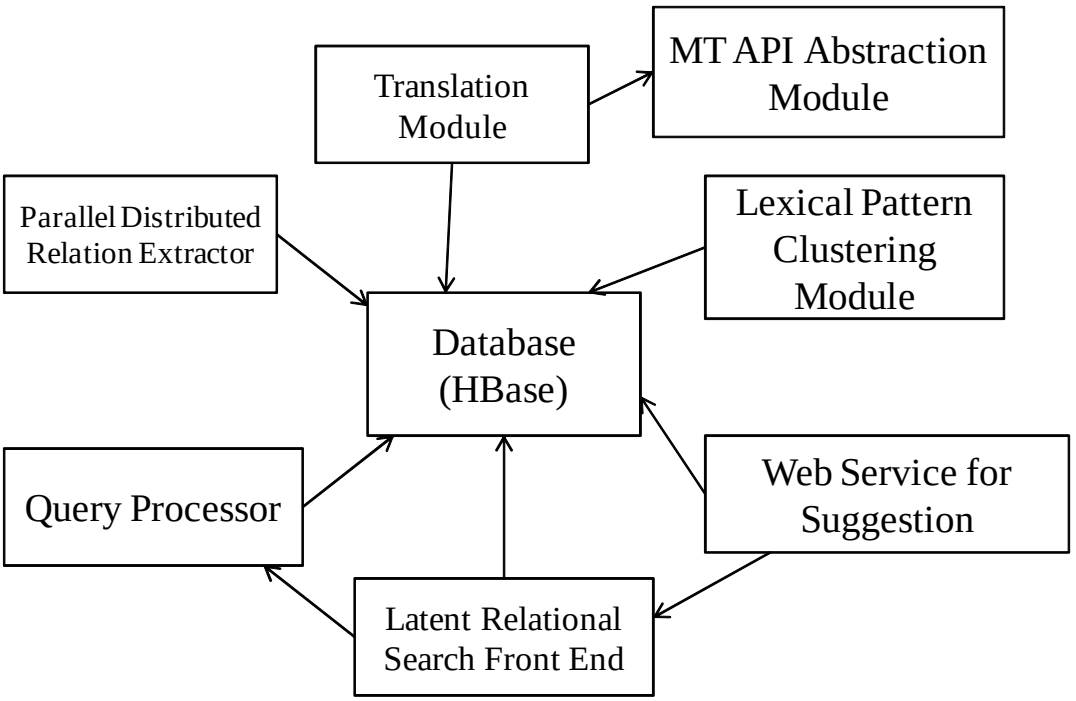


Fig. 6.1. The architecture of the Milresh system

store does not manage any data integrity constraints, such as the primary key constraint or the foreign key constraint. Instead, a key-value store system only allows the description of simple mapping from keys to values, similar to a hash table in memory. The data integrity constraints in a key-value store are mainly maintained by the user of the database. Although this might lead to substantial work for the user, it allows the database management system to easily employ data intensive distributed parallel processing techniques, such as load balancing, data shedding ... Therefore, it is appropriate to store a large amount of entity pairs and lexical patterns for latent relational search.

The "Parallel Distributed Relation Extractor" in Figure 6.1 implements the relation extraction algorithm in Section 4.2. It is a distributed parallel version of the Extractor in previous chapters. The Parallel Distributed Relation Extractor is based on the MapReduce programming model [118], which is a distributed parallel programming framework for data intensive applications.

We allow the system to use several types of machine translation APIs (such as Google Translate API, Moses XMLRPC API, ...). Therefore, we create an abstraction layer for these APIs, and we call this module as the "MT API Abstraction Module".

To provide suggestion when a user partially input a keyword, we implement the "Web Service for Suggestion". This web service looks up the database for all entities that match the partially input keyword and outputs a list of suggested keywords to the front end. The "Latent Relational Search Front End" provides the user interface for Milresh. It uses the Query Processor to process a query and displays the answer list. It invokes the Web Service for Suggestion to get suggestion lists to render to the user.

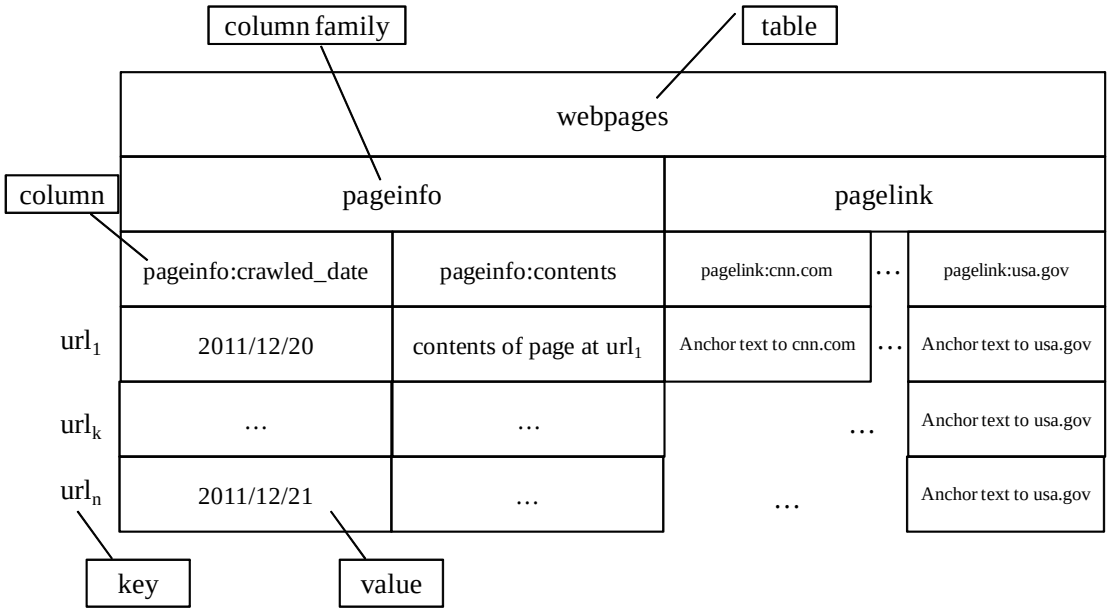


Fig. 6.2. An example of table in HBase

6.2 Database Schema

In Milresh, we use HBase, an open source implementation of Google BigTable [119], as the key-value store. HBase is a non-relational distributed storage system modeling after BigTable. It allows us to store the mapping from a key to column families, as shown in Figure 6.2. Each column family forms a basic unit of access control in the BigTable model and might contain a large number of columns. Each column in turn can store a single data value, such as an integer (four bytes), a long integer (eight bytes) or a string. For example, the table in Figure 6.2 is a table for storing crawled Web pages. Each Web page is stored in a row (specified by its URL). The table has two column families, namely, the "pageinfo" and "pagelink" family. The *pageinfo* column family includes information regarding the page contents and crawled date. The *pagelink* column family provides information regrading the set of out link from the page. The anchor text of each out link is stored in a cell in the table whose row representing the page and whose column is "pagelink:target_url", in which "target_url" is the target of the out link. There might be a very large number of out links from a Web page to other pages. Consequently, the number of columns in the *pagelink* family might also be very large.

The database schema for Milresh is briefly described in Table 6.1. The schema is designed following the concept of separation of concerns. The Parallel Distributed Relation Extractor works with tables *rs_counters*, *rs_documents*, *rs_entities*, *rs_entitypairs*, *rs_patterns* and *rs_sents*. The Translation Module reads data from the table *rs_patterns*, *rs_entities* (for translating entities) and writes to *rs_parallelpatterns*, *rs_entities*. The Lexical Pattern Clustering Module reads data from *rs_patterns* and stores the results into *rs_pattern_clusters*. The Suggestion Web Service reads data from the table *rs_entities*. Finally, the Query Processor uses data from *rs_entitypairs*, *rs_patterns* and *rs_pattern_clusters*.

Figure 6.3 shows the HBase table for storing information of entities in Milresh. The "partner" column family represents entities which make pair with the entity in the row

Table. 6.1. Database schema for Milresh

Table Name	Contents	Example Column Families, Columns
rs_counters	Counters, frequency statistics	statistics, statistics:total, ...
rs_documents	An indexed document (URL, contents ...)	docinfo, docinfo:url
rs_entities	Extracted entities	partner, partner:totalfreq
rs_entitypairs	Index from entity pairs to lexical patterns	freq, sents, freq:ptn1
rs_patterns	Inverted index from lexical patterns to entity pairs	entitypairs, ptninfo
rs_pattern_clusters	Pattern clusters	epvector, ptncluster
rs_parallelpatterns	Parallel patterns (from MT systems)	ptninfo, ptninfo:trans_result
rs_sents	Extracted sentences	sentinfo, sentinfo:contents, ...

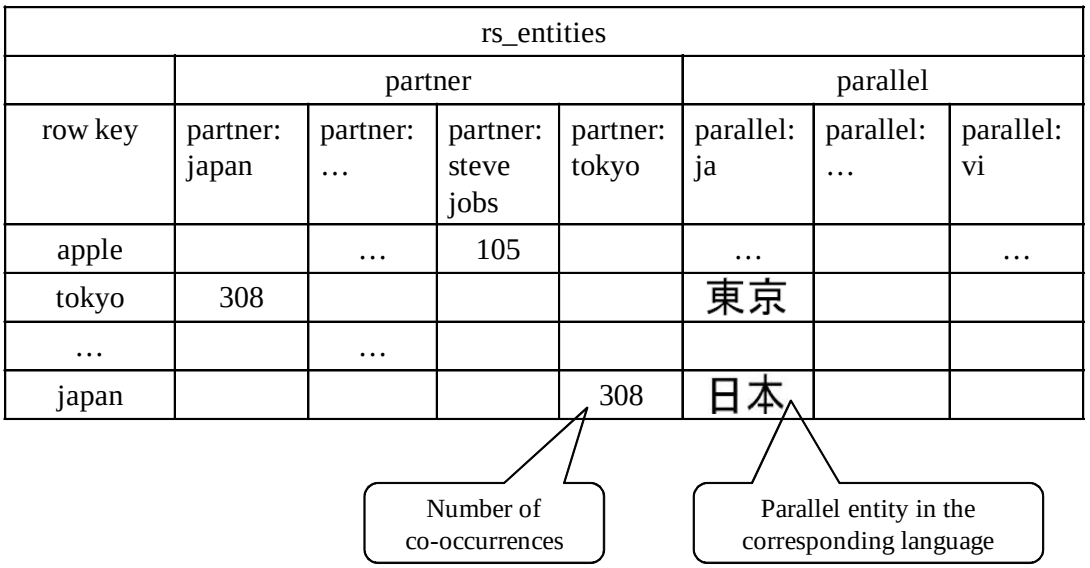


Fig. 6.3. The HBase table for storing entities in Milresh

key of the table. A column of HBase is formed by appending a column "qualifier" to the column family, such as in "partner:tokyo", "tokyo" is a column qualifier. In the "partner" column family, each column qualifier represents an entity, which makes pair with the entity in the row key. The value of the corresponding cell is the frequency of the entity pair in the analyzed corpus. The "parallel" column family stores information concerning parallel entities (i.e., the translation or transliteration of the entity in the row key). For a specific entity, we can store an arbitrary number of different parallel entities in different languages, as shown in Figure 6.3. This makes the proposed model suitable for processing latent relational search queries in multiple languages (i.e., more than two languages).

Figure 6.4 and Figure 6.5 present the most important tables in the index: the entity pair table and the lexical pattern table, respectively. In the table *rs_entitypairs*, there are two column families: the "freq" and the "sent" family. The "freq" column family represents

rs_entitypairs							
	freq				sent		
row key	freq:X is the capital of Y	freq:X is the CEO of Y	freq: …	freq:X is headquart-ered in Y	sent:X is the capital of Y	…	sent:X is headquart-ered in Y
apple, cupertino			…	108			10306
ballmer, microsoft		267	…				
…			…				
tokyo, japan	503		…		208374		

Number of co-occurrences

Sentence ID of the co-occurrence

Fig. 6.4. The HBase table for storing entity pairs in Milresh

rs_patterns							
	entitypair				ptninfo		
row key	entitypair: apple, cupertino	entitypair: ballmer, microsoft	entitypair: …	entitypair: tokyo, japan	ptninfo: cluster_ids	…	ptninfo: parallel_ja
X is the capital of Y			…	503	5034, 7092		…
X is the CEO of Y		267	…				
…			…				
X is headquartered in Y	108		…		2035, 9630, …		XはYに本社を置く

Number of co-occurrences

Cluster IDs of the pattern clusters that contain this pattern

Parallel pattern

Fig. 6.5. The HBase table for storing lexical patterns in Milresh. The left half of the table is the transpose the left half of the previous table: the column qualifiers in the previous figure become the row keys in this figure.

the numbers of co-occurrences between an entity pair (in the corresponding row key) and the lexical patterns that the entity pair co-occurs with. Each column qualifier in this family is a lexical pattern and the value of a cell is the number of co-occurrence between the corresponding row key (entity pair) and the column qualifier (lexical pattern). The column family "sent" represents the "positions" of these co-occurrences in the corpus. The "position" here is the sentence in which an entity pair in a row key co-occurs with a lexical pattern in a column qualifier. For example, in Figure 6.4, the entity pair "tokyo, japan" co-occurs with the lexical pattern "X is the capital of Y" in the sentence numbered 208374. Because updating a cell in HBase takes much more time than just overwriting the cell, we always overwrite a cell in the "sent" column family, instead of appending the sentence into the value of the cell. Therefore, if an entity pair and a lexical pattern co-occur in two different sentences in the corpus, then only one sentence is stored (the omitted sentences often have similar meaning with the stored sentence because they contain the same entity pairs and lexical patterns). However, we do not utilize this technique for the number of co-occurrences in the "freq" column family because it causes underestimation of the number of co-occurrences. Therefore, we must update the number of co-occurrences if a cell in the "freq" column family already has a value. Note that if an entity pair and a lexical pattern do not co-occur with each other then we will not store the cell in the table (i.e., if a cell has the value of zero then it is not actually stored, instead, it does not present in the table).

In the HBase table *rs_patterns* in Figure 6.5, each row key represents a lexical pattern. The column family "entitypair" represents the entity pairs that a lexical pattern co-occurs with. We can infer this information from the column family "freq" in the *rs_entitypairs* table because these column families are actually the transpose of each other. However, because accessing a row is much faster than accessing a column for every row, we must store this index to reduce the query processing time. The column family "ptninfo" stores information regarding the cluster ID of the pattern (which the pattern clustering algorithm assigns), and the parallel patterns in different languages. Although in the experiment, we only evaluate the system with English and Japanese, the proposed model can be applied for more than two languages. Therefore, the number of different languages in the "ptninfo" family might be more than two.

6.3 Parallel Distributed Relation Extraction

To be able to pre-process large corpora in high speed, we create a parallel distributed relation extractor based on the MapReduce programming model [118]. Specifically, we use Apache Hadoop, an open source implementation of the MapReduce framework to create the Parallel Distributed Relation Extractor. At the map phase, we input each document to the mapper and emit various information concerning extracted entities, entity pairs and lexical patterns, as shown in Figure 6.6. We also emit lexical patterns that are good for pattern translation (i.e., patterns that contain both X and Y and do not contain wildcards) at the map phase. Each lexical pattern for translation is emitted together with its Named Entity tags (NE tags, for convenience, we also call these tags as POS tags) for the variables X and Y in the pattern (e.g., LOCATION or PERSON).

At the reduce phase, information concerning the frequencies will be aggregated (becoming total frequencies, *tFreq*) and stored into HBase.

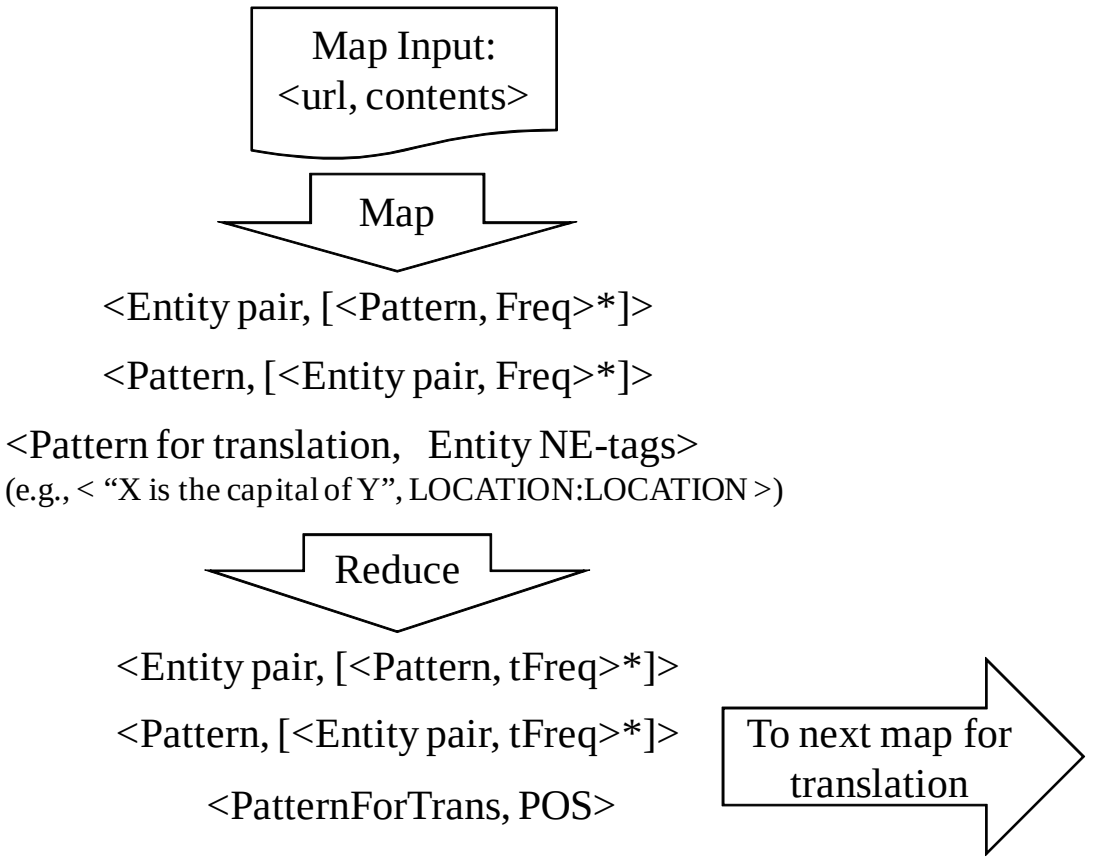


Fig. 6.6. The Map and Reduce phases in the Relation Extractor

6.4 Lexical Pattern and Entity Translation

We create the Translation Module for translating lexical patterns and entity pairs. This module invokes the underlying Machine Translation system using the "MT API Abstraction Module", which provides the abstract interface for several different machine translation systems, such as Google Translate API or the Moses API.

We invoke these machine translation systems via remote procedure calls (through XML-RPC or Restful Web Service APIs). We perform translation in parallel using the MapReduce framework, as shown in Figure 6.7. Specifically, the translation step can be executed right after the relation extraction process. After the relation extraction process, we obtain information regrading entity pairs, lexical patterns and the frequencies of those objects from the corpus. Therefore, in this step, we can utilize this information for further filtering out inappropriate lexical patterns before input them into the machine translation system (e.g., filtering out rare patterns).

The pattern and entity translation process is also a MapReduce application and is executed in parallel. Therefore, the process can be performed very fast, provided that the underlying machine translation system supports a large volume of queries per second. However, sending a large traffic to a machine translation system causes a huge cost (e.g., Google Translate API charges US$20 for 1 million characters of translation input, whereas, we might extract billions of lexical patterns, each containing tens of characters). Therefore, in the experiments in following sections, we could not evaluate the system by translating

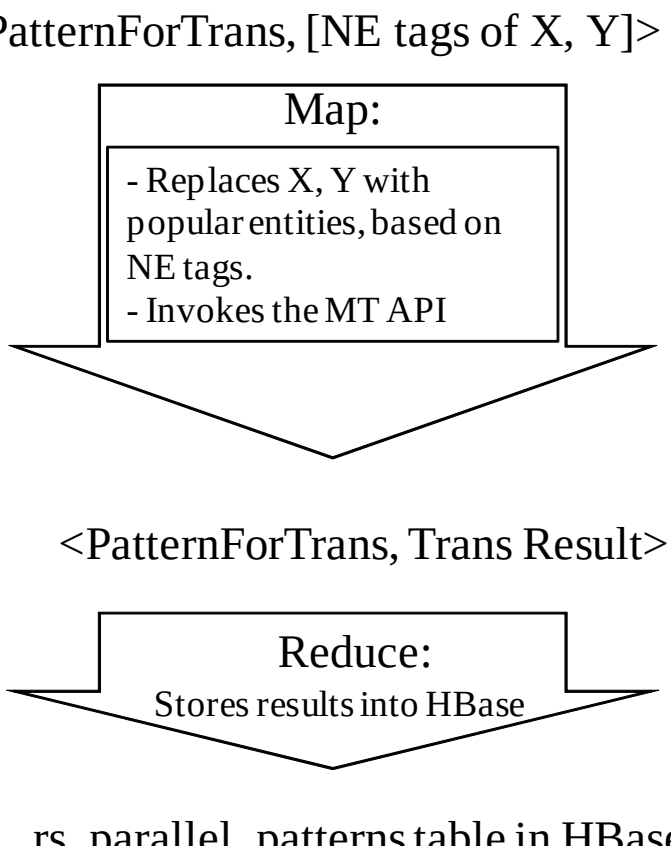


Fig. 6.7. The Map and Reduce phases for translation, after pattern extraction

all appropriate lexical patterns.

6.5 The Pattern Clustering Module

The Pattern Clustering module performs the pattern clustering algorithm in previous chapters to group semantically similar lexical patterns into pattern clusters. Because the number of lexical patterns that we extract from the corpus is very large, we omit the Latent Relational Analysis step (i.e., the Singular Value Decomposition of the matrix **A**) before clustering. The size of the matrix **A** might be billions of rows (lexical patterns) and millions of columns (entity pairs). Therefore, to reduce the pre-processing time, we omit the SVD operation on the matrix A. As noted in Section 5.6.4, when we omit LRA, the performance will be slightly degraded. However, with this large number of lexical patterns, it is difficult to perform the LRA operation.

Moreover, we design the query processor to allow processing queries even when the Pattern Clustering algorithm has not been finished. This is because the pattern clustering algorithm takes a long time to complete. Therefore, we do not require the pattern clustering algorithm to be finished to begin processing queries. Instead, we can process queries and clustering lexical patterns simultaneously. This makes an extracted entities to be immediately retrievable for the result list, with a loss in precision of the search engine. However, in practice, if an entity pair holds a prominent semantic relation then the precision is still high, irrespective of relevant lexical patterns are clustered together or

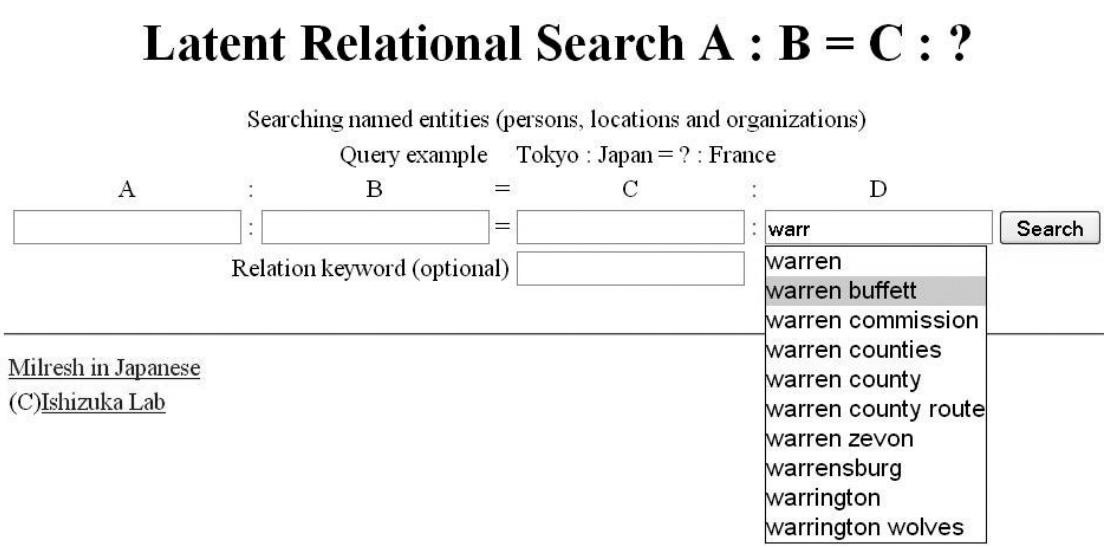


Fig. 6.8. An example of entity suggestion in Milresh when the partner entity is not given

not. Consequently, this method is appropriate to make the search engine more practical.

6.6 Input Entity Suggestion

The Suggestion Web Service in Figure 6.1 suggests users of Milresh with candidate entities in the index. We use the information in the HBase table *rs_entities* (as shown in Figure 6.3) to perform the suggestion task.

Given a partially input keyword, we must find appropriate entities to make a list of entities for suggestion. When the partner entity is not known (e.g., when the user is typing the first entity in the query), we suggest the high frequency entities that partially match the input keyword. For example, given the partially input keyword "warr", the system suggests entities such as "Warren Buffet", "Warrensburg" or "Warrington Wolves", as shown in Figure 6.8. We take the top 10 entities with highest frequencies to build the suggestion list. When there are several entities with the same frequency, we suggest all of them.

When a partner entity is given (for example, when the user has already input the first entity of the source pair and is typing the second entity), we lookup the index for entities that make pair with the partner entity. We then suggest entities with highest frequencies from this list, as shown in Figure 6.9.

6.7 The Query Processor and the Latent Relational Search Engine Front-End

The Query Processor implements the retrieval and ranking algorithm in previous chapters to retrieve a ranked list of entities as answers for a query. However, because the Query Processor in this section must interact with the HBase engine, it will result in disk accesses and network traffic. This might cause the query processing time to be longer than the time allowed for a search session of a user. Therefore, we limit the number of lexical patterns to be considered during the query processing phase by a frequency filter. Specifically, we only consider the top 100 lexical patterns with highest frequencies to retrieve the candidate

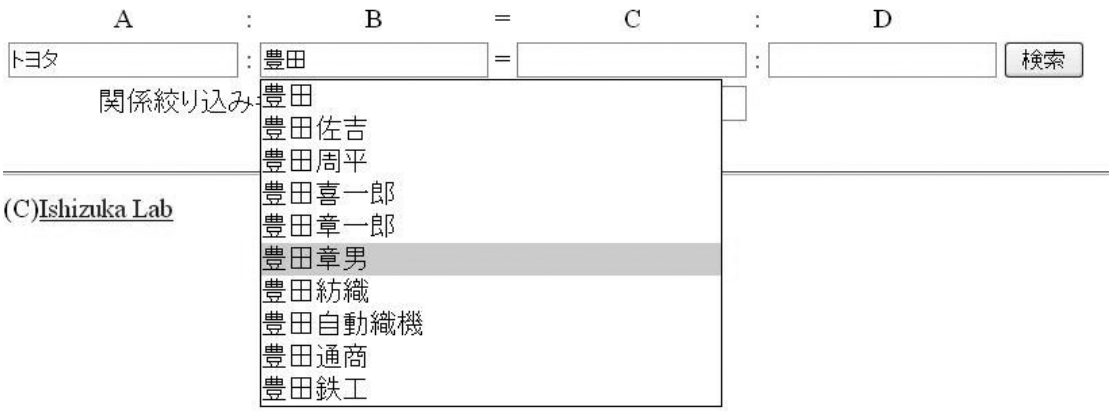


Fig. 6.9. An example of input entity suggestion when the first entity of the source pair is given. In the figure, the first entity is "Toyota" in Japanese, the second entity is partially given ("Toyoda" in this case). The system suggests a list of entities which are prefixed by "Toyoda", such as "Toyoda Akio" (the highlighted item).

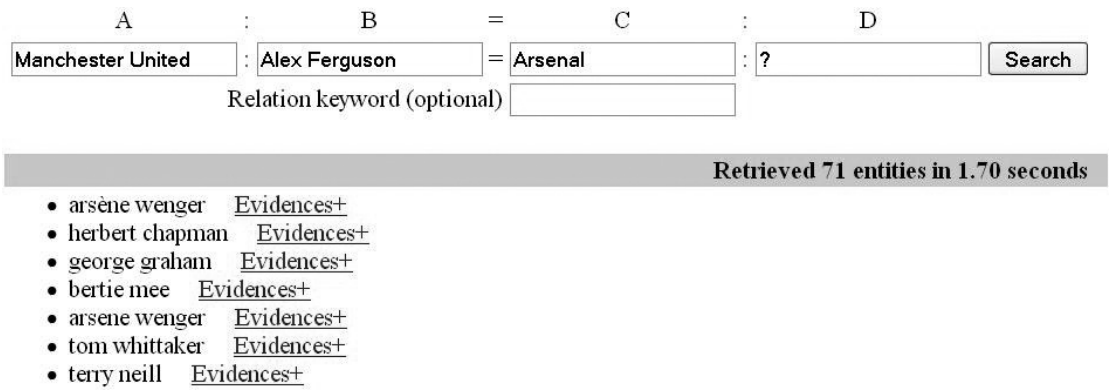


Fig. 6.10. An example result list

answer list.

The front-end for the search engine is a Tomcat Web application, which accepts input entities, provides the users with suggestions and queries the Query Processor to retrieve a result list for a query and then renders the final result list. We separate the entity retrieval phase from the supporting sentence retrieval phase while processing a query, as described in Section 4.8. Therefore, when we receive a query, we first retrieve the candidate entities and rank the candidate list. We then directly return this candidate list to the client, to reduce the query processing time from when the user inputs a query to the time when the user receives the result list. If the user click on the link to "Evidence+", we retrieve the evidence sentences for the selected entity, as shown in Figure 6.10, Figure 6.11 and Figure 6.12.

The front-end does not modify the database. Therefore, we can easily run the application in several machines for load balancing.

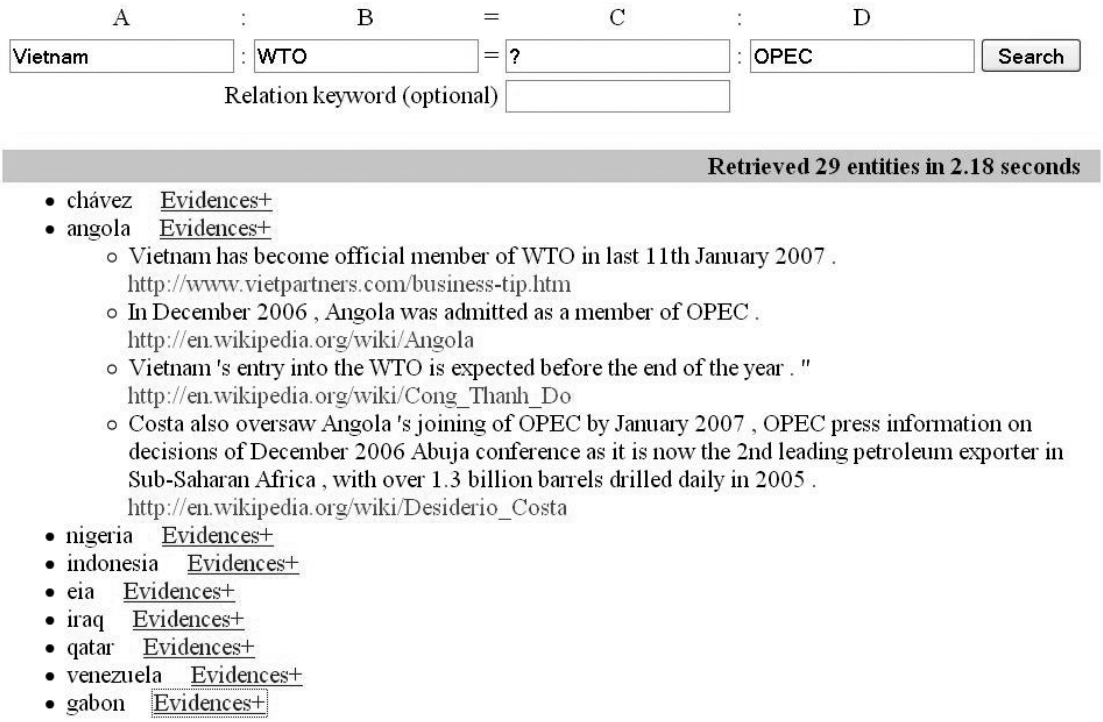


Fig. 6.11. An example supporting sentence list in English

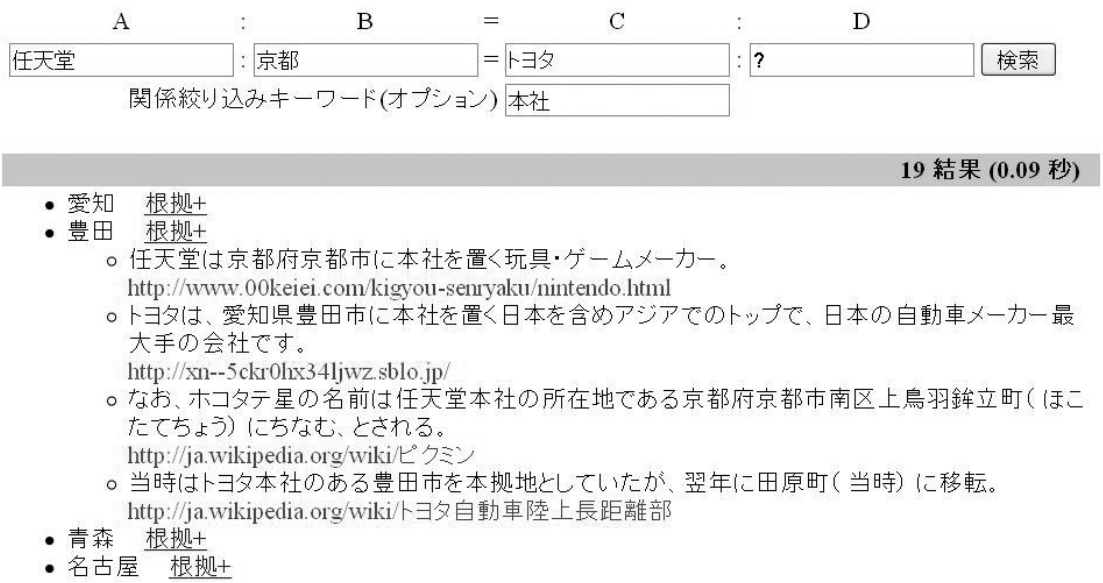


Fig. 6.12. An example supporting sentence list in Japanese (the query is {(Nintendo, Kyoto), (Toyota, ?)}, which is a company-headquarters query)

6.8 Evaluation

6.8.1 Datasets and experiments

In this section, we demonstrate the ability of answering sophisticated queries of many relation types using Milresh. Specifically, we evaluate the system with the standard query set from the INEX 2008 Entity Ranking task [117]. INEX is one of the main four forums for evaluating Information Retrieval systems, along with TREC, CLEF and NTCIR[*2]. We use the INEX 2008 Entity Ranking task as the benchmark for comparison because the entity ranking task is similar to latent relational search and the data for evaluation (the query set and the gold standard answers, as well as the auto-evaluation script) can be freely downloaded from the INEX 2008 homepage[*3].

In the INEX 2008 Entity Ranking task, there are 35 topics (questions) of several relation types[*4]. The task is to retrieve the Wikipedia page IDs of the answer entities for those questions. For example, the question number 104 is "I am looking for characters in the Harry Potter universe that are part of Gryffindor house or team and that play Quidditch". There are several answer entities for this questions, such as Charlie Weasley, Cormac McLaggen, ... who are the team members of the Quidditch team. In the INEX Entity Ranking task, the corpus that was used is the entire English Wikipedia.

To create an index for the search engine, we downloaded the Japanese Wikipedia XML data dump[*5] and the English Wikipedia XML data dump[*6]. We use the Parallel Distributed Relation Extractor in Figure 6.1 to extract text and perform relation extraction from these data dumps. As the result, we retrieved about seven million Wikipedia articles (of which 1.6 million articles are in Japanese Wikipedia). From these articles, we extracted 213,731,476 sentences. We use the default trained model of the Stanford Named Entity Recognizer (which is trained to recognize three types of named entities: LOCATION, ORGANIZATION and PERSON) to recognize named entities in each English sentence and the MeCab POS tagger to tag each Japanese sentence. We then perform the extraction process on these tagged sentences. After the extraction process, we obtained 6,688,119 named entities, which form 30,778,223 entity pairs. The total number of lexical pattern extracted is 945,857,689. Translating a large number of lexical patterns and entities using Google Translate causes a huge traffic and cost. Therefore, we do not translate entities and lexical patterns at the pre-processing step. Instead, when we receive a query, we only translate the lexical patterns and entities that are related to the input entities. Specifically, we only translate the source entity pair and lexical patterns that co-occur with the source entity pair (we do not use the MapReduce application for translation in this experiment, we only translate using a single thread). We then apply the second phase in the proposed clustering algorithm (HLPC) to these lexical patterns. Therefore, the time for processing a cross-lingual latent relational search query in this experiment is long because we must translate many lexical patterns. Consequently, using Milresh with this setting to process cross-lingual latent relational search queries is somewhat impractical (it takes a long time). However, note that if we can translate entities and lexical patterns in the pre-processing phase, then we can process cross-lingual latent relational search queries in high speed, as can be seen in the previous chapter. We were able to complete the

[*2] https://inex.mmci.uni-saarland.de/about.html
[*3] http://www.l3s.de/~demartini/XER08/
[*4] http://www.l3s.de/~demartini/XER08/inex08-xer-topics-final.xml
[*5] http://dumps.wikimedia.org/jawiki/20110921/
[*6] http://dumps.wikimedia.org/enwiki/20100817/

Table. 6.2. Performance of the system on the INEX 2008 Entity Ranking task (xinfAP is the inferred average precision, there are total of 35 questions)

Criterion	Monolingual (E-E)	Cross-lingual (J-E)
Percentage of answered questions	42.9%	34.3%
xinfAP (over all questions)	0.076	0.054
xinfAP (answered questions only)	0.178	0.156

pre-processing steps in seven days using five machines (each machine is an Intel Core i7, 3GHz x 6 processors, 24GB of RAM).

In the INEX Entity Ranking task, it is required to retrieve the Wikipedia page IDs of the pages that correspond to the answer entities (for example the entity "Charlie Weasley" has the page ID of 156649). Therefore, we must build a mapping from an entity to its Wikipedia page ID. Wikipedia provides an API to retrieve the entity that corresponds to a specified page ID [*7]. However, for evaluation, we must build a reverse mapping from an entity to its page ID. Consequently, we analyze the INEX 2008 answer list and retrieve all page IDs from the list. We then input these page IDs into the API provided by Wikipedia to build a hash table from entities to their Wikipedia page IDs.

We then manually create latent relational search queries to answer the questions in INEX. For example, for the question "I am looking for characters in the Harry Potter universe that are part of Gryffindor house or team and that play Quidditch", we create the monolingual query {(Steve Bruce, Manchester United), (?, Gryffindor Quidditch)} and the Japanese-to-English cross-lingual query {(*Yasuda Michihiro*, *Gamba Osaka*), (?, Gryffindor Quidditch)}. We query the proposed search engine to retrieve answer entities, and we use the mapping described above to retrieve the corresponding Wikipedia page IDs. Using the retrieved Wikipedia page IDs, we can output the result list in the INEX/TREC format. We then use the gold standard answers and the evaluation script from INEX[*8] to automatically evaluate the result. Therefore, the evaluation conditions are essentially identical with those of the INEX 2008 Entity Ranking task. That is, the question set and the evaluation metric are identical with those of INEX. However, the Wikipedia data dumps might be slightly different (because we download the dumps at a different date with the date that the dataset in INEX was dumped; INEX does not make the dumps that it used for evaluation public).

6.8.2 Experiment results

The result of the above experiment is shown in Table 6.2. The first row of Table 6.2 shows the percentage of questions that we could retrieve at least an answer (in the gold standard answer list). With monolingual queries (English-to-English), we could answer 15 (out of 35) questions. With cross-lingual queries (Japanese-to-English), we could answer 12 (out of 35) questions. The questions that we could not answer can be classified into two types. The first type contains questions that we could not create latent relational search queries. For example, the question number 126, "I want to compile an exhaustive list of toy train manufacturers that are still in business", is such a question. This question describes a sophisticated relation existing between an ORGANIZATION (company) and a product (toy train). We found no way to create a latent relational search query to answer this question. This also illustrates a situation in which it is difficult to use latent relational

[*7] http://en.wikipedia.org/w/api.php?action=query&indexpageids&pageids=156649
[*8] http://www.l3s.de/~demartini/XER08/INEX08-XER-testing-final.zip

search to fulfill the information need. Specifically, when the user wants to express multiple properties of a semantic relation (e.g., not only the relation "manufacturing" between a company and a product, but also the property of the relation, such as "still in business"), it is difficult to find an example entity pair as the source entity pair to input to a latent relational search engine. Another similar example is the question "I want a list of Nobel laureates in Physics from Bell Labs, who are French and still alive". A user might easily imagine some queries such as {(Leo Esaki, IBM), (?, Bell Labs)}, but the query does not directly answer the question.

The second type contains questions that we were able to make some reasonable queries, but the proposed system could not retrieve any answer. For example, for the question number 124, "The user wants a list of novels that won the Booker Prize", we created the query {(A Visit From the Goon Squad, Pulitzer Prize), (?, Booker Prize)}. However, we could not retrieve any answer. This is because the Stanford Named Entity Recognizer does not recognize "A Visit From the Goon Squad" as a named entity, instead it recognizes "Goon Squad" as a named entity.

The second row in Table 6.2 shows the inferred average precision (xinfAP) [120] of the entire query sets. Average precision (AP) represents the average of the precision values of an information retrieval system at all recall levels. For an information retrieval system, there is a tradeoff between precision and recall: one can achieve high precision by outputting only confident results, which makes the recall low. On the other hand, one can output every result to achieve 100% recall. However, this method would make the precision very low. Therefore, the average precision (AP) measures the average of the precision values at various recall levels to accurately reflect the performance of an information retrieval system. The average precision is not easy to compute, as it requires to calculate the integral of the precision function (with recall as the parameter). Several methods for estimating the average precision have been proposed [120]. These methods often calculate a measure called the "inferred average precision" (infAP). Yilmaz et al. propose a method for inferring the average precision, in which they call the metric as "xinfAP" [120]. This is the evaluation metric that INEX uses to evaluate the performance of an entity retrieval system [117]. The values in the second row of Table 6.2 are these xinfAP values. For the entire query set, the result of the proposed system is lower than the baseline of INEX 2008 [117]. This is because we failed to create queries for the majority of questions. In addition, for some questions, the system could not answer because the key entities contain names of novels, names of movies or names of awards that the Stanford Named Entity Recognizer is not trained to recognize (it actually mistakenly recognizes some movie names as LOCATIONs or PERSONs, therefore, we could extract some movies, novels or awards).

If we exclude the questions that we could not create appropriate latent relational search queries and the questions that the system could not answer, then the result is as shown in the last row of Table 6.2. This metric reflects the ability of ranking answer entities of the proposed system when we find at least one result. With monolingual search queries, the proposed system slightly outperforms the baseline systems in INEX 2008 (xinfAP is 0.178, compared to 0.111 and 0.159 of the two baselines [117]). The performance is about a half of the best performance in INEX 2008 (0.341). However, note that in the INEX 2008 Entity Ranking task, the participants are allowed to use the information in the title and description of the questions, which include several informative keywords. On the other hand, in the proposed system, we only have the input source entity pair (which is manually created by human), and we need to represent the relation by some lexical patterns. Even in this setting, the proposed system achieves the same level of average precision with baseline systems in INEX. Moreover, with Japanese-to-English cross-lingual queries, we also achieve an xinfAP of 0.156, which is only slightly smaller

Table. 6.3. Excerpt of monolingual queries to answer INEX questions

Question (topic ID)	Query (monolingual)
Harry Potter Quidditch Gryffindor character (104)	{(Steve Bruce, Manchester United), (?,Quidditch Gryffindor)}
State capitals of the U.S (108)	{(Hong Gai, Quang Ninh province), (?, New Hampshire)}
State capitals of the U.S (108)	{(Hong Gai, Quang Ninh province), (?, Florida)}
Formula 1 drivers that won the Monaco Grand Prix (113)	{(Serena Williams, Wimbledon), (?, Monaco Grand Prix)}
Italian nobel prize winners (116)	{(Leo Esaki, Japan), (?, Italy)}
Musicians who appeared in the Blues Brothers movies (127)	{(Mick Molloy, TomBoy), (?, Blues Brothers)}
Star Trek Captains (130)	{(Kirk Chase Joker, Harry Potter), (?, Star Trek)}
Professional baseball team in Japan (135)	{(New York Yankees, America), (?, Japan)}

than that of the monolingual queries. This proves that the proposed method could be used to answer real-world questions, provided that the user of the system could imagine good input source entity pairs to formulate queries. Finding a good source pair as an example is much simpler than enumerating a long list of appropriate keywords.

Table 6.3 shows some example queries that we created to answer questions in INEX. For some questions (such as "I want a list of living classical composers, who are born in Nordic countries.", we need to break the questions into several queries, such as {(Mamoru Fujieda, Japan), (?, Denmark)}, {(Mamoru Fujieda, Japan), (?, Norway)}, ...).

6.8.3 Query processing time of Milresh

As mentioned in Section 6.8.1, we could not translate the entire set of extracted lexical patterns using the Google Translate APIs in the pre-processing step. We only translate related lexical patterns when we receive a cross-lingual query. Therefore, the query processing time for cross-lingual queries of the system in this setting is very long. Moreover, the time also depends on external systems (translation APIs) and events (e.g., external network traffic). Consequently, in this section, we only evaluate the query processing time of Milresh for monolingual latent relational search queries. Specifically, we use eight English monolingual query sets and eight Japanese monolingual query sets correspond to eight relation types in Table 4.3 for evaluation. After the evaluation, we recognize that the time for processing queries in the English monolingual query set regarding the CAPITAL relation is worst (longest). Consequently, we describe in detail the query processing time for queries in this query set below.

The English monolingual query set CAPITAL includes 90 queries concerning the capital cities of ten countries. It takes the longest time because the pairs that hold the capital relation (e.g., (Tokyo, Japan)) are very popular entity pairs. As the result, these entity pairs co-occur with a large number of lexical patterns. Therefore, the candidate set of each query in this query set is very large and the time for calculating the relational similarity is also long.

Table 6.4 shows an excerpt of the query patterns and the query processing time we observed in the experiment. Each block contains queries with the same input source entity

Table. 6.4. Excerpt of query patterns and query processing time of Milresh (consecutive queries in a block are input into Milresh consecutively). The last row shows the average time and standard deviation of 90 queries in the CAPITAL set.

Query	Query processing time (s)
{(hanoi, vietnam), (?, japan)}	1.83
{(hanoi, vietnam), (?, france)}	1.26
{(hanoi, vietnam), (?, germany)}	1.19
...	...
{(tokyo, japan), (?, vietnam)}	3.26
{(tokyo, japan), (?, france)}	2.40
{(tokyo, japan), (?, germany)}	2.33
...	...
{(paris, france), (?, vietnam)}	4.59
{(paris, france), (?, japan)}	3.64
{(paris, france), (?, germany)}	3.76
...	...
{(berlin, germany), (?, vietnam)}	3.26
{(berlin, germany), (?, japan)}	2.59
{(berlin, germany), (?, france)}	2.73
...	...
{(london, england), (?, vietnam)}	7.13
{(london, england), (?, japan)}	5.27
{(london, england), (?, france)}	5.12
...	...
{(vienna, austria), (?, vietnam)}	5.42
{(vienna, austria), (?, japan)}	4.20
{(vienna, austria), (?, france)}	4.36
...	...
{(berne, switzerland), (?, vietnam)}	2.38
{(berne, switzerland), (?, japan)}	1.73
{(berne, switzerland), (?, france)}	1.84
...	...
{(seoul, korea), (?, vietnam)}	3.53
{(seoul, korea), (?, japan)}	2.24
{(seoul, korea), (?, france)}	2.31
...	...
{(madrid, spain), (?, vietnam)}	5.15
{(madrid, spain), (?, japan)}	4.10
{(madrid, spain), (?, france)}	4.22
...	...
{(cairo, egypt), (?, vietnam)}	2.19
{(cairo, egypt), (?, japan)}	1.90
{(cairo, egypt), (?, france)}	1.92
...	...
Average	3.00 $\pm$ 1.27

pair (e.g., the first block contains queries in which the source pair is (Hanoi, Vietnam)). The queries in the same block are input into Milresh in the same order with the order of the queries in the table.

From Table 6.4, we can observe that the first query in each block always takes longer time than other queries in the block. This is because Milresh must access to the database (HBase) to retrieve the source entity pair and the related lexical patterns when processing

Table. 6.5. Comparison between the average query processing time of the proposed method with that of the method **TC** in [8]

Method	Average query processing time (s)
TC (querying external search engines) [8]	771.34 ± 535.63
Proposed method (local index)	3.00 ± 1.27

a query. In the first query in each block, HBase does not have the cache for the row corresponding to the source entity pair. On the other hand, from the second query in each block, HBase has the cache, therefore the query processing time is faster. The average query processing time of 90 queries is 3.00 ± 1.27 seconds. We observe that for some queries, the query processing time is up to 30 seconds. However, the number of these queries is very small. Consequently, we conclude that Milresh guarantees practical query processing time for the majority of queries.

To compare the query processing time of Milresh with that of a latent relational search engine based on the method **TC** by Kato et al. [8], we reimplemented the method **TC** using Google[*9] as the external keyword-based Web search engine. To process a latent relational search query, the method **TC** must issue tens or hundreds of keyword-based Web search queries to Google. We experience that if a host issues consecutive queries to Google without any waiting time between two queries, then Google will block the host. Consequently, we must wait about 20 seconds between each query, to avoid the access denial from Google. With this setting, the comparison of the average query processing time of 90 queries is shown in Table 6.5. Because of the wait time between two Google queries, the average query processing time of the method **TC** is very long. Even if we could reduce the wait time between two Google queries to one second and we suppose that in that one second we can process all data from Google, then we have the average query processing time of 771 / 20, which is about 38 seconds (in practice we would not achieve this query processing time without a cooperation from Google).

Table 6.6 shows some practical queries and results of the Milresh system. The first and the second query in Table 6.6 represent the situations in which the user wants a comprehensive list of entities in a specific semantic relation with a given entity. Using a keyword-based Web search engine, the user must input several keywords, such as "member of OPEC", "entry to OPEC", "join OPEC" or "Chelsea manager", "Chelsea coach" ... to obtain the list. On the other hand, with latent relational search, the user can simply give the search engine an example entity pair that holds the same relation, such as (Vietnam, WTO) or (Alex Ferguson, Manchester United) to achieve the same results. Therefore, latent relational search can be effectively used when the relation is stated in several different ways.

The third query in Table 6.6 represents a more complicated example: the user wants to find a famous search engine service/company in China. It is not easy to imagine a query such as "search engine giants in China" or "Chinese search engine giants" to query a keyword-based Web search engine. The most natural query for an American/Japanese user should be {(Google, USA), (?, China)}.

Other examples in the table show some situations where the user specifies additional properties of the relation (e.g., "ancient" capital) or the user wants the exact answer entities of some queries. Note that with traditional keyword-based Web search engines, the user must read a set of text snippets to find the answer. On the other hand, latent relational search directly provides the answer entities for each question.

[*9] http://www.google.com

Table. 6.6. Some example queries and results of Milresh

Information need	Query	Answer
List of member countries of OPEC	{(Vietnam, WTO), (?, OPEC)}	Indonesia, Angola, Iraq, ...
List of coaches of Chelsea Football Club	{(Alex Ferguson, Manchester United), (?, Chelsea)}	Jose Mourinho, Carlo Ancelotti, ...
Search engine giants in China	{(Google, USA) (?, China)}	Baidu
Ancient capital of Vietnam	{(Kyoto, Japan), (?, Vietnam)}	Hue
List of cities which are/were the capital of Japan	{Tokyo, Japan), (?, Japan)}	Nara, Kyoto, Tokyo,...
Professional baseball teams in Japan	{New York Yankees, America), (?, Japan)}	Yomiuri Giants, Saitama Seibu Lions, ...
Einstein's birthplace	{(Leonardo da Vinci, Vinci), (Albert Einstein, ?)}	Ulm, Wurttemberg, Germany
Apple's headquarters	{(Google, Mountain View), (Apple, ?)}	Cupertino
...	...	...

Chapter 7

Conclusion

7.1 Conclusion

In this thesis, we investigated the problem of latent relational search using Web corpora. We proposed a retrieval model for monolingual latent relational search based on the concept of relational similarity between two entity pairs. Following previous work on relation extraction, we represent the semantic relations between two entities in an entity pair by lexical patterns of the context surrounding the two entities. We then propose a method for extracting entities and relations from Web corpora to create an index for a high speed latent relational search engine. We rely on previous work on dimensionality reduction techniques in relational similarity measurement to propose a relational similarity measuring method that is optimized for latent relational search. The proposed relation representation method enables our search engine to achieve high precision and Mean Reciprocal Rank (MRR) because it works well even when the lexical patterns between entity pairs are not identical, thereby solving the data sparseness problem regarding exactly matched lexical patterns. We evaluated the performance of the proposed method on many types of relations. The evaluation results show that we obtained a high MRR as well as precision for the Top 1 ranked result. While comparing with a previous method for latent relational search, the proposed method achieves better results. Specifically, when evaluating with an ideal corpus, the proposed search engine retrieves the correct answer in the Top 1 ranked result for 94% of monolingual queries in English and 88% in Japanese.

Moreover, we propose cross-language latent relational search to exploit the multi-lingual Web for latent relational search. Cross-lingual latent relational search is an advanced latent relational search paradigm which allows answering the query {(A, B), (C, ?)} when the input pair (A, B) is written in another language from the language of the entity C. We propose a method for extending the retrieval model in monolingual latent relational search to obtain a retrieval model for cross-lingual latent relational search. In particular, we propose a novel two-phase lexical pattern clustering algorithm to recognize paraphrased lexical patterns across languages, thereby effectively ranking candidates and retrieving evidence sentences for cross-lingual queries. When evaluating with English-Japanese cross-lingual latent relational search query sets, the search engine achieves an MRR of 0.605. Importantly, for the majority of cross-lingual queries, the search engine retrieves supporting sentences that are semantically similar in two different languages. This implies that the results of the search engine can be used for building parallel corpora or for supporting human translators.

Finally, we implemented a large-scale latent relational search engine named Milresh. We use Wikipedia data dumps as corpora to build the index for the search engine, thereby showing that the proposed model can be applied to build a large-scale latent relational search engine with practical corpora. To evaluate the search engine, we extract more

than seven million articles in the English and Japanese Wikipedia data dumps to build an index for the search engine and use the search engine to answer several sophisticated questions in the INEX 2008 Entity Ranking task. The results show that, the search engine was able to answer 15 (out of 35) queries in monolingual mode, where as, in cross-lingual settings, the number of successfully answered questions was 12 (out of 35). The average query processing time of monolingual queries is three seconds, which is acceptable for real-world users' search sessions. This demonstrates that the proposed system could be used for answering sophisticated questions concerning entities and relations on the Web.

From these results, we expect that latent relational search could open a new direction in information retrieval and question answering, especially for dealing with questions in which search engine users could not provide relevant keywords.

7.2 Future Work

7.2.1 Ranking Evidence Sentences

In this work, we simply retrieve a set of evidence sentences by using the common lexical patterns between two entity pairs. We then prune the set of evidence sentences to restrict the size of the set. When pruning the sentence set, we simply use the order in which we retrieve the sentence in the database to rank and prune these sentences. This causes some inappropriate sentences to be ranked as top results and some appropriate sentences to be mistakenly dropped. For example, when a user searches for evidence sentences of the pairs (Tokyo, Japan) and (Paris, France), we might retrieve several sentences such as "He was born in Tokyo, Japan." and "Jacques was born in Paris, France". These sentences are retrieved because they share the lexical pattern "was born in X, Y". However, these sentences might not be appropriate when the user is looking for the capital relation.

A solution to the problem is to assign scores to lexical patterns to judge which lexical patterns are important to an entity pair. For example, for the pair (Tokyo, Japan), we want to prioritize the lexical pattern "X is the capital of Y" when retrieving evidence sentences. Therefore, we need a method that can assign high score to this lexical pattern, when considering the entity pair (Tokyo, Japan). We are conducting research on this topic to extract important lexical patterns for an entity pair thereby can rank important sentences as the top of the evidence sentence list.

7.2.2 Extracting Common Noun Pairs

This work proposes the general method for latent relational search. It does not restrict the types of the entities that it can index. However, in the implementation, we omit all common nouns, we only consider three types of named entities because named entities are often interesting to search engine users. In future, we want to extract all noun pairs and noun-verb pairs. If we also extract common nouns, we can search for proper noun - common noun pairs such as {(Google, web search), (YouTube, ?)}, for which the answer could be "video sharing". Another example if we can extract common noun is that, we can answer questions like {(Operating System, kernel), (Information Retrieval system, ?)}, for which the answer could be "index".

To be able to extract and index common nouns, we must invent a method to filter out inappropriate common noun pairs, otherwise we will face the explosion problem in the number of extracted pairs. A method to filter out entity pairs that do not hold a strong semantic relation is using a dependency parser to analyze each sentence and extract only pairs with strong relations.

7.2.3 Using Deeper Analysis to Improve the Precision

A limitation of the proposed system is that it currently does not perform deep linguistic analysis, such as syntactic analysis or co-reference resolution. Lacking co-reference resolution might cause problems while extracting relationships between entities. For example, the current relation extraction algorithm could not extract the CAPITAL relation between Tokyo and Japan from the two sentences "Tokyo is one of the largest city in Asia. It is the capital of Japan". To compensate for this problem, we must analyze a larger text corpus (with which we can still extract the desired relationships, even when we fail to extract several relations mentioned indirectly by co-references). Therefore, the proposed method might require a huge corpus to achieve a high coverage over the query spectrum.

Similarly, because of lacking syntactic and semantic analysis, sometime we face the problem regarding negative sentences and speculative sentences. For example, from the sentence "Microsoft does not acquire Yahoo.", the propose extraction algorithm also extracts the pattern "X $*$ acquire Y". This might lead to errors in answering queries concerning the acquisition relation, such as {(Google, YouTube), (Microsoft, ?)}. Moreover, there are also some speculative sentences, which do not represent valid facts. For example, the sentence "It is guessed that Microsoft will acquire Yahoo." is a speculative sentence. These type of sentences frequently appear on the Web because there are several rumors concerning the acquisition of Yahoo by Microsoft. The acquisition of Yahoo by Microsoft is only a rumor, but the proposed extraction algorithm would extract lexical patterns concerning the acquisition relation because it does not recognize that the above sentence is only a speculative sentence. This might result in inappropriate answers in the final result list for a query.

We are conducting research about negative and speculative sentence detection. Using the result of the detection process, we can add a negative marker into lexical patterns extracted from negative sentences. For example, from the sentence "Microsoft does not acquire Yahoo.", we would add a negative tag such as "negptn" at the end of the extracted lexical patterns, such as "X $*$ acquire Y *negptn*" to differentiate between the patterns from a positive sentence. These modifications would improve the performance of the proposed latent relational search engine.

However, from the research concerning negative sentence detection, we notice that with current limitations of natural language processing (e.g., syntactic analysis speed, precision of dependency parsers ...), the proposed extraction algorithm in this thesis is practical for processing a huge corpus to achieve an index for a latent relational search engine. Future developments in linguistic processing technologies would give us more options in our algorithm to achieve a more accurate relation extractor.

7.2.4 Exploiting Temporal Aspects in Relational Similarity Measuring

The set of semantic relations between two entities is not a static set. Instead, these relations are changed over time. For example, a year before the time that this thesis is written, Steve Jobs was the CEO of the Apple Inc. Therefore, the correct answer for the query {(Microsoft, Steve Ballmer), (Apple, ?)} should be "Steve Jobs". However, the answer might become less reasonable now, because Steve Jobs is not the current CEO of the Apple Inc. With the proposed latent relational similarity measuring method, the score for Steve Jobs would be very high, because there are many common lexical patterns between the pair (Apple, Steve Jobs) and (Microsoft, Steve Ballmer). Consequently, it would be useful if we integrate temporal aspects of the extracted semantic relations in the relational similarity measuring step. This might make the score for "Tim Cook" (the

current CEO of Apple) higher than that of "Steve Jobs", thereby improve the ranking process. Moreover, by considering temporal features, it is allowed to specify the "time range" in a query. This leads to latent relational search queries to exactly retrieve the CEO of the Apple Inc. in a specified time period.

Publications and Research Activities

(1) Nguyen Tuan Duc, Danushka Bollegala and Mitsuru Ishizuka. Cross-Language Latent Relational Search between Japanese and English Languages Using a Web Corpus. *ACM Transactions on Asian Language Information Processing (ACM TALIP)* (accepted, to appear).

(2) Nguyen Tuan Duc, Danushka Bollegala and Mitsuru Ishizuka. Cross-Language Latent Relational Search: Mapping Knowledge across Languages. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI 2011)*, San Francisco, USA, pp.1237-1242 (2011.8) (AI and the Web track).

(3) Nguyen Tuan Duc, Danushka Bollegala and Mitsuru Ishizuka. Exploiting Relational Similarity between Entity Pairs for Latent Relational Search. *Journal of the Information Processing Society of Japan (IPSJ)*, Vol. 52, No. 4, pp.1790-1802 (2011.4) (in Japanese).

(4) Nguyen Tuan Duc, Danushka Bollegala and Mitsuru Ishizuka. Relation Representation and Indexing Method for a Fast and High Precision Latent Relational Web Search Engine. *Transactions of the Japanese Society for Artificial Intelligence (TJSAI)*, Vol. 26, No. 2, pp.307-312 (2011.2) (short paper, in Japanese) .

(5) Nguyen Tuan Duc, Danushka Bollegala and Mitsuru Ishizuka. Using Relational Similarity between Word Pairs for Latent Relational Search on the Web. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI 2010)*, Toronto, Canada, pp.196-199 (2010.9) (short paper).

(6) Nguyen Tuan Duc, Danushka Bollegala and Mitsuru Ishizuka. Relation Representation and Indexing Method for Fast and High Precision Latent Relational Web Search Engine. In *Proceedings of the 24th Annual Conference of the Japanese Society for Artificial Intelligence (JSAI 2010)*, (in Japanese) (received **JSAI Annual Conference Award**).

(7) Nguyen Tuan Duc, Danushka Bollegala, and Mitsuru Ishizuka. Latent Relational Search Engine. *Report for 2011 Google Research Award on Information Retrieval, Extraction and Organization*, Googleplex, Mountain View, CA (2011.8) (Principal Investigator and Award Winner: Prof. Mitsuru Ishizuka).

(8) Danushka Bollegala, Mitsuru Ishizuka and Nguyen Tuan Duc. Relational Search: Approaches and Systems. *Japan patent application No. 2009-275762*, (2009.12) (in Japanese)

Co-authored work (selected):

(1) Kazuki Hirashima, Sho Kawasaki, Nguyen Tuan Duc, Danushka Bollegala and Mitsuru Ishizuka. Ranking Evidence Sentences in Latent Relational Search. In *Proceedings of the 74th National Convention of the Information Processing Society of Japan (IPSJ 2012)* (2012.3) (to appear, in Japanese).

(2) Nobuhiko Ochiai, Nguyen Tuan Duc, Danushka Bollegala and Mitsuru Ishizuka. Negative Sentence Detection for Improving Precision of Latent Relational Search. In *Proceedings of the 74th National Convention of the Information Processing Society of Japan (IPSJ 2012)* (2012.3) (to appear, in Japanese).

(3) Yorihiro Nobuta, Nguyen Tuan Duc, Danushka Bollegala and Mitsuru Ishizuka. Method for Indexing Broader Range of Entity Types in Latent Relational Search. In *Proceedings of the 74th National Convention of the Information Processing Society of Japan (IPSJ 2012)* (2012.3) (to appear, in Japanese).

(4) Tomokazu Goto, Nguyen Tuan Duc, Danushka Bollegala and Mitsuru Ishizuka. Improving Relational Search Performance Using Relational Symmetries and Predictors. *Transactions of the Japanese Society for Artificial Intelligence (TJSAI)*, Vol. 26, No. 6, pp. 649-656 (2011.9) (in Japanese)

(5) Sho Kawasaki, Nguyen Tuan Duc, Danushka Bollegala and Mitsuru Ishizuka. Accurate Relation Name Extraction from Entity Pair Set Using the Web. In *Proceedings of the 73rd National Convention of the Information Processing Society of Japan, (IPSJ 2011)* (2011.3) (in Japanese)

(6) Tomokazu Goto, Nguyen Tuan Duc, Danushka Bollegala and Mitsuru Ishizuka. Exploiting Symmetry in Relational Similarity for Ranking Relational Search Results. In *Proceedings of the 11th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2010)*, Daegu, Korea, pp. 595-600 (2010.8) (short paper) (received **Best Poster Award**).

References

[1] Jiafeng Guo, Gu Xu, Xueqi Cheng, and Hang Li. Named Entity Recognition in Query. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009*, pages 267–274, Boston, MA, USA, 2009.

[2] Xiaoxin Yin and Sarthak Shah. Building Taxonomy of Web Search Intents for Name Entity Queries. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010*, pages 1001–1010, Raleigh, North Carolina, USA, 2010.

[3] Oren Etzioni, Michael J. Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised Named-entity Extraction from the Web: An Experimental Study. *Artificial Intelligence*, 165(1):91–134, 2005.

[4] Jakob Halskov and Caroline Barriere. Web-based Extraction of Semantic Relation Instances for Terminology Work. *Terminology*, 14(1):20–44, 2008.

[5] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open Information Extraction from the Web. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007*, pages 2670–2676, Hyderabad, India, 2007.

[6] Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. Relational Duality: Unsupervised Extraction of Semantic Relations between Entities on the Web. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010*, pages 151–160, Raleigh, North Carolina, USA, 2010. ACM.

[7] Michele Banko and Oren Etzioni. The Tradeoffs Between Open and Traditional Relation Extraction. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, ACL 2008*, pages 28–36, Columbus, Ohio, USA, 2008.

[8] Makoto P. Kato, Hiroaki Ohshima, Satoshi Oyama, and Katsumi Tanaka. Query by Analogical Example: Relational Search using Web Search Engine Indices. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009*, pages 27–36, Hong Kong, 2009. ACM.

[9] Tomokazu Goto, Nguyen Tuan Duc, Danushka Bollegala, and Mitsuru Ishizuka. Exploiting Symmetry in Relational Similarity for Ranking Relational Search Results. In *Proceedings of the 11th Pacific Rim International Conference on Artificial Intelligence, PRICAI 2010*, pages 595–600, Daegu, Korea, 2010.

[10] Xiaoxin Yin, Wenzhao Tan, and Chao Liu. FACTO: A Fact Lookup Engine Based on Web Tables. In *Proceedings of the 20th International Conference on World Wide Web, WWW 2011*, pages 507–516, Hyderabad, India, 2011.

[11] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[12] Hang Cui, Ji-Rong Wen, Jian-Yun Nie, and Wei-Ying Ma. Probabilistic Query Expansion Using Query Logs. In *Proceedings of the 11th International Conference on World Wide Web, WWW 2002*, pages 325–332, Honolulu, Hawaii, USA, 2002.

[13] Daniel Crabtree, Peter Andreae, and Xiaoying Gao. Exploiting Underrepresented

Query Aspects for Automatic Query Expansion. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2007*, pages 191–200, San Jose, California, USA, 2007.

[14] Yuan Lin, Hongfei Lin, Song Jin, and Zheng Ye. Social Annotation in Query Expansion: A Machine Learning Approach. In *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011*, pages 405–414, Beijing, China, 2011.

[15] David J. Chalmers, Robert M. French, and Douglas Hofstadter. High-level Perception, Representation, and Analogy: A Critique of Artificial Intelligence Methodology. *Journal of Experimental & Theoretical Artificial Intelligence*, 4(3):185–211, 1992.

[16] Douglas Hofstadter and FARG. *Fluid Concepts and Creative Analogies. Computer Models of the Fundamental Mechanisms of Thought.* Basic Books, 1995.

[17] Dedre Gentner. Structure-mapping: A Theoretical Framework for Analogy. *Cognitive Science*, 7(2), 1983.

[18] Tony Veale. The Analogical Thesaurus. In *Proceedings of the 15th Conference on Innovative Applications of Artificial Intelligence, IAAI 2003*, pages 137–142, Acapulco, Mexico, 2003. AAAI Press.

[19] Peter D. Turney. The Latent Relation Mapping Engine: Algorithm and Experiments. *Journal of Artificial Intelligence Research (JAIR)*, 33:615–655, 2008.

[20] Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. Measuring the Similarity between Implicit Semantic Relations from the Web. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009*, pages 651–660, Madrid, Spain, 2009. ACM.

[21] Cody C. T. Kwok, Oren Etzioni, and Daniel S. Weld. Scaling Question Answering to the Web. In *Proceedings of the 10th International World Wide Web Conference, WWW 2001*, pages 150–161, Hong Kong, China, 2001.

[22] Susan T. Dumais, Michele Banko, Eric Brill, Jimmy J. Lin, and Andrew Y. Ng. Web Question Answering: Is More Always Better? In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2002*, pages 291–298, Tampere, Finland, 2002.

[23] Hideki Isozaki, Katsuhito Sudoh, and Hajime Tsukada. NTT's Japanese-English Cross-Language Question Answering System. In *Proceedings of NTCIR-5 Workshop Meeting, NTCIR 2005*, Tokyo, Japan, 2005.

[24] Anselmo Peñas, Pamela Forner, Richard F. E. Sutcliffe, Álvaro Rodrigo, Corina Forascu, Iñaki Alegria, Danilo Giampiccolo, Nicolas Moreau, and Petya Osenova. Overview of ResPubliQA 2009: Question Answering Evaluation over European Legislation. In *Proceedings of the 10th Workshop of the Cross-Language Evaluation Forum, CLEF 2009*, pages 174–196, 2009.

[25] Sergio Ferrández, Antonio Toral, Óscar Ferrández, Antonio Ferrández, and Rafael Muñoz. Exploiting Wikipedia and EuroWordNet to Solve Cross-Lingual Question Answering. *Information Sciences*, 179(20):3473–3488, 2009.

[26] Eiichiro Sumita. Example-Based Machine Translation Using DP-matching between Word Sequences. In *Proceedings of the Workshop on Data-driven Methods in Machine Translation, DMM 2001*, pages 1–8, Stroudsburg, PA, USA, 2001. ACL.

[27] Peter D. Turney. Similarity of Semantic Relations. *Computational Linguistics*, 32(3):379–416, 2006.

[28] Peter D. Turney, Michael L. Littman, Jeffrey Bigham, and Victor Shnayder. Combining Independent Modules in Lexical Multiple-Choice Problems. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing*

RANLP 2003, pages 101–110, Borovets, Bulgaria, 2003.

[29] Peter D. Turney. Measuring Semantic Similarity by Latent Relational Analysis. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI 2005*, pages 1136–1141, Edinburgh, Scotland, 2005.

[30] Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. Measuring the Similarity between Implicit Semantic Relations using Web Search Engines. In *Proceedings of the 2nd International Conference on Web Search and Web Data Mining, WSDM 2009*, pages 104–113, Barcelona, Spain, 2009. ACM.

[31] Peter D. Turney and Michael L. Littman. Corpus-based Learning of Analogies and Semantic Relations. *Machine Learning*, 60(1-3):251–278, 2005.

[32] Douglas L. Medin, Robert L. Goldstone, and Dedre Gentner. Similarity Involving Attributes and Relations: Judgments of Similarity and Difference Are Not Inverses. *Psychological Science*, 1(1):64–69, 1990.

[33] Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. A Web Search Engine-Based Approach to Measure Semantic Similarity between Words. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 23(7):977–990, 2011.

[34] Peter D. Turney and Patrick Pantel. From Frequency to Meaning: Vector Space Models of Semantics. *Journal of Artificial Intelligence Research (JAIR)*, 37:141–188, 2010.

[35] Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. Measuring Semantic Similarity between Words Using Web Search Engines. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007*, pages 757–766, Banff, Alberta, Canada, 2007.

[36] Peter D. Turney. Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In *Proceedings of the 12th European Conference on Machine Learning, ICML 2001*, pages 491–502, Freiburg, Germany, 2001.

[37] Philip Resnik. Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language. *Journal of Artificial Intelligence Research (JAIR)*, 11:95–130, 1999.

[38] Dekang Lin. An Information-Theoretic Definition of Similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning, ICML 1998*, pages 296–304, Madison, Wisconsin, USA, 1998.

[39] Alexander Budanitsky and Graeme Hirst. Evaluating WordNet-based Measures of Lexical Semantic Relatedness. *Computational Linguistics*, 32(1):13–47, 2006.

[40] Dekang Lin. Automatic Retrieval and Clustering of Similar Words. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics, COLING-ACL 1998*, pages 768–774, Quebec, Canada, 1998.

[41] Petr Knoth, Jakub Novotny, and Zdenek Zdráhal. Automatic Generation of Inter-passage Links Based on Semantic Similarity. In *Proceedings of the 23rd International Conference on Computational Linguistics, COLING 2010*, pages 590–598, Beijing, China, 2010.

[42] Aminul Islam and Diana Zaiu Inkpen. Semantic Text Similarity Using Corpus-Based Word Similarity and String Similarity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2(2), 2008.

[43] Rebecca Green, Lisa Pearl, Bonnie J. Dorr, and Philip Resnik. Mapping Lexical Entries in a Verbs Database to WordNet Senses. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics, ACL 2001*, pages 244–251, Toulouse, France, 2001.

[44] Sadao Kurohashi and Makoto Nagao. Automatic Detection of Discourse Structure by Checking Surface Information in Sentences. In *Proceedings of the 15th International*

Conference on Computational Linguistics, COLING 1994, pages 1123–1127, Kyoto, Japan, 1994.

[45] Christopher D. Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[46] Tony Veale. Creative Language Retrieval: A Robust Hybrid of Information Retrieval and Linguistic Creativity. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, ACL 2011*, pages 278–287, Portland, Oregon, USA, 2011.

[47] Amos Tversky. Features of Similarity. *Psychological Review*, 84(4):327–352, 1977.

[48] Douglas L. Medin, Robert L. Goldstone, and Dedre Gentner. Respects for Similarity. *Psychological Review*, 100(2):254–278, 1993.

[49] Peter D. Turney. A Uniform Approach to Analogies, Synonyms, Antonyms, and Associations. In *Proceedings of the 22nd International Conference on Computational Linguistics, COLING 2008*, pages 905–912, Manchester, UK, 2008.

[50] Krisztian Balog, Pavel Serdyukov, and Arjen P. de Vries. Overview of the TREC 2010 Entity Track. In *Proceedings of the 19th Text REtrieval Conference, TREC 2010*, Maryland, USA, 2010.

[51] Brian Falkenhainer, Kenneth D. Forbus, and Dedre Gentner. The Structure-Mapping Engine: Algorithm and Examples. *Artificial Intelligence*, 41(1):1–63, 1989.

[52] Jin Yan and Kenneth D. Forbus. Similarity-Based Qualitative Simulation. In *Proceedings of the 27th Annual Meeting of the Cognitive Science Society*, Stresa, Italy, 2005.

[53] Kenneth D. Forbus, Matthew Klenk, and Thomas R. Hinrichs. Companion Cognitive Systems: Design Goals and Lessons Learned So Far. *IEEE Intelligent Systems*, 24(4):36–46, 2009.

[54] Vivi Nastase, Jelber Sayyad-Shirabad, Marina Sokolova, and Stan Szpakowicz. Learning Noun-Modifier Semantic Relations with Corpus-based and WordNet-based Features. In *Proceedings of the 21st National Conference on Artificial Intelligence, AAAI 2006*, Boston, Massachusetts, USA, 2006.

[55] Vivi Nastase and Stan Szpakowicz. Exploring Noun-Modifier Semantic Relations. In *Proceedings of the 5th International Workshop on Computational Semantics, IWCS 2003*, Tilburg, The Netherlands, 2003.

[56] Dmitry Davidov and Ari Rappoport. Classification of Semantic Relationships between Nominals Using Pattern Clusters. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, ACL 2008*, pages 227–235, Columbus, Ohio, USA, 2008.

[57] Danushka Bollegala. *A Study on Attributional and Relational Similarity between Word Pairs on the Web*. PhD thesis, The University of Tokyo, Japan, 2009.

[58] Sergey Brin. Extracting Patterns and Relations from the World Wide Web. In *Proceedings of the International Workshop on the World Wide Web and Databases, WebDB 1998*, pages 172–183, Valencia, Spain, 1998.

[59] Deepak Ravichandran and Eduard H. Hovy. Learning Surface Text Patterns for a Question Answering System. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, ACL 2002*, pages 41–47, Philadelphia, PA, USA, 2002.

[60] Oren Etzioni, Michael J. Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-Scale Information Extraction in KnowItAll: (Preliminary Results). In *Proceedings of the 13th International Conference on World Wide Web, WWW 2004*, pages 100–110, New York, NY, USA, 2004.

[61] Michael J. Cafarella, Doug Downey, Stephen Soderland, and Oren Etzioni. Know-

ItNow: Fast, Scalable Information Extraction from the Web. In *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, HLT-EMNLP 2005*, pages 563–570, Vancouver, British Columbia, Canada, 2005.

[62] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A Core of Semantic Knowledge. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007*, pages 697–706, Banff, Alberta, Canada, 2007.

[63] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A Large Ontology from Wikipedia and WordNet. *Journal of Web Semantics*, 6(3):203–217, 2008.

[64] Marti A. Hearst. Automatic Acquisition of Hyponyms from Large Text Corpora. In *Proceedings of the 14th International Conference on Computational Linguistics, COLING 1992*, pages 539–545, Nantes, France, 1992.

[65] Matthew Berland and Eugene Charniak. Finding Parts in Very Large Corpora. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics, ACL 1999*, pages 57–64, College Park, Maryland, USA, 1999.

[66] Roxana Girju, Adriana Badulescu, and Dan I. Moldovan. Learning Semantic Constraints for the Automatic Discovery of Part-Whole Relations. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL 2003*, Edmonton, Canada, 2003.

[67] Roxana Girju, Adriana Badulescu, and Dan I. Moldovan. Automatic Discovery of Part-Whole Relations. *Computational Linguistics*, 32(1):83–135, 2006.

[68] Haibo Li, Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. Using Graph Based Method to Improve Bootstrapping Relation Extraction. In *Proceedings of the 12th International Conference on Computational Linguistics and Intelligent Text Processing, CICLing 2011*, pages 127–138, Tokyo, Japan, 2011.

[69] Patrick Pantel and Marco Pennacchiotti. Espresso: Leveraging Generic Patterns for Automatically Harvesting Semantic Relations. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics, ACL 2006*, pages 113–120, Sydney, Australia, 2006.

[70] Eugene Agichtein and Luis Gravano. Snowball: Extracting Relations from Large Plain-Text Collections. In *Proceedings of the 5th ACM Conference on Digital Libraries, ACM DL 2000*, pages 85–94, San Antonio, TX, USA, 2000.

[71] Jun Zhu, Zaiqing Nie, Xiaojiang Liu, Bo Zhang, and Ji-Rong Wen. StatSnowball: A Statistical Approach to Extracting Entity Relationships. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009*, pages 101–110, Madrid, Spain, 2009.

[72] Sharon A. Caraballo. Automatic Construction of a Hypernym-Labeled Noun Hierarchy from Text. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics, ACL 1999*, pages 120–126, College Park, Maryland, USA, 1999.

[73] Patrick Pantel and Deepak Ravichandran. Automatically Labeling Semantic Classes. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL 2004*, pages 321–328, 2004.

[74] Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S. Weld. Open information extraction from the web. *Communications of the ACM (CACM)*, 51(12):68–74, 2008.

[75] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data.

In *Proceedings of the 18th International Conference on Machine Learning, ICML 2001*, pages 282–289, Williamstown, MA, USA, 2001.

[76] Ralph Grishman and Beth Sundheim. Message Understanding Conference - 6: A Brief History. In *Proceedings of the 16th International Conference on Computational Linguistics, COLING 1996*, pages 466–471, Copenhagen, Denmark, 1996.

[77] Krisztian Balog, Arjen P. de Vries, Pavel Serdyukov, Paul Thomas, and Thijs Westerveld. Overview of the TREC 2009 Entity Track. In *Proceedings of the 18th Text REtrieval Conference, TREC 2009*, Gaithersburg, Maryland, USA, 2009.

[78] Kumiko Tanaka-Ishii and Hiroshi Nakagawa. A Multilingual Usage Consultation Tool Based on Internet Searching: More than a Search Engine, Less than QA. In *Proceedings of the 14th international conference on World Wide Web, WWW 2005*, pages 363–371, Chiba, Japan, 2005.

[79] Kumiko Tanaka-Ishii and Yuichiro Ishii. Multilingual Phrase-Based Concordance Generation in Real-Time. *Information Retrieval*, 10(3):275–295, 2007.

[80] Tomokazu Goto, Nguyen Tuan Duc, Danushka Bollegala, and Mitsuru Ishizuka. Improving Relational Search Performance using Relational Symmetries and Predictors. *Transactions of the Japanese Society for Artificial Intelligence*, 26(6):649–656, 2011. (in Japanese).

[81] Susan T. Dumais, Todd A. Letsche, Michael L. Littman, and Thomas K. Landauer. Automatic Cross-Language Retrieval Using Latent Semantic Indexing. In *Proceedings of the AAAI Symposium on Cross-Language Text and Speech Retrieval*, pages 18–24, Stanford University, CA, USA, 1997.

[82] Lisa Ballesteros and W. Bruce Croft. Resolving Ambiguity for Cross-Language Retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1998*, pages 64–71, Melbourne, Australia, 1998.

[83] Jian-Yun Nie, Michel Simard, Pierre Isabelle, and Richard Durand. Cross-Language Information Retrieval Based on Parallel Texts and Automatic Mining of Parallel Texts from the Web. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1999*, pages 74–81, Berkeley, CA, USA, 1999.

[84] Adam L. Berger and John D. Lafferty. Information Retrieval as Statistical Translation. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1999*, pages 222–229, Berkeley, CA, USA, 1999.

[85] Ying Zhang, Phil Vines, and Justin Zobel. Chinese OOV Translation and Post-Translation Query Expansion in Chinese–English Cross-Lingual Information Retrieval. *ACM Transactions on Asian Language Information Processing (TALIP)*, 4(2):57–77, 2005.

[86] Atsushi Fujii, Masao Utiyama, Mikio Yamamoto, and Takehito Utsuro. Evaluating Effects of Machine Translation Accuracy on Cross-Lingual Patent Retrieval. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009*, pages 674–675, Boston, MA, USA, 2009.

[87] Ruth Sperer and Douglas W. Oard. Structured Translation for Cross-Language Information Retrieval. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2000*, pages 120–127, Athens, Greece, 2000.

[88] Ari Pirkola. The Effects of Query Structure and Dictionary Setups in Dictionary-Based Cross-Language Information Retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information*

Retrieval, SIGIR 1998, pages 55–63, Melbourne, Australia, 1998.

[89] George W. Furnas, Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, Richard A. Harshman, Lynn A. Streeter, and Karen E. Lochbaum. Information Retrieval using a Singular Value Decomposition Model of Latent Semantic Structure. In *Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1988*, pages 465–480, Grenoble, France, 1988.

[90] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science and Technology (JASIS)*, 41(6):391–407, 1990.

[91] Richard Sproat, Tao Tao, and ChengXiang Zhai. Named Entity Transliteration with Comparable Corpora. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, COLING-ACL 2006*, pages 73–80, Sydney, Australia, 2006.

[92] Nasreen Abdul Jaleel and Leah S. Larkey. Statistical Transliteration for English-Arabic Cross Language Information Retrieval. In *Proceedings of the 12th ACM International Conference on Information and Knowledge Management, CIKM 2003*, pages 139–146, New Orleans, Louisiana, USA, 2003.

[93] Kevin Knight and Jonathan Graehl. Machine Transliteration. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics, ACL 1997*, pages 128–135, 1997.

[94] Peter Prettenhofer and Benno Stein. Cross-Language Text Classification using Structural Correspondence Learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, ACL 2010*, pages 1118–1127, Uppsala, Sweden, 2010.

[95] Alfio Massimiliano Gliozzo and Carlo Strapparava. Exploiting Comparable Corpora and Bilingual Dictionaries for Cross-Language Text Categorization. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics, COLING/ACL 2006*, Sydney, Australia, 2006.

[96] Dmitry Davidov and Ari Rappoport. Automated Translation of Semantic Relationships. In *Proceedings of the 23rd International Conference on Computational Linguistics, COLING 2010*, pages 241–249, Beijing, China, 2010.

[97] Gerard Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM (CACM)*, 18(11):613–620, 1975.

[98] M. E. Maron and J. L. Kuhns. On Relevance, Probabilistic Indexing and Information Retrieval. *Journal of the ACM*, 7(3):216–244, 1960.

[99] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the 7th International Conference on World Wide Web, WWW 1998*, pages 107–117, Brisbane, Australia, 1998.

[100] William B. Cavnar and John M. Trenkle. N-Gram-Based Text Categorization. In *Proceedings of the 3rd Annual Symposium on Document Analysis and Information Retrieval, SDAIR 1994*, pages 161–175, Las Vegas, USA, 1994.

[101] Hidayet Takci and Ibrahim Sogukpinar. Centroid-Based Language Identification Using Letter Feature Set. In *Proceedings of the 5th International Conference on Computational Linguistics and Intelligent Text Processing, CICLing 2004*, pages 640–648, Seoul, Korea, 2004.

[102] Paul McNamee. Language Identification: A Solved Problem Suitable for Undergraduate Instruction. *Journal of Computing Sciences in Colleges*, 20(3):94–101, 2005.

[103] Yusuke Shinyama and Satoshi Sekine. Preemptive Information Extraction using

Unrestricted Relation Discovery. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, HLT-NAACL 2006*, pages 304 – 311, New York, USA, 2006.

[104] Zellig Harris. Distributional Structure. *Word*, 10(23):146–162, 1954.

[105] Dekang Lin and Patrick Pantel. DIRT – Discovery of Inference Rules from Text. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2001*, pages 323–328, San Francisco, California, USA, 2001.

[106] R. Sibson. SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method. *Computer Journal*, 16(1):30–34, 1973.

[107] D. Defays. An Efficient Algorithm for a Complete Link Method. *Computer Journal*, 20(4):364–366, 1977.

[108] Sho Kawasaki, Nguyen Tuan Duc, Danushka Bollegala, and Mitsuru Ishizuka. Accurate Relation Name Extraction from Entity Pair Set using the Web. In *Proceedings of the 73rd National Convention of the Information Processing Society of Japan, IPSJ 2011*, Tokyo, Japan, 2011. (in Japanese).

[109] Razvan C. Bunescu and Raymond J. Mooney. Learning to Extract Relations from the Web using Minimal Supervision. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, ACL 2007*, pages 576–583, Prague, Czech Republic, 2007.

[110] Dragomir R. Radev, Hong Qi, Harris Wu, and Weiguo Fan. Evaluating Web-based Question Answering Systems. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation, LREC 2002*, pages 1153–1156, Canary Islands, Spain, 2002.

[111] Chirag Shah and W. Bruce Croft. Evaluating High Accuracy Retrieval Techniques. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2004*, pages 2–9, Sheffield, UK, 2004.

[112] Marc Najork, Hugo Zaragoza, and Michael J. Taylor. Hits on the Web: How Does It Compare? In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2007*, pages 471–478, Amsterdam, The Netherlands, 2007.

[113] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, ACL 2007*, pages 177–180, Prague, Czech Republic, 2007.

[114] Thomas K. Landauer and Susan T. Dumais. Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction and Representation of Knowledge. *Psychological Review*, 104:211–240, 1997.

[115] Gene Golub and William Kahan. Calculating the Singular Values and Pseudo-Inverse of a Matrix. *Journal of the Society for Industrial and Applied Mathematics: Series B, Numerical Analysis*, 2(2):205–224, 1965.

[116] Graham Neubig. The Kyoto Free Translation Task. http://www.phontron.com/kftt, 2011.

[117] Gianluca Demartini, Arjen P. de Vries, Tereza Iofciu, and Jianhan Zhu. Overview of the INEX 2008 Entity Ranking Track. In *Proceedings of the 7th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2008*, pages 243–252, Dagstuhl Castle, Germany, 2008.

[118] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on

Large Clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI 2004)*, pages 137–150, San Francisco, CA, USA, 2004.

[119] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert Gruber. Bigtable: A Distributed Storage System for Structured Data. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI 2006)*, pages 205–218, Seattle, WA, USA, 2006.

[120] Emine Yilmaz, Evangelos Kanoulas, and Javed A. Aslam. A Simple and Efficient Sampling Method for Estimating AP and NDCG. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008*, pages 603–610, Singapore, 2008.