Linguistic and Semantic Passage Retrieval Strategies for Question Answering

Matthew W. Bilotti

CMU-LTI-09-019

December 4, 2009

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Eric Nyberg, Chair
Jamie Callan
Jaime Carbonell
Eric Brown, IBM T. J. Watson Research Center

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This work was supported in part by ARDA/DTO Advanced Question Answering for Intelligence (AQUAINT) program award number NBCHC040164.

Abstract

Question Answering (QA) is the task of searching a large text collection for specific answers to questions posed in natural language. Though they often have access to rich linguistic and semantic analyses of their input questions, QA systems often rely on off-the-shelf bag-of-words Information Retrieval (IR) solutions to retrieve passages matching a set of terms extracted from the question.

There is a fundamental disconnect between the capabilities of the bag-of-words retrieval model and the retrieval needs of the QA system. Bag-of-words IR retrieves documents matching a query, but the QA system really needs documents that contain answers. Through question analysis, the QA system has compiled a sophisticated *information need* representation for what constitutes an answer to the question. This representation is composed of a set of linguistic and semantic constraints satisfied by answer-bearing passages. Unfortunately, off-the-shelf IR libraries commonly used in QA systems can not, in general, check these types of constraints at query-time. Poor quality retrieval can cause a QA system to fail if no answer-bearing text is retrieved, if it is not ranked highly enough, or if it is outranked or overwhelmed by false positives, text that matches the query well, yet supports a wrong answer.

This thesis proposes two linguistic and semantic passage retrieval methods for QA, one based on structured retrieval and the other on rank-learning techniques. In addition, a methodology is proposed for mapping annotated text consisting of labeled spans and typed relations between them into an annotation graph representation. The annotation graph supports query-time linguistic and semantic constraint-checking, and serves as a unifying formalism for the QA system's information need and for retrieved passages. The proposed methods rely only on the relatively weak assumption that the QA system's information need can be represented as an annotation graph. The two approaches are shown to retrieve more answer-bearing text, more highly ranked, compared to a bag-of-words baseline for two different QA tasks. Linguistic and semantic passage retrieval methods are also shown to improve end-to-end QA system accuracy and answer MRR.

Acknowledgments

I would like to thank my advisor, Eric Nyberg, for all he has done, and the rest of my committee, Jamie Callan, Jaime Carbonell and Eric Brown, for helping to guide this work.

I gratefully acknowledge the support for this work provided by ARDA/DTO Advanced Question Answering for Intelligence (AQUAINT) program award number NBCHC040164.

Special thanks to Paul Ogilvie and Jon Elsas, with whom I have co-authored papers; to Andy Schlaikjer, Justin Betteridge and Le Zhao for many helpful discussions; and to Roberto Pensotti for his help in preparing the Italian data used in this thesis.

Most importantly, I would like to thank my wife and family for their unwavering support of this effort.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since the time of the earliest digital computers, the amount of information available in the form of electronic text has grown exponentially. Issues of scale inherent in the management of large collections of information, such as the World Wide Web, complicate efforts to provide access and to maintain an organizational structure. Today's ubiquitous search engines are the cutting edge of research in Information Retrieval (IR), a field born in the 1950s out of the need to apply library science techniques to collections of electronic text.

At the same time, computational linguists were harnessing the increasing processing power of modern computers available in recent decades to build tools to support machine understanding of human languages. As the capabilities of Natural Language Processing (NLP) technologies grew, researchers began to experiment with combining them with already-mature IR technologies, giving rise to a new information management paradigm known as Question Answering (QA). Because of the shift in focus toward retrieving linguistic units representing answers, rather than text units such as passages or documents, QA was believed to be what users really wanted on some level [63].

1.1 What is Question Answering?

At the highest level, a Question Answering (QA) system provides specific answers to questions posed by users in human language. For example, a user interested in professional tennis may ask a QA system, *Who beat Federer?*, to which the system might reply *Nadal*, *Murray*, *Djokovic*, etc. In terms of input, a QA system supports natural language questions as opposed to queries formulated in a particular search engine's query syntax, such as `beat AND federer`. With respect to output, QA systems retrieve answers, concise pieces of text that respond to the input question in a linguistically justifiable way. In contrast, search engines return ranked lists of documents, which the user must then read in order to locate the information he or she was looking for.

Figure 1.1 shows a block diagram of a QA system, consisting of NLP technology encapsulating an IR component. At the highest level of abstraction, a QA system can be thought of as a pipeline consisting of an IR component bookended by NLP components [22]. The first module in the pipeline, Question Analysis, analyzes a natural language question input to system, producing an internal representation for the system's *information need*, which is a linguistic and semantic description of the answer to the question. The information need is passed to the embedded IR component, which uses it to retrieve passages likely to be answer-bearing. The final NLP component, Answer Generation, uses the information need to pinpoint answers among results retrieved by the IR module.

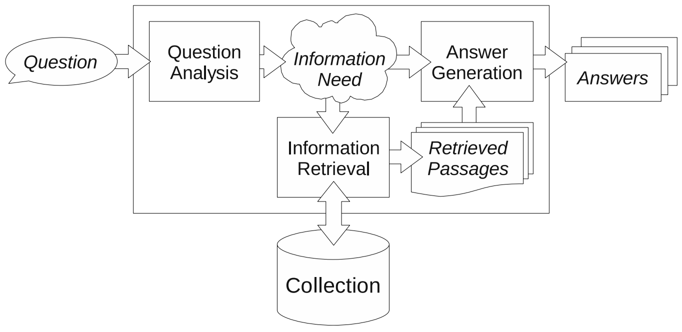

Figure 1.1: A QA system, consisting of an IR system encapsulated by NLP components: front-end Question Analysis and back-end Answer Generation.

1.2 The Role of Retrieval in QA

Much of the primary early research in QA focused on developing robust Question Analysis and Answer Generation modules, which did not exist before and without which natural language questions could not be understood, nor answers be located among retrieved text. In contrast, relatively little attention was paid to the IR component, which was considered more mature technology. The builders of early QA systems often used off-the-shelf IR tools and incorporated them into their systems with relatively little tuning [24].

Questions can not be answered without NLP, and if infinite processing power were available, the straightforward approach to QA would be to analyze the entire corpus, comparing each passage against the question to find the best answer. Real-world limitations make it impossible to use this approach, so most QA systems use an embedded IR component to narrow the search space for the answer to the question. The role of retrieval in QA is to function as a fast, but coarse, first pass filter to exclude from the search passages not likely to be answer-bearing [7]. The passages selected by the IR component would then be subjected to the extensive NLP analysis required to determine whether they contain answers to the question.

After the NLP components of the system, Question Analysis and Answer Generation, reached a certain level of maturity, QA system developers began to observe that certain system failure modes were attributable to the quality of the IR component [35], and that some part of the QA problem was in fact an IR problem, with different requirements than the traditional *ad hoc* retrieval task [7]. If the IR module fails to retrieve a passage containing the correct answer to the question, then the downstream Answer Generation component can not recover, and the system will reply with an incorrect answer. End-to-end QA system accuracy is also sensitive to the quality of the ranking of passages coming out of the IR component. If the answer-bearing passages are outnumbered by false positives or are ranked so low that they can not be considered, the likelihood that the system will return the correct answer is reduced.

These observed phenomena result from an underlying problem of mismatch of representation between the IR and NLP components of the QA system. For each input question, the Question Analysis module builds a sophisticated representation of the system's information need, which is a prescription for the answer to the input question. This information need representation is composed of NLP primitives such as linguistic and semantic constraints between question keyterms. The off-the-shelf IR components first used for QA could not check these constraints at query time, supporting only surface keyterm relationships, such as proximity, ordering and frequency.

Because the primitive elements of a QA system's information need representation are not in general directly queryable by an off-the-shelf IR component, the system must cast its information need into the weaker query representation supported by the IR system. This cast results in the relaxation of the linguistic and semantic constraints in the information need, and in the retrieval of false positives, those passages matching the IR query but not satisfying the higher-level constraints in the information need. To identify the answer-bearing passages among the false positives, the Answer Generation module must perform a potentially costly NLP analysis on each retrieved passage, and compare it against the information need.

1.3 Tailoring IR to the Needs of QA Systems

If QA is ever to become fast and accurate enough to compete with the Google information access paradigm, which emphasizes fast response to queries over accurate constraint checking at the level of the information need, researchers must tailor IR solutions for the QA task. Though they are also important to a successful QA system, it is not sufficient to focus solely on the NLP components of the system.

Early work on improving IR for QA began by considering the IR system as a black box. Different methods of expressing the question as a query were explored, including different query operators, query expansion and term weighting schemes, often by intuition [7, 21]. Variations in document representation and indexing strategies were also tried, including indexing meta-information on terms [11] and index-time passage segmentation [59]. Many researchers grew to distrust the rankings coming out of their embedded IR systems, and attempted to exercise external control over the ranking by querying within feedback loops [44] or with sequences of successively relaxed queries [26], or by using pseudo-relevance feedback [36] or explicit post-retrieval passage extraction and ranking modules [57].

In contrast, the work in this thesis goes inside the black box to adapt IR solutions to the needs of a QA system. The goal is to get the IR component to shoulder more of the burden of the QA task, by shrinking the search space as tightly as possible around the region of relevant text without an appreciable increase in false negatives. Conceptually, the IR system's query representation needs to be evolved closer to the QA system's information need representation, to enable query-time checking of the linguistic and semantic constraints that determine whether a passage is answer-bearing or not. This is the vision that guides the research in linguistic and semantic passage retrieval methods for QA systems documented in this thesis.

To illustrate the contrast between surface-level constraints queryable by many off-the-shelf IR systems and the deeper linguistic and semantic constraints necessary to determine whether text is answer-bearing, we return to the tennis fan's example question, *Who beat Federer?* There are two keyterms that can be extracted from the question, *beat* and *federer*, both of which are required, but *federer* is more important.

The wh-word becomes a placeholder for phrases identified as person names in the text. There is an implicit proximity relationship that should hold among the keyterms and the placeholder, but whether the ordering is important is unclear. In English, word ordering is strongly predictive of semantics, but linguistic constructions such as passive voice, topicalization and relative clause movement can defeat the intent of the ordering constraint.

Table 1.1: Passages retrieved for the question, *Who beat Federer?*; passages 1 and 2, shown in bold face, are answer-bearing.

	Passage	*beat*	*federer*	Proximity	Ordering
1	***Nadal beat Federer.***	●	●	●	●
2	***Federer was defeated by Nadal.***		●	●	
3	*Federer beat Safin.*	●	●	●	
4	*Safin was beaten by Federer.*	●	●	●	●

Table 1.1 shows four passages that a QA system's IR component might retrieve for the question, *Who beat Federer?* Passages 1 and 2 are answer-bearing, but passages 3 and 4 describe matches in which Federer prevailed, which are certainly more common than those in which he was defeated. Note that a bag-of-words retrieval model can not distinguish among passages 1, 2 and 4, because they both contain the keyterms *federer* and *beat*, assuming that the various morphological forms are conflated. A retrieval model that enforces the ordering constraints can not distinguish between 1 and 4, or 2 and 3.

For the question, *Who beat Federer?*, the relationship among the keyterms and placeholder that is the most important to the QA system is not proximity or ordering, but rather a semantic constraint that specifies that *federer* is the *patient* of the *beat* event, that is, the player who is beaten. The *agent* of the event, or the winning player, is represented by the placeholder that matches extents in the text that have been identified as person names. In Table 1.1, the answer-bearing passages (1 and 2) satisfy these semantic constraints, and no others do.

Pondering the example, several research questions come to mind:

1. Can an IR system be built that allows for query-time constraint checking of the linguistic and semantic constraints in the QA system's information need?

2. What kinds of index structures and retrieval models would be useful?

3. Would such an approach improve passage retrieval quality, where *better quality* is defined as more answer-bearing passages, more highly-ranked, and fewer false positives?

4. Would such an approach improve end-to-end QA system quality, where *better quality* is defined to mean that the top-ranked answer is correct more often, and the average rank of the highest-ranked correct answer is closer to the top?

1.4 Hypothesis

For a given QA system, there exists a linguistic and semantic passage retrieval technique that outperforms a non-linguistic and semantic baseline, providing measurably *better quality* passage retrieval.

Furthermore, in a QA system utilizing a linguistic and semantic passage retrieval technique to obtain *better quality* passage retrieval, the system will exhibit *better quality* end-to-end performance.

1.5 Contributions of this Thesis

This thesis provides a general methodology for mapping any text annotation scheme based on labeling spans of text, and relating those spans with typed relations, into a unifying representation called the *annotation graph*. This formalism is used for representing both QA system information needs and passages retrieved by a QA system's embedded IR module, and enables linguistic and semantic comparison between the two.

In addition, this thesis proposes two specific linguistic and semantic passage retrieval strategies for QA systems, one based on structured retrieval methods, and the other based on rank-learning techniques. Both methods support query-time checking of the linguistic and semantic constraints expressed in an annotation graph. These methods are generalized with respect to the specific types of constraints used, as well as features of the text, such as language and domain. The only requirement that a QA system must meet to be able to integrate either of these passage retrieval methods is the relatively weak requirement that the system's internal linguistic and semantic representation can be reduced to an annotation graph.

This thesis provides concrete solutions to an issue that continually challenges the

QA research community; namely, the issue of poor quality retrieval and its detrimental effect on QA system performance. This thesis provides QA systems access to linguistic and semantic passage retrieval methods that can be used to improve retrieval quality, thereby lessening the burden on downstream Answer Generation, and can be made to operate fast, in order to take a step toward the eventual goal of real-time QA. The results in this thesis suggest avenues for future research that can take the field even further.

This work also has implications for the learning-to-rank field. In 2007, Croft articulated a vision for the rank learning research community involving "different types of information needs, more linguistic features, more integration with structured data, [and] different applications" [13]. In applying rank-learning techniques to the task of linguistic and semantic passage retrieval for QA, this thesis represents a step toward that broad vision.

1.6 Outline of this Thesis

The remaining chapters of this thesis are organized as follows:

- Chapter 2 examines the interaction between Question Answering and Information Retrieval, beginning with historical perspectives, summarizing the latest approaches, and highlighting some of the issues that the methods proposed in this thesis are intended to address.

- In Chapter 3, the concept of an *annotation graph* is introduced as a generalized representation for a QA system's information need. The chapter describes how a variety of representations can be reduced to an annotation graph, given the appropriate type system.

- Chapter 4 describes resources used in the experiments in the later chapters of this thesis.

- The first of the linguistic and semantic passage retrieval strategies proposed in this thesis, based on mapping the task of annotation graph retrieval into a structured retrieval problem, is outlined in Chapter 5.

- Chapter 6 proposes another strategy for ranking text using the annotation graph representation and rank-learning methods.

- In Chapter 7, the rank-learning passage retrieval strategy introduced in Chapter 6 is applied to a second, very different, problem instance. The effect of passage retrieval quality on end-to-end QA system performance is also discussed in this chapter.
- The contributions of this thesis are summarized in Chapter 8.

Chapter 2

Background and Related Work

This chapter studies historical and contemporary work on Question Answering (QA), focusing on its relationship to Information Retrieval (IR). This chapter is not an exhaustive survey of prior and related work, but instead seeks to analyze alternative approaches to the task of retrieval for QA and to identify issues that are addressed by subsequent chapters in this thesis.

2.1 Answering Questions over Structured and Unstructured Data

Work on early Question Answering (QA) systems began in the early 1960s, during which two separate, competing threads of research emerged. One school of thought grew out of research in natural language interfaces for database systems, such as Baseball [20] and Lunar [65], each of which used shallow parsing techniques and dictionary-based approaches to translate user questions into database queries.

This approach matured into START[1], the first publicly-available QA system on the web, which uses hand-edited question matching schemata and generation templates to provide access to a structured knowledge base. Wolfram Alpha is a similar project, but does not claim to offer full-blown QA capabilities, instead opting for a simpler query syntax [2]. The primary advantage of these approaches is that their accuracy is very high, and that they support interesting modes of information ag-

[1]See: `http://start.csail.mit.edu/`

[2]See: `http://www.wolframalpha.com/`

gregation, such as joins against the backing database. The primary disadvantage is that coverage is low and the burden of knowledge engineering is high.

The other research thread explored Question Answering over unstructured text. The ORACLE system [45] worked by reformulating input questions into declarative statements, then scanning a corpus for an exact match containing the same terms in the same order. PROTOSYNTHEX [54] adopted a more sophisticated matching approach, using morphological normalization and dependency graph analysis to score text matching the question.

More recently, Wendlandt and Driscoll proposed labeling documents with features indicating the types of questions they could answer, such as *amount* and *duration* for *how much* and *how long* questions [64]. They showed how these categories could be combined with keyterm information to retrieve documents likely to contain answers to questions, but do not explain how to enforce that the document contains the answer to the specific question described by the keyterms.

The MURAX system is an open-domain QA system that answers questions over a corpus of encyclopedia articles [28]. It is one of the earliest examples of the pipelined architecture widely considered to be the standard approach to the QA problem [22]. MURAX uses part-of-speech tagging and shallow chunking to analyze the question, to formulate queries for a Boolean retrieval system that ranks by keyterm proximity, and to extract answers in the form of noun phrases from retrieved text. MURAX was a successful system that many later QA systems were modeled after, but it later became clear, when similar techniques were applied to newswire text, that the informative and well-curated nature of the encyclopedia text was responsible for much of the system's success.

2.2 Open-Domain QA Evaluations at TREC and CLEF

Between 1999 and 2007, research and development in open-domain QA was fueled by interest from the U.S. National Institute of Standards and Technology (NIST), which organized large-scale evaluations for QA systems as part of its annual Text REtrieval Conferences[3] (TRECs). The Cross-Language Evaluation Forum[4] (CLEF; pronounced "clay", as in the French word for "key"), organized by the Istituto di Scienza

[3]See: `http://trec.nist.gov`
[4]See: `http://www.clef-campaign.org/`

e Tecnologie dell'Informazione of the Italian Consiglio Nazionale delle Ricerche, is often considered the European counterpart to TREC, and has organized comparative evaluations in monolingual and cross-lingual QA since 2003. Each year, the task became more difficult by including new question types, such as definition and list questions, requiring tighter answer bounds and better support for answers. This section discusses some of the notable advances in the state-of-the-art of QA made by various participants in TREC and CLEF evaluations.

The predictive annotation approach [51] involves pre-processing the collection with named entity recognition, then indexing the location of those entities within the text. This approach inherits philosophically from Wendlandt and Driscoll's method [64], but the positional information in the index allows the system to enforce proximity relationships between keyterms and entities at query-time, ensuring that the entities in the text answer the question described by the keyterms. In terms of storing semantic information in the index, the predictive annotation approach inspires, in part, the structured retrieval approach to linguistic and semantic passage retrieval proposed in Chapter 5.

There has also been a great deal of work on query expansion among TREC participants who observed that there are relatively few keyterms available for extraction from the question. Hovy, *et. al.* [23] report improved retrieval quality using a form of controlled synonymy based on WordNet [17]. Greenwood has explored the use of WordNet pertainyms for query expansion, boosting performance on questions containing a location name [21]. Paşca and Harabagiu [44] present an approach in which queries are expanded using morphology, lexical derivations and semantic equivalents, a kind of highly-controlled synonymy based on hand-maintained resources. Indiscriminate query expansion and pseudo-relevance feedback were shown to be poor retrieval strategies for QA [37, 31].

Many QA systems participating in TREC over the years utilized a two-stage retrieval approach consisting of a document retrieval stage, followed by a sliding-window passage scoring algorithm [9]. Tellex, *et. al.*, survey a variety of published passage scoring functions used for QA, and find that those that favor keyterm proximity provide the best end-to-end system performance [57]. Monz proposes a density-based passage scoring function that incorporates a document-level similarity score [38]. The main disadvantage of the two-stage approach is that passage ranking quality depends on a document retrieval step able to reliably retrieve documents containing relevant passages.

Cui, *et al.* [14] propose a method of passage scoring based on dependency graph similarity between the passage and the question using the Minipar dependency parser [30]. The method considers paths through the dependency graph between keyterm pairs, and uses a version of IBM Translation Model 1 to estimate the likelihood that the path between a pair of passage keyterms represents the same relationship that exists between that pair of keyterms in the question. The method was shown to be effective against a weak baseline, and appears to require significant amounts of training data. In Section 6.8, the rank-learning approach proposed in this thesis is compared against the Cui approach.

In 2005, TREC offered a relationship question task, which emphasized retrieval of short passages describing the relationships between two entities, be they familial, financial, political, etc. Narayanan and Harabagiu [40] proposed a method of answering these questions based on probabilistic inference over a rich semantics based in part on the ASSERT shallow semantic parser [49]. Subsequently, the JAVELIN system attempted to improve on this work by introducing a retrieval strategy in which the same type of semantic relationships between keyterms were represented in the index and made available for query-time checking [42, 41]. The JAVELIN retrieval approach for relationship questions was among the first to integrate semantics directly into the retrieval process, and is a direct ancestor of the structured retrieval method discussed in Chapter 5.

Tiedemann proposed a similar approach for answering CLEF questions in Dutch, decomposing the output of a dependency parser into components that can be indexed [58]. The approach encodes specific pairwise relationships between keyterms as artificial index terms, which can then be queried upon using standard retrieval models. Unlike the structured retrieval approach described in Chapter 5, this method is not capable of checking relations among groups of three keyterms or larger, because they are not stored in the index.

2.3 Recent Work in Linguistic and Semantic IR for QA

Pizzato and Mollá [47] investigate indexing and retrieval of text annotated using a proprietary, lightweight representation based on semantic role labeling. This method is similar to the JAVELIN approach to relationship questions [42, 41] and the structured retrieval approach to linguistic and semantic passage retrieval for QA proposed in Chapter 5, but it uses a vector-space retrieval model instead. The vector representation consists of weighted term counts, as in the classic vector-space model, but also counts of terms playing specific semantic roles, as well as counts of pairwise semantic relationships between terms. It was shown that the method is effective, but sensitive to gaps in coverage by the semantic role labeling tool, which is, of course, true of any system relying on text annotation.

Verberne, *et. al.*, surveyed a variety of learning-to-rank paradigms for their applicability to the QA task, concluding that a pairwise approach using Support Vector Regression was the most appropriate strategy [61]. They were among the first to use linguistically-motivated features in a rank-learning context. Their feature set was based largely on term overlap between the question and a passage to be scored within different syntactic categories, such as verb and subject. A second set of features measured these same overlaps after expansion through WordNet synsets [17]. This approach is similar in spirit to the rank-learning approach proposed in Chapter 6, which takes the concept to the next level by using a feature set based on decomposition of the QA system's information need into atomic linguistic and semantic constraints.

2.4 Evaluation of IR for QA

The evaluation methodology for the linguistic and semantic passage retrieval methods proposed in this thesis differs in two key ways from that which is shared by the bulk of the related approaches described in this chapter. The first fundamental difference is in the standard used to determine whether a retrieved passage is answer-bearing or relevant to a given question. Much of the prior work uses a lenient evaluation standard that considers a passage relevant if it matches a hand-edited answer pattern. Unfortunately, this standard tends to overestimate relevance. Consider the example passage *Fenway Park opened in 1912*, which, if retrieved in response to the question *When did the Titanic sink?* would be considered relevant because it matches the answer pattern, despite clearly not answering the question. The experiments in this thesis consider a passage relevant to a question if and only if it is contained in a test collection constructed by a human assessor charged with judging passages as to whether they contain and support the answer to the question. See Chapter 4 for more information on the test collections used in this thesis.

The work presented in this thesis not only uses a different standard for ground truth in evaluation, but also a different metric for calculating the quality of a ranked list of passages. Many prior approaches rely on a metric known as Reciprocal Rank, in which a ranking is measured using the reciprocal of the one-based rank of the top-ranked passage that is relevant. A ranking in which the first passage is relevant would receive the maximum score of 1.0, while a ranking in which the first two passages are not relevant, but the third passage is relevant, would receive a score of 0.3333. This work uses Average Precision, which scales not only with the proportion of the universe of known relevant passages that are retrieved, but with the rank of each of these passages. Average Precision is maximized when all relevant passages are retrieved at the top of the ranked list, and when there are no non-relevant passages ranked ahead of any relevant passages. Because Average Precision does a better job of capturing the density of relevant passages at the top of the ranking, it is the preferred metric for the experiments in this thesis.

2.5 Related Tasks

The task of Recognizing Textual Entailment (RTE) was proposed as a general model for semantic inference and for abstracting over semantic variation in text [15]. Given a snippet of text, the task challenges systems to determine whether a second text fragment, known as the hypothesis, follows from the first. If so, the hypothesis is said to be entailed by the first text. As a simple example, the hypothesis *John loves Mary* is entailed by texts such as *John is enamored with Mary*, *John and Mary were married*, and perhaps even *John bought Mary flowers.*

QA and RTE are related in that a piece of text that answers a question entails a hypothesis formulated by inserting the answer to the question into a declarative statement derived from that question. The linguistic and semantic passage retrieval methods proposed for the QA task in this thesis are based on computing the essential similarity between an information need and a piece of text in terms of partial satisfaction of linguistic and semantic constraints. These methods would apply naturally to the task of RTE, and would fall somewhere in the middle of the continuum of approaches applicable to the task, which is anchored by full-blown logical inference and theorem proving on one end and by superficial methods such as word overlap on the other end.

Chapter 3

Representing the Information Need as an Annotation Graph

This chapter gives the formal definition of an *annotation graph*, a generalized, unifying representation for the information need of a Question Answering (QA) system, as well as for retrieved passages. The *information need* is a linguistic and semantic prescription for the answer to a question posed to a QA system. The information need is the output of the system's Question Analysis module, which is responsible for all of the front-end Natural Language Processing (NLP) required to understand the input question. This same representation serves as the input to the system's downstream modules; the embedded Information Retrieval (IR) component uses it to perform passage retrieval, and the Answer Generation module uses it to locate answers among the retrieved passages.

All schemes of text annotation that consist of typed text spans and typed binary relations between pairs of spans can be reduced to the annotation graph representation described here. This includes broad classes of named entity recognition, syntactic parsing and semantic analysis commonly used in QA systems. The passage retrieval techniques proposed in subsequent chapters are applicable to all QA systems whose internal information needs can be represented as annotation graphs.

3.1 What is an Annotation Graph?

An *annotation graph* is the unifying linguistic and semantic representation used for information needs as well as retrieved passages. Formally, an annotation graph G is defined as a tuple consisting of three elements: a set of elements E (vertices), a set of relations R (edges), and a construct to be defined presently called the type system, given by T.

$$G = (E = \{e_1, \ldots, e_{|E|}\}, R = \{r_1, \ldots, r_{|R|}\}, T)$$

Type systems are vocabularies of the types of elements and the types of relations that can be instantiated in a particular annotation graph. Formally, a type system is a tuple that consists of two sets, Te and Tr, which define all valid types for elements and relations, respectively.

$$T = (Te = \{te_1, \ldots, te_{|Te|}\}, Tr = \{tr_1, \ldots, tr_{|Tr|}\})$$

See Section 3.2 for a concrete example of a relatively simple, text-based type system. Subsequent chapters of this thesis will define richer type systems as a part of their treatment of linguistic and semantic strategies for passage retrieval for QA systems. All type systems mentioned in this thesis are collected in Appendix A for easy reference.

3.1.1 Elements

In a type system, an element type te_i is defined as a name and an optional parent pointer. If the parent is not null, it must point to another element type defined in the same type system. This is the mechanism whereby single inheritance for element types is implemented in the type system. If type t_c has parent t_p, an instance of type t_c can participate in any relation defined for type t_p. Note that the parent pointer used to implement the inheritance mechanism in the type system is not a relation or a relation type that can be instantiated on a per-graph basis.

$$te_i = (name, parent \vee \emptyset)$$

In an annotation graph, each element e_i is defined as an instance of an element type defined in the type system:

$$e_i = (te); te \in Te$$

3.1.2 Relations

A relation type in a type system tr_i declares an asymmetric relation that can be defined to hold between an instance of the domain element type, or any type derived from it, and an instance of the range element type, or any type derived from that type.

$$tr_i = (name,\, te_d,\, te_r);\; te_d,\, te_r \in Te$$

In an annotation graph, each relation r_i is defined by its relation type as given in the type system, an element of the domain type and an element of the range type:

$$r_i = (tr,\, e_d,\, e_r);\; tr \in Tr;\; e_d,\, e_r \in E;\; e_d = (te_d);\; e_r = (te_r)$$

3.2 The Annotation Graph as a Model for Text

The annotation graph is general enough to represent a wide variety of data, but the remainder of this thesis concerns itself with annotation graphs that represent text. In the most general sense, text refers to a grouping of primitive elements called terms. This section describes several techniques for modeling text as annotation graphs that are common to the experiments described in later chapters.

When representing a collection of passages as annotation graphs, the passage graphs share the same type system, and this type system defines one element for every term that occurs in at least one of the passages. The common type system also must define, at a minimum, an element type *keyterm* as a supertype for each of the individual term element types. For a minimal model for the structure of text, it is necessary to define an element type, such as `sentence`, and a single relation type, *enclosure*, which is defined to hold between a `sentence` and a *keyterm*. The formal definition of this type system, T_{bow}, which models a bag-of-words representation of text, can be found in Section A.1.

Note that the element type `sentence` is given in monospace type. Throughout this thesis, it is a matter of convention that leaf element types representing concrete *annotations* are distinguished using monospace type. With respect to text, an annotation is defined as a text field, which is a contiguous sequence of *keyterms*, to which some special semantics is attached. A group of *keyterms* participating in an *enclosure* relation with an instance of specific annotation type is “tagged” with the semantics associated with that type. The semantics of the `sentence` annotation type is quite simple; *enclosure* of a *keyterm* within the `sentence` simply means that the `sentence` contains that *keyterm*.

Additional element and relation types can be added to the type system to model other aspects of the text. It is possible, for example, to represent the order of *keyterms* in the `sentence`. To model these surface patterns, a relation type called *precedence* can be defined to hold between pairs of *keyterms*. Adding this relation type allows passages containing the same *keyterms* in different orders, which would map to identical annotation graphs under T_{bow}, to be distinguished at the level of the annotation graph representation. Section A.2 gives the formal definition for this type system, denoted T_{surf}. T_{surf} contains all of the element and relation type definitions in T_{bow}, so it can be written that $T_{surf} \supset T_{bow}$.

3.3 Question Analysis and the Information Need

In a QA system, the Question Analysis module is responsible for the front-end NLP analysis required to distill the input question into an information need representation under a given type system. Consider the example question, *Who beat Federer?* Figure 3.1 shows what the information need representation would look like under T_{bow} and T_{surf}.

$$G_{in,\,bow} = (E_{bow},\, R_{bow},\, T_{bow}) \qquad G_{in,\,surf} = (E_{surf},\, R_{surf},\, T_{surf})$$

$$E_{bow} = \left\{ \begin{array}{l} e_1 = (\texttt{sentence}), \\ e_2 = (\text{beat}), \\ e_3 = (\text{federer}) \end{array} \right\} \qquad E_{surf} = \left\{ \begin{array}{l} e_4 = (\texttt{sentence}), \\ e_5 = (\text{beat}), \\ e_6 = (\text{federer}) \end{array} \right\}$$

$$R_{bow} = \left\{ \begin{array}{l} (\mathit{enclosure},\, e_1,\, e_2), \\ (\mathit{enclosure},\, e_1,\, e_3) \end{array} \right\} \qquad R_{surf} = \left\{ \begin{array}{l} (\mathit{enclosure},\, e_4,\, e_5), \\ (\mathit{enclosure},\, e_4,\, e_6), \\ (\mathit{precedence},\, e_5,\, e_6) \end{array} \right\}$$

Figure 3.1: Information needs corresponding to the input question *Who beat Federer?* expressed under T_{bow}, as defined in Section A.1, and T_{surf}, as defined in Section A.2. The difference is that $G_{in,\,surf}$ models the *precedence* constraint between the *keyterms*.

The type systems defined thus far do not define the types of linguistic and semantic constraints a QA system might need to answer the question, *Who beat Federer?* Consider the following passages, shown in Table 3.1 that might be retrieved for this question:

Table 3.1: Passages retrieved for the question, *Who beat Federer?*; passages 1 and 2, shown in bold face, are answer-bearing. The three right-hand columns indicate whether the corresponding annotation graphs under T_{surf} satisfy the constraints in the information need $G_{in,\,surf}$, which is shown in Figure 3.1.

	Passage	*enclosure* `sentence` *beat*	*enclosure* `sentence` *federer*	*precedence* *beat* *federer*
1	***Nadal beat Federer.***	●	●	●
2	***Federer was defeated by Nadal.***		●	
3	*Federer beat Safin.*	●	●	
4	*Safin was beaten by Federer.*	●	●	●

The essential relationship between *beat* and *federer* is not well-modeled by mutual *enclosure* in a `sentence`; the counter-example is passage 2, which does not contain the term *beat* or a morphological variant thereof. Nor is the *precedence* relation between *beat* and *federer* predictive of relevance. Passages 1 and 4 satisfy the *precedence* relation, but, of the two, only passage 1 can be considered answer-bearing, so passage 4 is a false positive. Passages 2 and 3 do not satisfy the *precedence* relation, but because passage 2 is answer-bearing, it is a false negative.

In this example, it is necessary to introduce a type system with more abstract semantic types, corresponding to units of meaning rather than units of text, to select the answer-bearing passages, but not select false positives. Consider $T_{srl} \supset T_{surf}$, defined in Section A.3, which introduces new element and relation types necessary to model a simple version of semantic role labeling. In T_{srl}, predicate verbs are annotated as `targets`, and noun phrases participating in the event represented by the verb are labeled as either `agent`, for the initiator or source of the action, or `patient`, for that to or upon which the action is performed. An *attachment* relation is introduced to link `targets` to their *arguments*. See Figure 3.2 for a representation of the information need corresponding to the question *Who beat Federer?* under T_{srl}.

$$G_{in,\,srl} = (E,\; R,\; T_{srl})$$

$$E = \left\{ \begin{array}{l} e_1 = (\texttt{sentence}), \\ e_2 = (\texttt{target}), \\ e_3 = (\texttt{agent}), \\ e_4 = (\texttt{patient}), \\ e_5 = (\text{beat}), \\ e_6 = (\text{federer}) \end{array} \right\}; \quad R = \left\{ \begin{array}{l} (\mathit{enclosure},\; e_1,\; e_2), \\ (\mathit{enclosure},\; e_1,\; e_3), \\ (\mathit{enclosure},\; e_1,\; e_4), \\ (\mathit{enclosure},\; e_1,\; e_5), \\ (\mathit{enclosure},\; e_1,\; e_6), \\ (\mathit{enclosure},\; e_2,\; e_5), \\ (\mathit{enclosure},\; e_4,\; e_6), \\ (\mathit{precedence},\; e_5,\; e_6), \\ (\mathit{attachment},\; e_2,\; e_3), \\ (\mathit{attachment},\; e_2,\; e_4) \end{array} \right\}$$

Figure 3.2: Information need corresponding to the input question *Who beat Federer?* expressed under T_{srl}, which is defined in Section A.3. Note that the `agent`, which is the focus of the question, is specified in the information need, but does not participate in any *enclosure* relations with *keyterms*.

3.4 The Passage Retrieval Process

Within the context of a QA system, the IR component performs passage retrieval in response to an information need G_{in} represented as an annotation graph. From the formal perspective, the process consists of mapping a scoring function f over each passage in the collection, in order to compute some type of similarity metric between the information need and each passage. The passages can then be ranked in descending order of score.

$$f(G_{in},\; G_{passage}) = score$$

Consider a matching function f_{bow} that scores based on the overlap of *enclosure* relations holding between the `sentence` and specific *keyterms* between the information need and passage annotation graphs. If G_{in} contains n of these relations, the score is incremented $\frac{1}{n}$ for each relation that is also present in $G_{passage}$. For example, passages 1, 3 and 4 would achieve the maximum score of 1.0, while passage 2 would score 0.5, because it is does not satisfy the requested *enclosure* relation between the `sentence` and *beat*, or a morphological variant of it. Assuming ties are broken by passage number, the complete ranking under f_{bow} is given in Table 3.2. The average precision for this ranking is 0.75.

Table 3.2: Passage ranking under f_{bow} for the question, *Who beat Federer?*. Answer-bearing passages are given in bold face.

Score	Number	Passage
1.0	1	***Nadal beat Federer.***
1.0	3	*Federer beat Safin.*
1.0	4	*Safin was beaten by Federer.*
0.5	2	***Federer was defeated by Nadal.***

While f_{bow} scoring function is easy to understand, it does not do a good job of approximating the QA system's information need. Despite the fact that it is answer-bearing, passage 2 is penalized for missing a requested *enclosure* relation between the `sentence` and a specific *keyterm*. Perhaps a better scoring function, call it f_{surf}, would be able to assign value for the *precedence* relation between *beat* and *Federer* in $G_{in,\,surf}$, shown in Figure 3.1. Assuming that it is worth equal credit compared to the *enclosure* relations, the passages 1 and 4 would both score the maximum 1.0, passage 3 would score 0.67, and passage 2 would score 0.33 in the ranking shown in Table 3.3. As before, the average precision of this ranking is 0.75. One answer-bearing passage is tied for the top ranking, and the other is ranked last, so there is still room to improve upon this ranking. Note that there is no way to assign relative weights to the three different constraints in this example so as to rank both passages 1 and 2 ahead of passages 3 and 4, because passages 1 and 4 are indistinguishable under T_{surf}.

Table 3.3: Passage ranking under f_{surf} for the question, *Who beat Federer?*. Answer-bearing passages are given in bold face.

Score	Number	Passage
1.0	1	***Nadal beat Federer.***
1.0	4	*Safin was beaten by Federer.*
0.67	3	*Federer beat Safin.*
0.33	2	***Federer was defeated by Nadal.***

To recover the optimal ranking for this example, it is necessary to be able to distinguish between passages 1 and 4, which differ only in terms of the mapping of *Nadal* and *Federer* to `agent` and `patient` under T_{srl}. Consider a scoring function f_{srl}, which assigns credit for the overlap in *attachment* relations between the information need and passage annotation graphs in addition to the *enclosure* and *precedence* relations. For the optimal ranking of the passages, shown in Table 3.4, it would be necessary to weight the *attachment* relations higher than both the *enclosure* and *precedence* relations, which is a reasonable thing to do because they carry more information about answers in text insofar as the QA system is concerned. With a weight of 0.5 on *attachment* relations, and 0.25 each for the other relations, the passages are ranked in order 1, 2, 4, 3, with scores 1.0, 0.625, 0.5 and 0.25, respectively. The average precision of this ranking is 1.0.

Table 3.4: Passage ranking under f_{srl} for the question, *Who beat Federer?*. Answer-bearing passages are given in bold face.

Score	Number	Passage
1.0	1	***Nadal beat Federer.***
0.625	2	***Federer was defeated by Nadal.***
0.5	4	*Safin was beaten by Federer.*
0.25	3	*Federer beat Safin.*

Unfortunately, many of the off-the-shelf IR systems commonly used by QA system developers do not support constraint-checking at the level of T_{srl}, and instead, operate at level closer to T_{bow} or T_{surf}. Internally, these IR systems ignore the higher-order constraints and focus instead on relations that can be represented in their query languages, including *keyterm* proximity, frequency and ordering. As a result, passages that are not answer-bearing, but that score well based on lower-level relations, can be ranked highly. It is up to the downstream Answer Generation module, which is able to consider the full complement of linguistic and semantic constraints, to identify the answer-bearing passages among the false positives.

This thesis proposes methodologies for retrieval-time checking of linguistic and semantic constraints on text in service of reducing false positives, to improve both passage retrieval quality and end-to-end QA system accuracy. The generalized annotation graph formalism is assumed as the underlying representation for the linguistic and semantic constraints studied here, which derive from the NLP performed by the system's Question Analysis module, which includes named entity recognition, syntactic parsing and semantic chunking. Any QA system whose information need can be reduced to an annotation graph can make use of the linguistic and semantic passage retrieval methods proposed in this thesis. Chapter 5 outlines a passage retrieval strategy based on a structured retrieval model, while Chapters 6 and 7 discuss a method that applies rank-learning techniques.

Chapter 4

Experimental Methodology

This chapter introduces a variety of resources used in this thesis, including Natural Language Processing (NLP) and Information Retrieval (IR) tools, as well as data resources. Subsequent chapters detailing experiments will reference the descriptions here. Many of the items described here are used in more than one of the subsequent chapters in this thesis.

4.1 Test Collections

The test collections described here are the basis for the experiments in passage retrieval for Question Answering (QA) described in this thesis. Test collections consist of a set of questions, each of which has a corresponding set of answers and set of passage retrieval relevance judgments over a given text collection.

4.1.1 TREC QA 2002 "MIT 109"

As described in Section 2.2, the U.S. National Institute of Standards and Technology (NIST) held comparative evaluations for QA systems between 1999 and 2007 as part of their annual Text REtrieval Conferences[1] (TRECs). This test collection consists of 109 *factoids*, or factual, short-answer questions, drawn from the TREC 2002 QA track main task. An example of a factoid question is question 1398, *What year was Alaska purchased?*, the answer to which is *1867*.

[1]See: `http://trec.nist.gov`

The text collection used in the TREC 2002 QA track was the AQUAINT corpus [19]. The corpus consists of 1,033,162 documents (approximately 375 million words; 3.0 GB) of English newswire text covering three years of Associated Press (1998 through 2000), three years of the New York Times (1998 through 2000), and five years of the English-language stories from the Xinhua News Agency (1996 through 2000)

For each question, TREC provides a few[2] examples of relevant documents selected from the AQUAINT corpus, but these are explicitly not guaranteed to be exhaustive. This fact makes TREC questions difficult to use as training or evaluation data, because unjudged documents can not be assumed to be non-relevant. The particular 109 questions in this test collection were chosen because there exists a freely-available[3], extensive set of document-level relevance judgments over the AQUAINT corpus, developed at MIT, beyond what TREC provides [5, 7, 32]. For our example question, TREC provides 6 relevant documents, and the test collection provides an additional two.

For passage retrieval at the level of individual sentences, these document-level judgments were converted to sentence-level judgments according to the following procedure. The answer patterns for the TREC 2002 main task, provided by Ken Litkowski of CL Research and distributed by NIST[4], were used to identify sentences containing the answer to a particular question located in documents judged to be relevant for that question. Each of these sentences was manually judged to be *answer-bearing* or not, according to the definition that an answer-bearing sentence must completely contain the answer to the question without requiring inference or aggregation outside of the sentence.

To illustrate the concept of an answer-bearing sentence, consider the question, *Who killed Kennedy?* Even in a document describing the assassination, the sentence *Oswald killed him* can not be considered answer-bearing, because it requires anaphora resolution to understand that *him* refers to *Kennedy*. A sentence such as *Oswald assassinated Kennedy* is answer-bearing, because synonymy is allowed. Similarly, *Everest is the tallest mountain* would be considered an answer bearing sentence for the question, *What is the highest point on Earth?*.

For our example question, an answer-bearing sentence from AQUAINT document APW19980907.1163 reads *Alaska's economy has been based on its vast wealth*

[2]In the TREC 2002 QA track main task, on average, approximately 4 relevant documents were provided for each of the 500 factoid questions.

[3]See: `http://www.umiacs.umd.edu/~jimmylin/downloads/qa-test-collection.tar.gz`

[4]See: `http://trec.nist.gov/data/qa/2002_qadata/main_task_QAdata/patterns.txt`

of natural resources since the United States bought the territory from Russia in 1867. Another example of an answer-bearing sentence from document APW19990329.0045 is *In 1867, U.S. Secretary of State William H. Seward reached agreement with Russia to purchase the territory of Alaska for $7.2 million, a deal roundly ridiculed as "Seward's Folly."* The following sentence from document APW19981012.1283 can not be considered answer bearing, because it fails to support one aspect of the question premise; it does not necessarily indicate that a purchase has taken place: *In 1867, the United States took formal possession of Alaska from Russia.*

4.1.2 CLEF-QA 2004/2006

Considered the European counterpart to TREC, the Cross Language Evaluation Forum[5] (CLEF) began running comparative evaluations in both monolingual and cross-lingual QA in 2003. This test collection consists of 200 questions each from the 2004[6] and 2006[7] Italian monolingual QA tracks. The CLEF questions consist of a mix of factoid (82%), list (2%) and definition (16%) questions. See Figure 4.1 for an example of a CLEF factoid question in Italian, with gloss.

Quale	stato	si	contendono	da	40
which.sg	*state.sg*	*refl.*	*contend.pres.ind.3p.pl*	*for*	*40*
anni	il	Pakistan	e	l'	India?
year.pl	*the.masc.sg*	*Pakistan*	*and*	*the.sg*	*India*

Figure 4.1: Italian-language factoid question number 0122 from the CLEF 2006 QA track. In English, the question reads *For what state have Pakistan and India been contending for 40 years?* The answer is *Kashmir.*

The Italian-language text collection used in this track consists of 157,558 documents (approximately 26 million words; 352 MB) of newswire text drawn from stories published in La Stampa (a Torinese daily) in 1994, including newswire content from Ansa (Agenzia Nazionale Stampa Associata), and stories published by the Schweizerische Depeschenagentur (SDA), the Swiss national news agency, in 1994 and 1995. The collection was prepared by stopping the 100 most frequent terms, mostly closed class terms such as determiners, conjunctions, prepositions and articulated prepositions, auxiliaries, and forms of common verbs such as *essere*, *avere* and

[5]See: `http://clef-campaign.org/`

[6]See: `http://clef-qa.itc.it/2004/down/test04/clefqa04-test-IT-IT.txt`

[7]See: `http://clef-qa.itc.it/down/qa_test06/clefqa06_test_IT-IT.txt`

andare.

As for the English-language test collection, the organization of the track provided sample relevant documents, and the judgments were broadened by a native speaker of the language. The same procedure and definition for answer-bearing sentences was used to compile a set of sentence-level judgments from the document-level relevance judgments. See Figure 4.2 for two answer-bearing sentences for our example question.

AGZ.940127.0056.814630			
India *India*	e *and*	Pakistan *Pakistan*	hanno *aux.3p.pl*
combattuto *combat.past.part*	per *for*	il *the.masc.sg*	Kashmir *Kashmir*
due *two*	guerre, *war.pl*	nel *in+the.masc.sg*	1947 *1947*
e *and*	nel *in+the.masc.sg*	1956. *1956*	
AGZ.940907.0080.82456			
Nel *in+the.masc.sg*	suo *his/her.masc.sg*	intervento *statement*	il *the.masc.sg*
primo *prime.masc.sg*	ministro *minister.sg*	pachistano *Pakistani*	ha *aux.3p.sg*
anche *also*	sollevato *raise.past.part*	la *the.fem.sg*	questione *issue.sg*
del *of+the.masc.sg*	Kashmir, *Kashmir*	il *the.masc.sg*	territorio *territory.sg*
che *that*	da *for*	40 *40*	anni *year.pl*
il *the.masc.sg*	Pakistan *Pakistan*	contende *contend.pres.ind.3p.sg*	alla *with+the.fem.sg*
vicina *nearby.fem.sg*	India. *India*		

Figure 4.2: Two answer-bearing sentences for Italian-language CLEF question 20060122, which corresponds to: *For what state have Pakistan and India been contending for 40 years?* The answer is *Kashmir.*

Additionally, a set of block-level relevance judgments was created for the Italian-language data. A *block* is defined as a sliding three-sentence window, except in cases where document structure existing in the corpus defines a paragraph containing fewer than three sentences. A block is considered answer-bearing if any one of the sentences it contains is answer-bearing on its own, or if the answer to the question is split across two or three sentences, as in the case of anaphora or ellipsis. A typical example of this phenomenon can be found in Figure 4.3, which shows two sentences that are answer-bearing when taken together, but not individually. The first sentence refers to *the two countries*, which are identified in the second sentence as *India and Pakistan*. The answer *Kashmir*, however, is in the first sentence, and referred to as *the contested territory* in the second.

AGZ.940921.0085.862679/862680			
La *the.fem.sg*	"linea *line.sg*	di *of*	controllo" *control.sg*
è *be.pres.ind.3p.sg*	in *in*	realtà *reality.sg*	la *the.fem.sg*
linea *line.sg*	del *of+the.masc.sg*	cessate *cease.sg*	il *the.masc.sg*
fuoco *fire.sg*	stabilita *established.fem.sg*	dopo *after*	l' *the.sg*
ultima *last.fem.sg*	guerra *war.sg*	tra *between*	i *the.masc.pl*
due *two*	paesi *country.pl*	per *for*	il *the.masc.sg*
Kashmir, *Kashmir*	combattuta *fought.fem.sg*	nel *in+the.masc.sg*	1965. *1965*
Il *the.masc.sg*	primo *first.masc.sg*	conflitto *conflict.sg*	armato *armed.masc.sg*
tra *between*	India *India*	e *and*	Pakistan *Pakistan*
per *for*	il *the.masc.sg*	territorio *territory.sg*	conteso *contested.masc.sg*
si *refl.*	verificò *occur.past.3p.sg*	nel *in+the.masc.sg*	1947. *1947*

Figure 4.3: Two sentences from an answer-bearing block for Italian-language CLEF question 20060122, which corresponds to: *For what state have Pakistan and India been contending for 40 years?* The answer is *Kashmir*.

4.2 English-Language NLP Tools

The NLP tools described in this section were used to prepare the AQUAINT corpus for the English-language QA experiments.

4.2.1 MXTerminator

MXTerminator is a sentence segmentation tool based on a maximum-entropy model [52]. It is the first step in the corpus preparation process because some downstream tools, such as ASSERT, discussed below, require input to be in the form of an individual sentence. In addition, the sentence boundaries are used as a unit of retrieval for the English-language passage retrieval experiments.

The tool works by considering each occurrence of sentence-final punctuation as a potential sentence break. MXTerminator's model classifies each occurrence as to whether or not it is a true sentential boundary using lexical features, such as capitalization and membership in dictionaries of known abbreviations, of the words to the left and the right of the occurrence. The model is shown to be approximately 98% accurate on newswire text, and casual observation of the segmented AQUAINT data seems to confirm this.

4.2.2 BBN Identifinder

BBN Identifinder is a widely-used tool for Named Entity Recognition [4], which is based on a hidden Markov model. Each token in running text is classified by the model as to which of a set of named entity types it belongs to, or none at all, based on the previous token's classification, and features of the previous and current token, including capitalization and presence of digits and symbols, as well as the bigram score with respect to the trained model for that class. The model is shown to achieve up to 95% in F-measure on English-language data.

The version of Identifinder used in this thesis can recognize the following types: `date`, `location`, `money`, `org`, `percent`, `person`, `time`, and `weekday`. Of these, only `date`, `location`, `org` and `person` occur in the TREC 2002 "MIT 109" test collection, either explicit in the question text, or as the expected answer type. For the example question, *What year was Alaska purchased?*, the token *Alaska* is recognized as a `location`, and the expected answer type is determined to be `date`. Similarly, in the answer-bearing sentence, *In 1867, the United States took formal possession of Alaska* *from Russia.*, *1867* is labeled as a `date`, while *United States*, *Alaska* and *Russia* are recognized as entities of type `location`.

4.2.3 ASSERT

ASSERT is a shallow semantic parser [50] that identifies verbs, as well as the noun phrase or prepositional phrase chunks that represent their arguments. The model is trained on PropBank [27], which is a set of hand-edited semantic role labels on the Penn Treebank [33]. ASSERT is implemented as a two-level classification problem using Support Vector Machines (SVMs). The tool works by first applying a syntactic parser to input text, then, for each verb, classifying each node in the constituent structure tree as to whether or not that node represents an argument to that verb, using features of the tree such as the path between the verb and candidate argument. The second level of classification employs a set of one-versus-all SVMs to determine the argument's semantic role label. The authors evaluated the tool on a sample of the AQUAINT corpus and reported that precision and recall of argument labels were 65.2% and 61.5%, respectively.

ASSERT gives verbs the label `target`, while arguments are labeled with semantic roles according to PropBank [27]. In general, role labels can vary among verb frames, but the developers of PropBank have made an effort to label `arg0`, the agent, or actor, in a predicate, and `arg1`, the patient, or to whom or upon what the action is performed, consistently across verb frames. In addition to these, the following argument types are represented in the test collections used in the experiments in this thesis: higher-order complements representing roles such as beneficiary and instrument for certain verb frames, `arg2`, `arg3` and `arg4`; and the following adjuncts: `argm-adv` (adverbial), `argm-dir` (directional), `argm-loc` (locative), `argm-mnr` (manner) and `argm-tmp` (temporal).

Our example question, *What year was Alaska purchased?*, contains a single `target` verb, *purchased*. The PropBank verb frame for *purchase* contains a single sense that prescribes five roles: `arg0` represents the purchaser, `arg1` represents the thing purchased, `arg2` represents the seller, `arg3` represents the price paid and `arg4` represents the beneficiary [27]. The ASSERT analysis of the question would label *Alaska* as `arg1`, and would label *what year* as a temporal adjunct `argm-tmp`, which is where one would expect the answer to the question to occur, as in the answer bearing sentence shown in Figure 4.4. Note that there will also be ASSERT output for the verb *reached*, which is omitted from the figure for clarity.

[`arg0` *U.S. Secretary of State William H. Seward*] *reached agreement with* [`arg2` *Russia*] *to* [`target` *purchase*] [`arg1` *the territory of Alaska*] [`arg3` *for $7.2 million*], *a deal roundly ridiculed as "Seward's Folly."*

Figure 4.4: ASSERT analysis showing predicate-argument structure for the verb *purchased* for an answer-bearing sentence for the question *What year was Alaska purchased?*

4.3 Italian-Language NLP Tools

The NLP tools described in this section were used to prepare the La Stampa and SDA text collections for the Italian-language QA experiments.

4.3.1 TextPro

TextPro, made freely-available for research purposes by the Fondazione Bruno Kessler, is a powerful suite of tools for Italian NLP, supporting lemmatization, morphological analysis, part-of-speech tagging, sentence segmentation and Named Entity Recognition [46]. TextPro's sentence segmenter, which appears to be a rule and dictionary-based approach, was used to provide individual sentences for input to the Chaos parser, described below.

Lemmatization is the process of converting an inflected form of a noun or verb to its lemma, or dictionary form. For verbs, this is normally the infinitive, and for nouns, the masculine singular form is selected. This is more important for a highly-inflected language such as Italian than for English, where a more simple stemming algorithm will do. TextPro's lemmatization feature is based on a morphological analyzer developed in Prolog and appears to the casual observer to be fairly accurate, but it was not used in the experiments in this thesis because it would create a significant alignment problem between the TextPro output and that of the Chaos parser, which tokenizes and lemmatizes its input differently.

4.3.2 Chaos

Chaos [3] is a chunk-level dependency parser for both Italian and English maintained by the Artificial Intelligence Research Group at the Università di Roma "Tor Vergata." The parser produces analyses in a proprietary representation called the eXtended Dependency Graph (XDG), in which the nodes are constituents and the

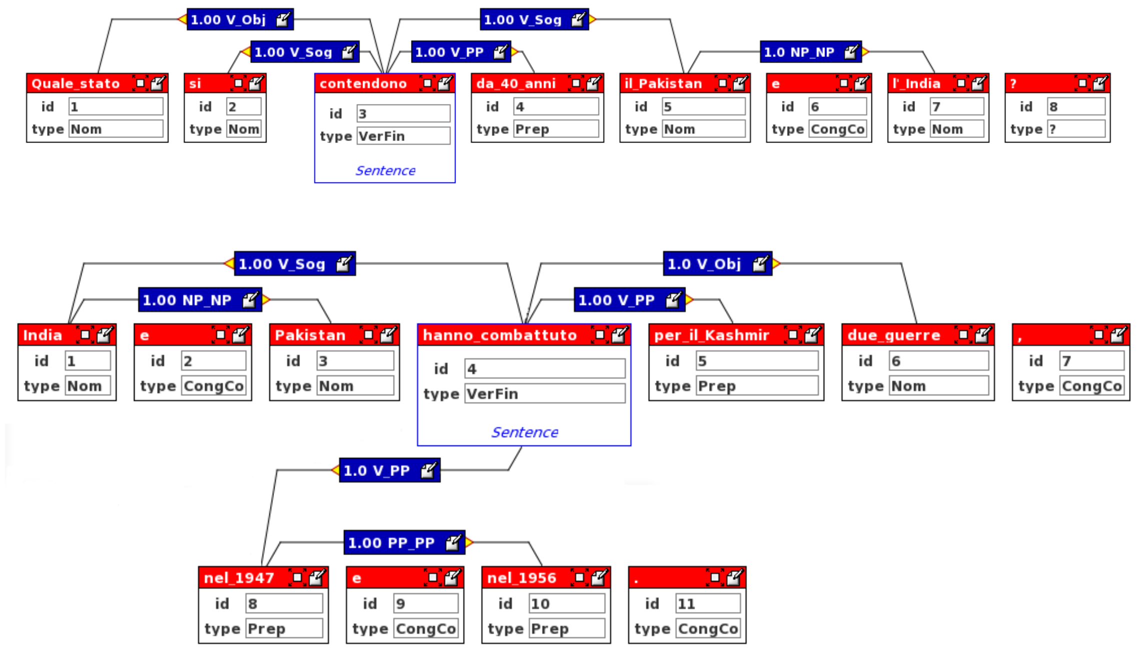

Figure 4.5: Chaos analysis for the Italian-language CLEF question 20060122 corresponding to *For what state have Pakistan and India been contending for 40 years?* (above), and for an answer-bearing sentence for that question (below).

edges are inter-constituent dependencies, which are usually grammatical functions, such as subject, and direct and indirect objects. Plausibility scores are associated with the edges allowing ambiguities such as prepositional attachments to be neatly packed within the representation.

The XDG formalism bears many similarities to the ASSERT representation used in the English-language experiments in this thesis. Both tools center their analyses on verbs in a sentence, identifying chunks of text that have special relationships with the verb, be they semantic roles, in the case of ASSERT, or grammatical functions, in the case of Chaos. The difference between the two tools is the level of abstraction. Semantic role labeling abstracts over the syntactic distinctions of active and passive voice, in which the patient, or `arg1`, becomes the grammatical subject of the sentence and the agent, or `arg0`, moves to a prepositional, or oblique, argument.

Figure 4.5 shows an example of Chaos output corresponding to our example question, *Quale stato si contendono da 40 anni il Pakistan e l'India?* (above), as well as for an answer-bearing sentence for that question (below). The root of the dependency graph is a verb; *contendono* in the question, and *hanno combattuto* in the answer. Inter-constituent dependencies are shown as labeled arcs between constituents. `V_Obj` identifies the direct object, which is the interrogative phrase *quale stato* in the question, and *due guerre* in the answer. Temporal and other types of prepositional arguments are labeled as `V_PP`, such as *da 40 anni* in the question and *nel 1947 e nel 1956* in the answer. It is a feature of the Chaos parser that *nel 1956* is not connected directly to the verb, but is instead attached to *nel 1947* through a coordinating conjunction. The same phenomenon is evident in the subjects of the verbs in both the question and the answer, which are labeled `V_Sog`, standing for *soggetto*, which means subject in Italian.

4.4 Information Retrieval Tools

The IR tools tools described in this section form the foundation of the passage retrieval strategies studied in this thesis.

4.4.1 Indri Search Engine

Indri is the latest search engine in the open-source Lemur toolkit for Language Modeling and Information Retrieval [8]. The Indri retrieval model is a combination of the language modeling approach [48, 29] and the inference network model [60]. It supports rich structured queries like previous systems based on the inference network model, for example InQuery [10], but the probabilities are estimated using language modeling instead of the Okapi ranking function [34, 56].

The Indri index structures and retrieval model directly support the concept of a field, which is a typed extent defined over a contiguous sequence of tokens. Fields can enclose each other and overlap each other arbitrarily. Fields are commonly used as a means for separating distinct portions of documents, or different document representations, so that they can be independently scored. Examples include title and body fields, chapter, section, subsection, paragraph and sentence fields, and fields containing the URL and anchor text for web documents. For the AQUAINT

[8]See: `http://www.lemurproject.org`

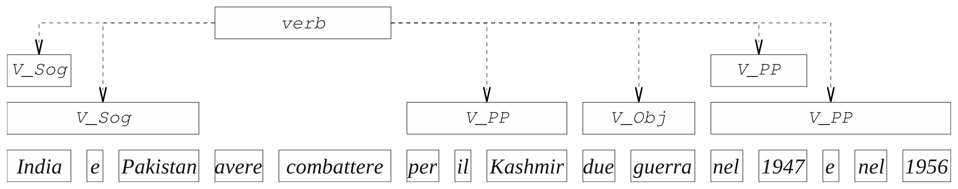

Figure 4.6: Indri index representation for the sentence, *India e Pakistan hanno combattuto per il Kashmir due guerre, nel 1947 e nel 1956.* Fields representing annotations are shown as boxes above the row of tokens at the bottom. Parent pointers are represented as arrows.

corpus used in the English-language QA experiments, field types derived from NLP analysis include named entity types, such as `date` and `person`, `target` verbs, and arguments such as `arg0` and `argm-tmp`[9].

The version of Indri used throughout this thesis extends the notion of a field to include a *parent* pointer, through which a field can optionally point to another field within the same document [43, 8]. This pointer provides a means of checking relationships between fields that do not enclose one another. Support for this relation makes it possible to represent the relationships between verbs and their arguments in the index, and to check the relationship at query time, because neither field type encloses the other.

Indexing a corpus processed with Chaos requires some additional effort because Indri does not support typed pointers. If it were to, the Chaos representation could be indexed directly with field names corresponding to the constituent types, and inter-constituent dependencies being represented as typed pointers between fields. This issue is mitigated by naming the non-verb constituent fields after the inter-constituent dependency type. For example, in Figure 4.5, the phrase *due guerre* would be indexed as a field of type `V_Obj`, not `Nom`. Similarly, the phrase *India e Pakistan*, also of type `Nom`, would be indexed as `V_Sog` because it is the relationship to the verb that is more important than the constituent type. There will actually be two `V_Sog` fields for the verb in this example; one corresponding to *India*, and the other corresponding to *India e Pakistan*, which is constructed by following the inter-constituent dependency between *India* and *Pakistan*. See Figure 4.6 for a graphical depiction of the index representation for the sentence corresponding to the lower portion of Figure 4.5.

[9]The on-disk size of the index scales with the number of field instances and is about 7.7 GB, 77% larger than the AQUAINT corpus indexed without fields.

4.4.2 Committee Perceptron

The Committee Perceptron is an on-line algorithm for learning linear ranking functions for learning-to-rank tasks [16]. It is a generalization of the Voted Perceptron [12] and Pocket Perceptron algorithms [18]. Committee Perceptron has been shown to perform at least as well as the state-of-the art RankSVM algorithm [25], while requiring much less training time. Like RankSVM, the Committee Perceptron learns a function that minimizes the number of non-preferred passages ranked ahead of preferred passages; this, in turn, maximizes a lower bound on Mean Average Precision.

The Committee Perceptron is trained on pairwise preference relationships between pairs of judged passages $S = R \times N = \{\mathbf{p}_{nq}, \mathbf{p}_{rq}\}$, which come from sampling sets of relevant and non-relevant passages, R and N, respectively, for each training question and pairing them, such that a pairwise preference relationship is expressed preferring each relevant training passage over each non-relevant training passage.

Passages are represented as feature vectors $\mathbf{p}_{iq} = [\mathbf{f}_1(p_i, q), \ldots, \mathbf{f}_M(p_i, q)]$ where the $\mathbf{f}_j$; $j = 1 \ldots M$ are the feature values. Features are scaled to zero-mean unit-variance per-question prior to training and testing.

The training process involves iterating over passage pairs in S and updating a hypothesis weight vector $\mathbf{w}^i$, also of length M, on each ranking mistake, as in a standard perceptron. The update rule adds or subtracts the difference between the relevant passage and the non-relevant passage to the current weight vector: $\mathbf{w}^{i+1} = \mathbf{w}^i + (\mathbf{p}_{rq} - \mathbf{p}_{nq})$.

Prior to the hypothesis update, however, the algorithm maintains the committee. If the committee size is smaller than the maximum committee size N_{com}, the current hypothesis $\mathbf{w}^i$ is added to the committee. If the committee is full, and the current hypothesis' success counter c_i is greater than the success counter of the least successful hypothesis in the committee, that least successful hypothesis is evicted in favor of the current hypothesis. The success counter is incremented for each consecutive correctly-ranked pair of items.

At the end of the training process, the committee K contains the N_{com} best hypotheses, which are weight vectors $\mathbf{w}^k$ of length M, with their associated success counters c_k. Each of these hypotheses constitutes a passage ranking function that scores passages by taking the dot product of the weight vector and the passage feature vector. The overall scoring function is an average of the hypotheses, weighted by the success counters. For a complete description of the committee perceptron algorithm, refer to Table 4.1 [6].

For the rank-learning experiments in this thesis, the parameters are set to default values known to be broadly applicable to passage retrieval tasks: the committee size $N_{com} = 30$, and the number of passage pairs to sample $T = 10,000$. These parameter values were shown to be effective without the need for any additional tuning in the experiments described in Chapters 6 and 7.

Table 4.1: Committee Perceptron Algorithm for Passage Ranking, courtesy of Elsas [6]

Input:
Number of passage pairs to sample T
Committee size N_{com}
List of training relevant/non-relevant passage pairs $S = R \times N = \{(\mathbf{p}_{nq}, \mathbf{p}_{rq})\}$

Output:
Set of feature weight vectors and their success counters $K = \{(\mathbf{w}^k, c_k) | k = 1 \dots N_{com}\}$

```
1. Initialize $i = 0$, success counter $c_i = 0$, initial parameters $\mathbf{w}^0$, committee $K = \emptyset$.
2. For $t = 0, \dots, T$:
        From $S$, sample query $q$ and relevant/non-relevant passages $(\mathbf{p}_{nq}, \mathbf{p}_{rq})$
            If $Score(\mathbf{p}_{nq}, \mathbf{w}^i) \geq Score(\mathbf{p}_{rq}, \mathbf{w}^i)$ then
                $(\mathbf{w}_{\min}, c_{\min}) \in K$ s.t. $c_{\min} = \min_k c_k \in K$
                If $c_i > c_{\min}$ then: add $(\mathbf{w}^i, c_i)$ to $K$
                If $|K| > N_{sub}$: remove $(\mathbf{w}_{\min}, c_{\min})$ from $K$
                update: $\mathbf{w}^{i+1} = \mathbf{w}^i + (\mathbf{p}_{rq} - \mathbf{p}_{nq})$ and $i = i + 1$
            Else update: $c_i = c_i + 1$
3. Output:
        Committee $K$
        $Score^*(\mathbf{p}_{iq}, \mathbf{w}^1, \dots, \mathbf{w}^{N_{com}}) = \mathbf{p}_{iq} \cdot \frac{\sum_k \mathbf{w}^k \times c_k}{\sum_k c_k}$
```

Chapter 5

Structured Retrieval

This chapter introduces an approach to linguistic and semantic passage retrieval for Question Answering (QA) based on mapping the problem of annotation graph retrieval onto the existing structured retrieval features of the Indri search engine, described in Section 4.4.1. Indri's structured retrieval operators make it possible to check the linguistic and semantic constraints in the annotation graph at retrieval time.

The experiments presented here and originally published in [8], evaluate whether the structured retrieval approach can provide a *better quality* passage ranking when compared to a baseline retrieval approach consisting of keyterms drawn from question, with named entity support for the expected answer type, which is considered to be a strong baseline for QA.

5.1 Structured Retrieval for Question Answering

Structured retrieval is an Information Retrieval (IR) paradigm in which documents are viewed as sets of typed, contiguous regions of text called *fields*. Structured retrieval allows these fields to be scored against a query individually, and for these per-field scores to be combined into an overall, document-level score. The classic structured retrieval application is the case of a document representation consisting of a title field and a body field. The intuition that a term match inside the title field is more predictive of relevance than a match inside the body field is encoded into the scoring function by placing more of the weight on the title field match score when combining it with the body field match score. To apply structured retrieval techniques to a QA task, the fundamental insight is that the same mechanism that can be used to represent document structure as fields can also be used to represent linguistic and semantic annotations on text [43, 8].

5.2 A Type System for English QA

The structured retrieval experiments in this chapter make use of the type system $T_{bbn+assert}$, which, defined in Section A.6, supports sentence segmentation by MX-Terminator, Named Entity Recognition by BBN Identifinder and Semantic Role Labeling by ASSERT. See Section 4.2 for more information on these English-language Natural Language Processing (NLP) tools.

To illustrate an example of the type of information need that would be generated by a Question Analysis module utilizing this type system, consider question 1402 from the TREC 2002 "MIT 109" test collection, defined in Section 4.1.1, *What year did Wilt Chamberlain score 100 points?* The expected answer to this question is a year, so the wh-word becomes a placeholder of *entity* type `date`. The verb *score* is the `target` verb for the predicate-argument structure in the question. A `person` named *Wilt Chamberlain* is the one doing the scoring, so he is tagged with the agentive Semantic Role Label, `arg0`. Similarly, the *100 points* is the patient, or `arg1`, of the action. The answer would be expected to show up in a temporal adjunct `argm-tmp` attached to the `target`. See Figure 5.1 for a depiction of this information need.

5.3 Structured Query Operators

With *annotations* represented as fields in the Indri index, and *relations* expressed using enclosure or parent pointers, as described in Section 4.4.1, the same structured query operators that allow for scoring a document based on the score of its fields can support query-time linguistic and semantic constraint checking[1]. Constraints consisting of *enclosure* relations between *annotations* and *keyterms* in the information need can be checked using the same query syntax that checks term occurrences within fields, for example: `#combine[person](` *wilt chamberlain* `)`. *Enclosure* between *annotation* pairs can be checked using nested `#combine` operators, or the `#any:`*field* operator, which matches occurrences of fields enclosed within other fields,

[1]Indri queries containing many nested structured query operators take more time to evaluate than bag-of-words queries. See Zhao and Callan (2009) [67] for discussion of this issue.

$$G_{1402} = (E,\ R,\ T_{bbn+assert})$$

$$E = \left\{ \begin{array}{lll} e_1 = (\texttt{sentence}), & e_6 = (\texttt{person}), & e_{10} = (\text{chamberlain}), \\ e_2 = (\texttt{target}), & e_7 = (\texttt{date}), & e_{11} = (\text{score}), \\ e_3 = (\texttt{arg0}), & e_8 = (\text{year}), & e_{12} = (\text{100}), \\ e_4 = (\texttt{arg1}), & e_9 = (\text{wilt}), & e_{13} = (\text{points}) \\ e_5 = (\texttt{argm-tmp}), & & \end{array} \right\}$$

$$R = \left\{ \begin{array}{lll} (enclosure,\ e_1,\ e_2), & (enclosure,\ e_1,\ e_{11}), & (enclosure,\ e_5,\ e_7), \\ (enclosure,\ e_1,\ e_3), & (enclosure,\ e_1,\ e_{12}), & (enclosure,\ e_5,\ e_8), \\ (enclosure,\ e_1,\ e_4), & (enclosure,\ e_1,\ e_{13}), & (enclosure,\ e_6,\ e_9), \\ (enclosure,\ e_1,\ e_5), & (enclosure,\ e_2,\ e_{11}), & (enclosure,\ e_6,\ e_{10}), \\ (enclosure,\ e_1,\ e_6), & (enclosure,\ e_3,\ e_6), & (attachment,\ e_2,\ e_3), \\ (enclosure,\ e_1,\ e_7), & (enclosure,\ e_3,\ e_9), & (attachment,\ e_2,\ e_4), \\ (enclosure,\ e_1,\ e_8), & (enclosure,\ e_3,\ e_{10}), & (attachment,\ e_2,\ e_5), \\ (enclosure,\ e_1,\ e_9), & (enclosure,\ e_4,\ e_{12}), & \\ (enclosure,\ e_1,\ e_{10}), & (enclosure,\ e_4,\ e_{13}), & \end{array} \right\}$$

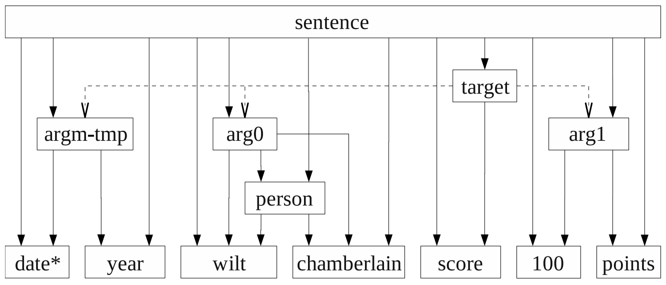

Figure 5.1: Information need corresponding to the input question *What year did Wilt Chamberlain score 100 points?* expressed under $T_{bbn+assert}$.

as in: `#combine[argm-tmp]( #any:date )`. Constraints consisting of *attachment* relations between `targets` and *arguments* use a “dot-slash” syntax, which instructs the retrieval model to score “child” fields, or fields whose parent pointers point to the current field, as in: `#combine[./arg1](` *100 points* `)`.

5.4 Experimental Methodology

Experiments first published in [8] test the effectiveness of the structured retrieval techniques described above for the task of passage retrieval within the context of a QA system. The experiments used the TREC 2002 “MIT 109” test collection described in Section 4.1.1 in a sentence retrieval task. The test collection was split 55/54 into a training and test set, with a roughly equal distribution of question types. The training set was used to select the optimal smoothing parameters, which turned out to be Jelinek-Mercer with the document weight $\lambda_D = 0.2$ and the collection weight $\lambda_C = 0.2$.

5.4.1 Query Formulation

To measure retrieval quality in isolation and arrive at an upper bound for the appropriateness of structured retrieval techniques to this task, these experiments used gold-standard query formulation. Queries were formulated under both $T_{bbn+assert}$, defined in Section A.6, and T_{bbn}, shown in Section A.4, which lacks the `target`, *argument* and *attachment* definitions.

Two queries were automatically formulated for each answer-bearing sentence for each question in the test collection, one using the T_{bbn} type system, and the other using $T_{bbn+assert}$. The queries are based on *keyterms* from the question occurring within the answer-bearing sentence. The annotation graph representation of the sentence is pruned by removing all words that do not occur in the question. In addition, extra *enclosure* relations are removed such that each *keyterm* or *entity* participates in exactly one *enclosure* relation with its smallest enclosing element, where size is computed by the number of *keyterms* enclosed. Ties are broken in favor of *entities*. *Enclosure* relations in which a larger element encloses an *argument* are also removed. The remaining graph looks like a query tree and the linguistic and semantic constraints expressed in the graph can be checked using Indri query operators, as described in Section 5.3.

To illustrate the gold-standard query formulation, take question 1402 from TREC

2002, *What year did Wilt Chamberlain score 100 points?* There are 14 sentences in the AQUAINT corpus judged to be answer-bearing, 12 of which are unique. One answer bearing sentence is *On March 2, 1962, in Hershey, PA, Wilt Chamberlain scored 100 points against the Knicks.* The question keyterms *Wilt*, *Chamberlain*, *score*, *100* and *points* occur in this sentence. All other words are pruned, and a placeholder named entity of type *date* is introduced to stand in for the expected answer type. Assume that morphological normalization is in use so that *scored* and *score* are considered to be the same word, and also that named entity recognition software has identified *Wilt Chamberlain* as a `person`, and *March 2, 1962* as a `date`. Figure 5.2 shows the pruned annotation graph representation of this sentence under T_{bbn}.

$$G_{sent,\,bbn} = (E,\, R,\, T_{bbn})$$

$$E = \left\{ \begin{array}{l} e_1 = (\texttt{sentence}), \\ e_2 = (\texttt{date}), \\ e_3 = (\texttt{person}), \\ e_4 = (\text{wilt}), \\ e_5 = (\text{chamberlain}), \\ e_6 = (\text{score}), \\ e_7 = (100), \\ e_8 = (\text{point}) \end{array} \right\}; \quad R = \left\{ \begin{array}{l} (enclosure,\, e_1,\, e_2), \\ (enclosure,\, e_1,\, e_3), \\ (enclosure,\, e_1,\, e_4), \\ (enclosure,\, e_1,\, e_5), \\ (enclosure,\, e_1,\, e_6), \\ (enclosure,\, e_1,\, e_7), \\ (enclosure,\, e_1,\, e_8), \\ (enclosure,\, e_3,\, e_4), \\ (enclosure,\, e_3,\, e_5) \end{array} \right\}$$

Figure 5.2: Pruned annotation graph for the sentence, *On March 2, 1962, in Hershey, PA, Wilt Chamberlain scored 100 points against the Knicks* under type system T_{bbn}.

The pruned annotation graph representation for the sentence $G_{sent,\,bbn}$ can be mapped directly into Indri query syntax according to the following procedure. For a sentence retrieval task, the outermost query clause is a `#combine[sentence]` operator, which retrieves, scores and ranks sentence extents directly. The *enclosure* relations between the `sentence` and *entities* map directly map directly into the Indri concept of extent enclosure. Enclosure of a *keyterm* element becomes standard keyterm enclosure. An *entity* containing other *entities* or *keyterms* is represented by an inner `#max( #combine[`*entity*`]( ... ))` clause. The `#max` operator selects the best matching of the enclosed instances of type *entity* to score the enclosing extent, in the case where there are more than one[2]. If the field is empty, it becomes a place-

[2] When a nested `#combine` operator returns more than one matching extent, the scores will

holder of the form `#any:`*entity* operator. The complete Indri query formulated for this sentence under the T_{bbn} type system is shown in the top half of Figure 5.4, which can be found at the end of this section.

Under $T_{bbn+assert}$, the annotation graph representation of the answer-bearing sentence is richer. ASSERT identifies *score* as the `target` verb in the sentence. A `person` named *Wilt Chamberlain* is the `arg0`, *100 points* is in the `arg1` position. The `date` instance occurs within an `argm-tmp`, which is a temporal adjunct to the verb. Figure 5.3 shows the pruned graphical representation of the sentence under $T_{bbn+assert}$.

The annotation graph $G_{sent,\,bbn+assert}$ can also be mapped directly into Indri query syntax, but the mapping is a bit more complicated. As before, the outermost operator is a `#combine[sentence]` that retrieves sentences directly. This operator contains clauses for all *annotations* and *keyterms* that are directly enclosed by the `sentence`. Again, empty *annotations* become placeholders of the form `#any:`*annotation*, while non-empty *annotations* become clauses of the form `#max( #combine[`*annotation*`]( ... ))`. *Arguments* that are attached to `targets` are mapped into clauses of the form `#max( #combine[./`*argument*`]( ... ))`, which uses parent pointers to check the `target`-*argument attachment* relation. In the bottom half of Figure 5.4, the complete query under $T_{bbn+assert}$ for this sentence is shown.

5.4.2 Experimental Conditions

There are two experimental conditions, the *single structure* and the *every structure* cases. In the *single structure* case, the QA system is searching for a specific answer-bearing structure known to be usable for answer extraction, and formulates a query to retrieve it explicitly, so this case can be thought of as an exact match application. To model this situation, each answer-bearing sentence is considered a unique topic, where only retrieved sentences that share the same annotation graph representation under the type system are treated as relevant. There are 369 such topics in the training set, and 250 in the test set.

In the *every structure* case, the QA system considers every answer-bearing structure useful for a particular question. In this case, one query is run for each answer-

be added together, resulting in an extremely low score, which will bring down the score of the enclosing extent. The `#max` operator is not the only way to choose among a list of scored extents, but it was the best method available at the time of the original publication of the experiments in this chapter [8]. See Zhao and Callan (2008) [66] for an exploration of a variety of evidence combination techniques that can provide a composite score for a list of scored extents.

$$G_{sent,\,bbn+assert} = (E,\, R,\, T_{bbn+assert})$$

$$E = \left\{ \begin{array}{l} e_1 = (\texttt{sentence}), \\ e_2 = (\texttt{target}), \\ e_3 = (\texttt{arg0}), \\ e_4 = (\texttt{arg1}), \\ e_5 = (\texttt{argm-tmp}), \\ e_6 = (\texttt{date}), \\ e_7 = (\texttt{person}), \\ e_8 = (\text{wilt}), \\ e_9 = (\text{chamberlain}), \\ e_{10} = (\text{score}), \\ e_{11} = (100), \\ e_{12} = (\text{point}) \end{array} \right\}, \quad R = \left\{ \begin{array}{l} (enclosure,\, e_1,\, e_2), \\ (enclosure,\, e_1,\, e_3), \\ (enclosure,\, e_1,\, e_4), \\ (enclosure,\, e_1,\, e_5), \\ (enclosure,\, e_1,\, e_6), \\ (enclosure,\, e_1,\, e_7), \\ (enclosure,\, e_1,\, e_8), \\ (enclosure,\, e_1,\, e_9), \\ (enclosure,\, e_1,\, e_{10}), \\ (enclosure,\, e_1,\, e_{11}), \\ (enclosure,\, e_1,\, e_{12}), \\ (enclosure,\, e_2,\, e_{10}), \\ (enclosure,\, e_3,\, e_7), \\ (enclosure,\, e_3,\, e_8), \\ (enclosure,\, e_3,\, e_9), \\ (enclosure,\, e_4,\, e_{11}), \\ (enclosure,\, e_4,\, e_{12}), \\ (enclosure,\, e_5,\, e_6), \\ (enclosure,\, e_7,\, e_8), \\ (enclosure,\, e_7,\, e_9) \end{array} \right\}$$

Figure 5.3: Pruned annotation graph for the sentence *On March 2, 1962, in Hershey, PA, Wilt Chamberlain scored 100 points against the Knicks* under the type system $T_{bbn+assert}$.

```
#combine[sentence](
   score 100 point #any:date
   #max( #combine[person]( wilt chamberlain ) ) )
```

```
#combine[sentence](
   #max( #combine[target]( score
     #max( #combine[./arg0](
       #max( #combine[person]( wilt chamberlain ) ) ) )
     #max( #combine[./arg1]( 100 point ) )
     #max( #combine[./argm-tmp]( #any:date ) ) ) ) )
```

Figure 5.4: Comparison of query formulation strategies for type systems T_{bbn} (above) and $T_{bbn+assert}$ (below) for the answer-bearing sentence,*On March 2, 1962, in Hershey, PA, Wilt Chamberlain scored 100 points against the Knicks.*

bearing sentence for a particular question, and the results are combined by Round Robin [62], a common meta-search technique that, despite its simplicity, was found to be the most effective for this task [8]. It is difficult, in general, to combine the results from the individual queries for a particular question, because the scores of the retrieved sentences are not directly comparable. It is a feature of Indri that very complex queries will produce scores orders of magnitude smaller than the scores resulting from simpler queries. Round Robin is successful in this application because it ignores score and only considers rank when merging result lists.

5.5 Experimental Results and Discussion

Table 5.1 compares the Mean Average Precision of the structured retrieval approach to linguistic and semantic passage retrieval, the Indri queries formulated under $T_{bbn+assert}$, to the baseline queries formulated under T_{bbn}. For each case, the difference between the $T_{bbn+assert}$ and T_{bbn} queries is statistically significant, with a $p < 0.0001$ according to Fisher's randomization test [55]. It can immediately be seen from the *single structure* results that, when the task is to find a specific semantic representation, more information than can be represented in T_{bbn} is required. The *every structure* case shows that the baseline is more effective when there are more types of passages that the QA system finds useful, but for this case as well, $T_{bbn+assert}$ produces a *better quality* ranking.

Figure 5.5 shows a comparison of the recall of the two approaches. As can be

Table 5.1: Comparison of Mean Average Precision for Single and Every Structure cases

	Single Structure ($N = 250$)			Every Structure ($N = 54$)		
Data	T_{bbn}	$T_{bbn+assert}$	p-value	T_{bbn}	$T_{bbn+assert}$	p-value
Train	0.0774	0.4837	< 0.0001	0.1417	0.2922	< 0.0001
Test	0.0872	0.4650	< 0.0001	0.1613	0.3423	< 0.0001

seen from Figures 5.5a and 5.5c, the extra information encoded in the $T_{bbn+assert}$ type system allows for much greater recall than T_{bbn} for the *single structure* case. For example, at rank 200, $T_{bbn+assert}$ provides a 96.9% and 46.6% improvement over T_{bbn} on the training and test topics, respectively. Figures 5.5b and 5.5d compare the two type systems for the *every structure* case. The advantage that $T_{bbn+assert}$ enjoys over T_{bbn} is less pronounced, but the improvement at rank 200 is still 12.8% and 11.4% for the training and test topics, respectively.

Figure 5.5 shows that $T_{bbn+assert}$ has superior recall of answer-bearing sentences on average, compared to the T_{bbn} type system, but tells little about the types of queries for which $T_{bbn+assert}$ is most helpful. In an attempt to isolate the conditions where the additional element and relation types defined in $T_{bbn+assert}$ are most useful, Table 5.2 compares Mean Average Precision for queries used in the *single structure* experiment having different levels of complexity. The complexity level is defined as the number of `#combine` operators in the $T_{bbn+assert}$ query, not counting the outer `#combine[sentence]` operator. The table also gives p-values according to Fisher's randomization test [55] for the difference between Mean Average Precision using T_{bbn} and $T_{bbn+assert}$. For complexity levels greater than zero, use of $T_{bbn+assert}$ yields statistically significant (at the 0.01 level) improvements in MAP, except for levels 9 and 10 in the test data, for which there are too few examples to show significance.

Figure 5.6 shows the average recall at rank 200 for queries at different levels of complexity. The confidence intervals were constructed by simulating the distribution of average recall by averaging samples from a beta distribution fitted to the observed recall values using the method of moments estimator. The figure shows that $T_{bbn+assert}$ provides better recall when queries are more complex.

The results of these experiments demonstrate that the additional element and relation types represented by $T_{bbn+assert}$ provide superior recall of relevant sentences at higher ranks compared with the T_{bbn} baseline in the *single structure* case. The more specific queries boost precision at early ranks, but sacrifice recall. The *every structure* case shows us that this recall penalty can be mitigated by evaluating the

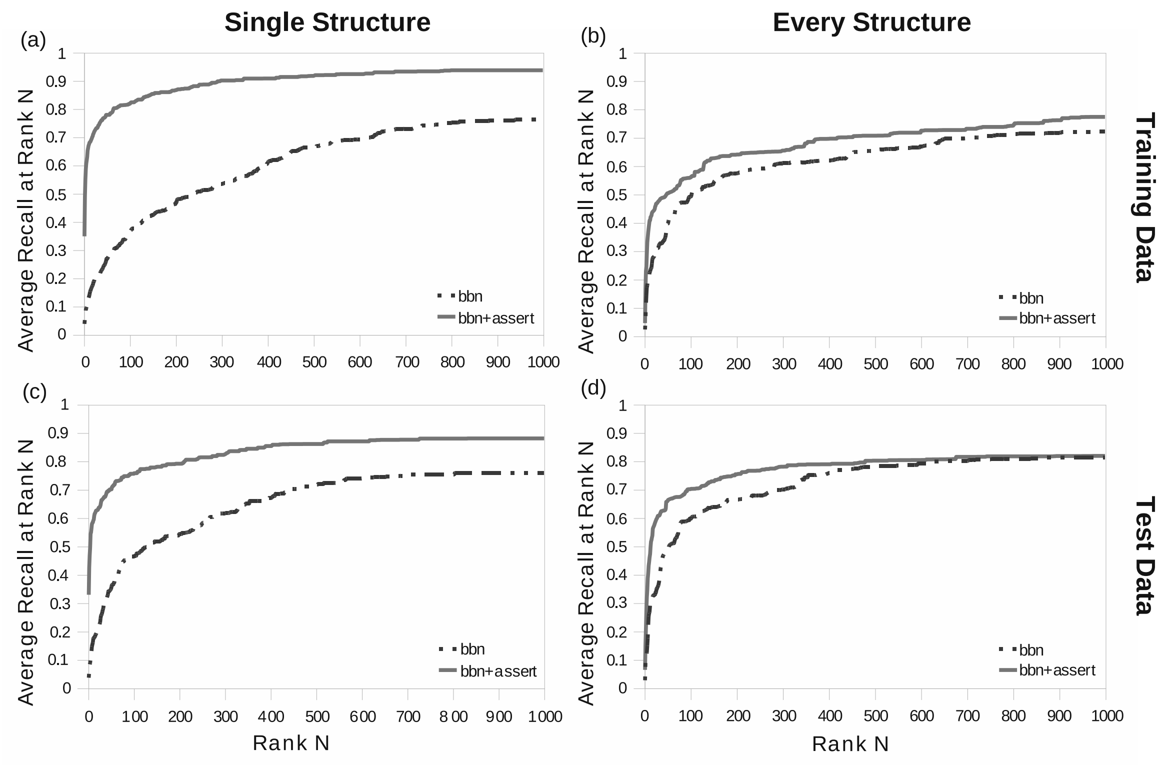

Figure 5.5: Recall of answer-bearing sentences for both T_{bbn} and $T_{bbn+assert}$ type systems, for both the *single structure* and *every structure* cases. Training topics are above and test topics are below.

Table 5.2: Mean Average Precision for *single structure* case by complexity level. The N column gives the number of queries at each complexity level.

Cplxty	Training Data				Test Data			
Level	N	T_{bbn}	$T_{bbn+assert}$	p-value	N	T_{bbn}	$T_{bbn+assert}$	p-value
0	21	0.1282	0.1516	0.3740	11	0.2167	0.2103	1.0000
1	49	0.0565	0.1758	< 0.0001	33	0.0551	0.2001	< 0.0001
2	50	0.0971	0.2973	< 0.0001	54	0.0727	0.2953	< 0.0001
3	46	0.1005	0.3907	< 0.0001	40	0.0567	0.3242	0.0001
4	66	0.1202	0.5788	< 0.0001	44	0.1524	0.6170	< 0.0001
5	40	0.0635	0.6435	< 0.0001	21	0.0795	0.7083	< 0.0001
6	28	0.0499	0.6370	< 0.0001	19	0.0301	0.5648	< 0.0001
7	27	0.0447	0.6199	< 0.0001	10	0.0372	0.8524	0.0021
8	20	0.0144	0.7667	< 0.0001	8	0.1073	1.0000	0.0009
9	9	0.0057	0.8333	0.0044	3	0.0444	1.0000	0.2514
10	11	0.0096	0.9545	0.0009	1	0.0333	0.1429	1.0000

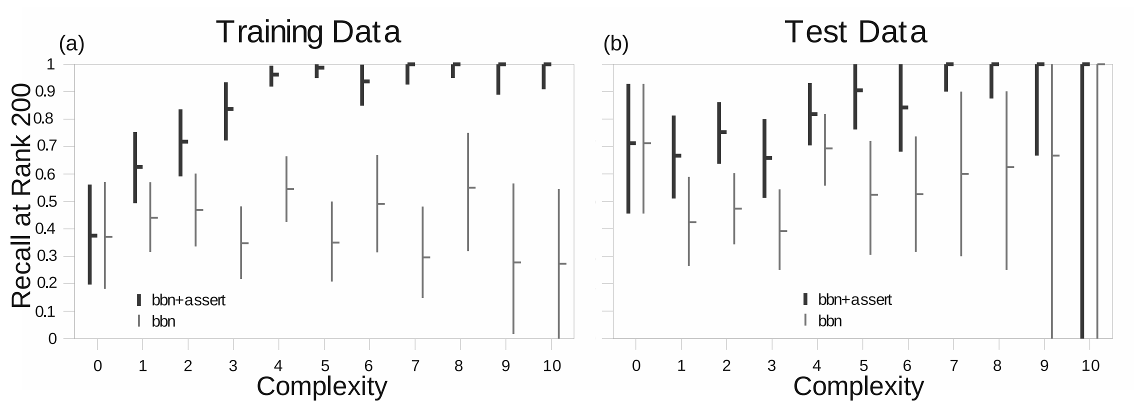

Figure 5.6: Recall at rank 200 for varying complexity of structure in the answer-bearing sentences. Points show the estimate of average recall, with error bars covering a 95% confidence interval.

$T_{bbn+assert}$ queries individually and combining via Round Robin meta-search.

The queries used in these experiments can be thought of as endpoints of a continuum of complexity; the T_{bbn} queries are the least complex, and the fully-specified queries under $T_{bbn+assert}$ are the most complex. The conclusion that `target`-*argument* structure improves retrieval effectiveness when queries are perfectly formulated is a valid one, but one may question whether it is applicable to realistic systems that would have difficulty formulating queries equal in quality to those formulated by a human. This raises the interesting research question of whether partially structured queries situated somewhere in the middle of this continuum would outperform a baseline query with no predicate-argument structure. A QA system may be able to formulate partially-structured queries with greater accuracy, because it is a less difficult task, but the problem is determining which parts of a complete structure should be included in the query, and which should be left out. Later chapters in this thesis explore this very question.

5.6 Surface Patterns

Sentence retrieval based on T_{bbn} is similar to proximity-based or density-based passage-scoring functions favored in the QA community, and is therefore a strong baseline for a QA application. Surface patterns, however, are an even stronger baseline. This section experiments with $T_{surf+bbn}$, a type system defined in Section A.7 capable of representing ordering constraints among *keyterms* as well as *entity* based constraints. The experiments described above are repeated, substituting $T_{surf+bbn}$ for T_{bbn}, in order to compare $T_{bbn+assert}$ against the stronger baseline. Figure 5.7 shows the example sentence discussed above, as represented under $T_{surf+bbn}$. Queries under $T_{surf+bbn}$ are formulated by organizing *keyterms* into a clause of the form `#od(` *keyterm* ... `)`, as shown in Figure 5.8, such that *precedence* relations are respected from left-to-right. Placeholders for *entity* types are included in the enclosing `#combine[sentence]` operator, but outside the `#od` operator. Use of the `#od` version of the operator, as opposed to the `#od`N version, enforces precedence within a window of unlimited size.

Table 5.3 compares T_{bbn}, $T_{surf+bbn}$ and $T_{bbn+assert}$ in terms of Mean Average Precision, including p-values according to Fisher's randomization test [55], which is used to test whether the moves from T_{bbn} to $T_{surf+bbn}$, and from $T_{surf+bbn}$ to $T_{bbn+assert}$, result in statistically significant improvements. For both the training and test sets, $T_{surf+bbn}$ provides a statistically significantly *better quality* ranking at the 0.01 level, when compared to T_{bbn}. Additionally, on the test data, it can be shown that $T_{bbn+assert}$

$$G_{sent,\,surf+bbn} = (E, R, T_{surf+bbn})$$

$$E = \left\{ \begin{array}{l} e_1 = (\texttt{sentence}), \\ e_2 = (\texttt{date}), \\ e_3 = (\texttt{person}), \\ e_4 = (\text{wilt}), \\ e_5 = (\text{chamberlain}), \\ e_6 = (\text{score}), \\ e_7 = (\text{100}), \\ e_8 = (\text{point}) \end{array} \right\}, \quad R = \left\{ \begin{array}{l} (\textit{enclosure},\, e_1,\, e_2), \\ (\textit{enclosure},\, e_1,\, e_3), \\ (\textit{enclosure},\, e_1,\, e_4), \\ (\textit{enclosure},\, e_1,\, e_5), \\ (\textit{enclosure},\, e_1,\, e_6), \\ (\textit{enclosure},\, e_1,\, e_7), \\ (\textit{enclosure},\, e_1,\, e_8), \\ (\textit{enclosure},\, e_3,\, e_4), \\ (\textit{enclosure},\, e_3,\, e_5), \\ (\textit{precedence},\, e_4,\, e_5), \\ (\textit{precedence},\, e_4,\, e_6), \\ (\textit{precedence},\, e_4,\, e_7), \\ (\textit{precedence},\, e_4,\, e_8), \\ (\textit{precedence},\, e_5,\, e_6), \\ (\textit{precedence},\, e_5,\, e_7), \\ (\textit{precedence},\, e_5,\, e_8), \\ (\textit{precedence},\, e_6,\, e_7), \\ (\textit{precedence},\, e_6,\, e_8), \\ (\textit{precedence},\, e_7,\, e_8) \end{array} \right\}$$

Figure 5.7: Pruned annotation graph for the sentence *On March 2, 1962, in Hershey, PA, Wilt Chamberlain scored 100 points against the Knicks* under the type system $T_{surf+bbn}$. Note the addition of the *precedence* relations.

```
#combine[sentence]( #any:date
   #od( wilt chamberlain score 100 point ) )
```

Figure 5.8: Query formulated under type system $T_{surf+bbn}$ for the sentence, *On March 2, 1962, in Hershey, PA, Wilt Chamberlain scored 100 points against the Knicks.*

is statistically significantly better than $T_{surf+bbn}$ at the 0.01 level.

Table 5.3: Comparison of Mean Average Precision among T_{bbn}, $T_{surf+bbn}$ and $T_{bbn+assert}$ for the *every structure* case

	Training Data ($N = 55$)		Test Data ($N = 54$)	
Type System	MAP	p-value	MAP	p-value
T_{bbn}	0.1417	–	0.1613	–
$T_{surf+bbn}$	0.2612	0.0002	0.2484	0.0088
$T_{bbn+assert}$	0.2922	0.1485	0.3423	0.0016

Figure 5.9 shows a comparison between the three type systems $T_{bbn+assert}$, $T_{surf+bbn}$, and T_{bbn}. $T_{surf+bbn}$ enjoys a clear advantage over T_{bbn} for the training data. That advantage fades by approximately rank 250 for the test data, but $T_{surf+bbn}$ still retrieves more relevant items at low ranks than T_{bbn} does. This is consistent with Table 5.3, as relevant items retrieved at low ranks boosts Mean Average Precision.

5.7 Degraded Annotation Quality

The section explores the effectiveness of the structured retrieval approach to linguistic and semantic passage retrieval for QA in a scenario in which the accuracy of the NLP analysis tools can not be guaranteed. To examine the difference in performance as NLP accuracy varies, it is necessary to select a text collection for which there exist gold-standard analyses.

This experiment uses a portion of the Penn Treebank containing one million words of Wall Street Journal (WSJ) text published in 1989 [33]. The text is hand-labeled for sentence boundaries, parts-of-speech and syntactic structure. The PropBank annotation project [27] contains hand-annotated verb predicate-argument structures and semantic role labels for verb arguments over this text collection.

For comparison, a version of the WSJ text with lower-accuracy semantic role labeling markup is constructed using the output of the ASSERT tool. Although ASSERT was trained on the PropBank data, it appears to be only 88.8% accurate in terms of number of arguments correctly identied and labeled. Gold-standard syntactic analysis was not made available to ASSERT as it annotated the WSJ text, so some portion of the errors are undoubtedly due to ASSERT's underlying syntactic parser.

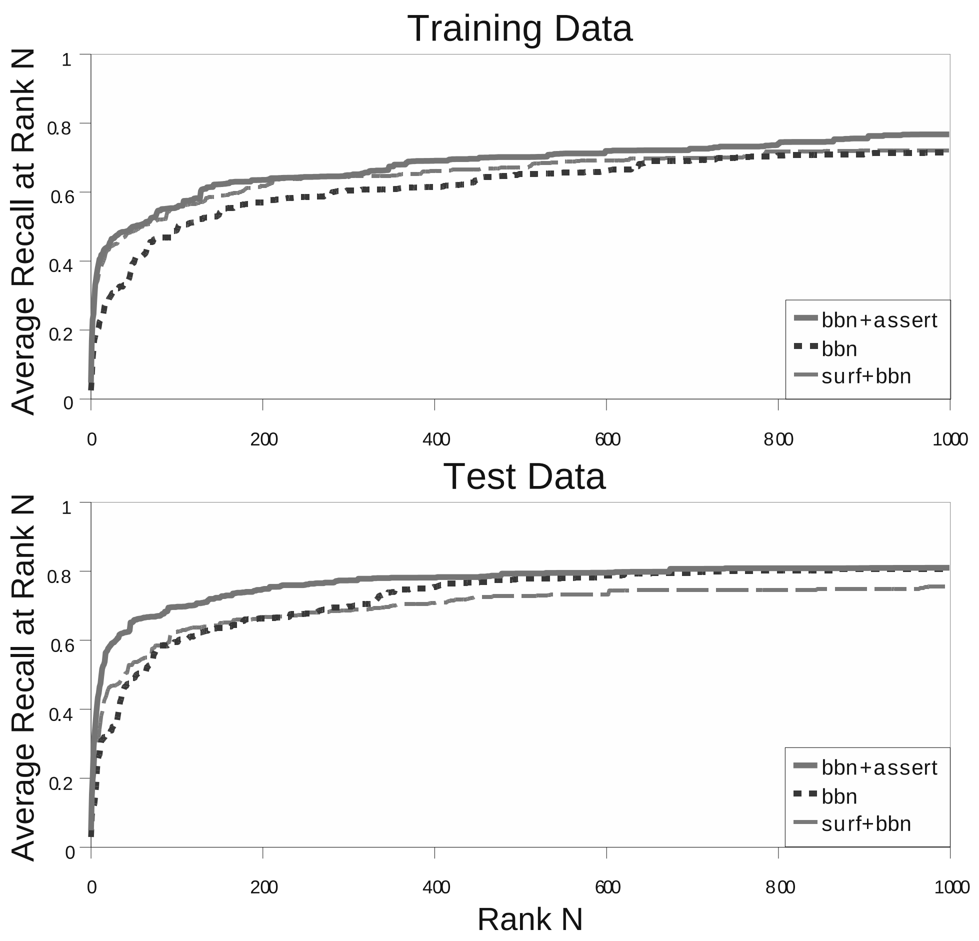

Figure 5.9: Average recall of answer-bearing sentences for type systems $T_{bbn+assert}$, $T_{surf+bbn}$, and T_{bbn}. Training topics are shown above, and test topics, below.

There are no sets of questions with associated relevance judgments readily available for use with the WSJ text collection, but it is small enough that it is possible to use an automatic procedure to generate an exhaustive list of all questions answerable by PropBank-style verb predicate-argument structures, along with their associated sentence-level relevance judgments.

For each sentence in the WSJ data, PropBank [27] contains zero or more hand-annotated predicate-argument structures, each of which contains zero or more arguments. These structures were grouped first by the predicate verb, and then by combinations of arguments shared in common. Each group is defined by a particular verb, and a particular set of arguments. A sentence containing s predicate-argument structures, where the i-th structure has a_i arguments, will appear in g groups:

$$g = \sum_{i=1}^{s} \sum_{j=1}^{a_i} \binom{a_i}{j}$$

Consider the following predicate-argument structure from document WSJ 0427, expressed in ASSERT's notation:

[`arg0` *Dow Jones*] [`target` *publishes*] [`arg1` *The Wall Street Journal, Barron's magazine, other periodicals and community newspapers*]

This sentence's two-argument structure puts it into three groups corresponding to the questions it can answer:

1. [`arg1` *What*] *does* [`arg0` *Dow Jones*] [`target` *publish*]?
2. [`arg0` *Who*] [`target` *publishes*] [`arg1` *The Wall Street Journal, Barron's magazine, other periodicals and community newspapers*]?
3. *Does* [`arg0` *Dow Jones*] [`target` *publish*] [`arg1` *The Wall Street Journal, Barron's magazine, other periodicals and community newspapers*]?

These groups also contain other sentences that have similar structures. For example, Group 1 contains two structures also having the `target` *publish* and the `arg0` *Dow Jones*. Once the grouping is complete, each group contains only and all of the sentences in the corpus containing predicate-argument structures that answer a particular question. Each group, then, constitutes exhaustive sentence-level relevance judgments for its corresponding question. Of the questions generated by this method, 96.3% had only one relevant sentence. These questions were discarded, as they would have skewed the averages, leaving 10,690 questions with associated judgments .

This experiment measures the difference in Mean Average Precision between the structured retrieval approach and the baseline, when ASSERT-generated labels are substituted for the gold-standard PropBank labels. ASSERT can omit, mis-label or incorrectly identify the boundaries of an annotation. As an example, consider the following gold-standard PropBank structure from WSJ_0003:

[`arg1` *A form of asbestos*] [`argm-tmp` *once*] [`target` *used*] [`arg2-pnc` *to make Kent cigarette filters*]

The corresponding ASSERT output is:

[`arg0` *A form of asbestos*] [`argm-tmp` *once*] [`target` *used*] *to make Kent cigarette filters*

ASSERT has fundamentally altered the meaning of the annotation by considering *a form of asbestos* to be the `arg0`, or agent, of the verb used as opposed to the `arg1`, or patient. ASSERT has also missed the `arg2-pnc` (purpose, not cause). Queries are formulated under T_{bow} and T_{assert} as defined in Sections A.1 and A.5, respectively. Each set of queries is evaluated on the WSJ data with PropBank labels and with ASSERT labels.

Table 5.4: Comparison of Mean Average Precision for T_{assert} and T_{bow} retrieval approaches with PropBank (gold-standard) and ASSERT (degraded) labels.

Approach	Labels	MAP	% Increase over baseline	p-value (N = 10,690)
Baseline T_{bow}	–	0.7684	–	–
T_{assert}	ASSERT	0.8698	13.19%	< 0.0001
T_{assert}	PropBank	0.9581	24.68%	< 0.0001

Table 5.4 compares the T_{assert} and T_{bow} retrieval approaches in terms of Mean Average Precision, separately for PropBank and ASSERT labels. The table includes p-values according to Fisher's randomization test [55]. The accuracy of the baseline T_{bow} approach is not affected by degraded annotation quality, since it completely ignores them when ranking results. It is clear that the T_{assert} approach is somewhat sensitive to the quality of the labels, but it still manages to produce a statistically significant improvement in MAP at the 0.0001 level over T_{bow}.

5.8 Conclusions

The structured retrieval approach proposed in this chapter was shown to significantly outperform a bag-of-words with named entity baseline in terms of Mean Average Precision, and to compare favorably against the baseline in terms of recall. The method was shown to be particularly effective for the most complex questions, those having the most linguistic and semantic constraints. It was shown that methods utilizing `target`-*argument* information from ASSERT are sensitive to the quality of the annotation of corpus.

Structured retrieval shows genuine promise as an approach to high-precision linguistic and semantic passage retrieval for QA systems, but the question remains as to whether it is reasonable to expect that a QA system will be able to predict *a priori* which of the linguistic and semantic constraints in its information need are the most important for retrieving answer-bearing passages. Choosing incorrectly would result in missing relevant text that fails to satisfy a certain constraint at best, or at worst, retrieving text that supports incorrect answers.

Errors in Question Analysis can result in the formulation of structured queries that do not match any structure in the text collection. These types of queries do not lend themselves to partial matching of the linguistic and semantic constraints encoded within, even though they may be near misses that fail to match on a single term. When structured queries do not match completely, retrieval quality degrades below the level of queries formulated under T_{bbn}, even though answer-bearing sentences exist that may be good matches for the queries on the basis of constraints involving *keyterms* and *entities* alone. Hope for better partial matching of structured queries lies in more recent work on structured retrieval models, including field-specific smoothing, modeling structural mismatch between query and sentence, and different methods of evidence combination [66, 67].

For QA system developers, it is important to note that the high precision numbers obtained in the experiments in this chapter will not immediately translate to other applications, because the experiments relied on unrealistically high-quality queries. To maximize the performance of the structured retrieval approach, a system must be able to successfully predict linguistic and semantic features that correlate highly with answer-bearing text, based on those of the question. The chapter provides no solution to this issue, yet assumes one exists in order determine whether structured retrieval techniques are appropriate for use in QA systems. The work presented here represents the best case assumption for Question Analysis quality, and should be considered an upper bound for retrieval effectiveness.

The strong assumptions required to get the best performance from the structured retrieval method, combined with the observation that querying linguistic and semantic constraints individually may perform better than querying large groups simultaneously, lead naturally to consideration in subsequent chapters of a rank-learning approach that uses features based on linguistic and semantic constraints individually and in small groups. The training component of such an approach would relieve QA system developers from having to decide *a priori* which of the constraints are most useful for passage ranking.

Chapter 6

Learning-to-Rank

This chapter studies a rank-learning approach to passage retrieval for Question Answering (QA) that is able to leverage linguistic and semantic features. The approach is designed to provide greater functionality than, and to address several of the weaknesses of, the passage retrieval strategy based on structured retrieval techniques described in the previous chapter.

Experiments in this chapter test the hypothesis that the rank-learning passage retrieval approach, using $T_{bbn+assert}$ as introduced in the previous chapter, can retrieve more relevant passages and rank them more highly, which constitutes a *better quality* passage ranking compared to a standard baseline, for the particular test collection under consideration. It is further hypothesized that the rank-learning approach compares favorably to the structured retrieval method proposed in the previous chapter.

6.1 Learning-to-Rank for Question Answering

Consider a QA system based on $T_{bbn+assert}$ as defined in Section A.6. The information need annotation graph produced by the Question Analysis module is a rich source of linguistic and semantic information that can be used to inform passage retrieval. The structured retrieval method presented in the previous chapter maps information need annotation graphs under $T_{bbn+assert}$ into complex Indri queries.

In contrast, the rank-learning approach proposed in this chapter decomposes the information need into atomic linguistic and semantic constraints, rather than using the annotation graph as a whole. These atomic constraints are used as features in a trained Committee Perceptron model [16], which is used to re-rank the output from a baseline retrieval strategy, consisting of Indri queries formulated under T_{bbn}, as described in Section 5.4.1. As explained in Section 4.4.2, the Committee Perceptron model is trained using pairwise preference relations between known relevant and non-relevant passages in the training data.

Learning-to-rank techniques are particularly applicable to the task of passage retrieval for QA for several reasons. In contrast to the structured retrieval approach studied in the previous chapter, rank-learning techniques allow for the linguistic and semantic constraints of interest to the QA system to be checked individually and in small groups, as opposed to all at once. Furthermore, each feature will be individually weighted, which can enable a more fine-grained partial matching functionality.

Another advantage of rank-learning techniques is that it is not necessary to determine *a priori* which constraints, or groups of constraints, are the most predictive of answer-bearing passages; this information comes naturally from the training process. If retrieval model scores for consistent baselines, such as bag-of-words retrieval, are included as features, the training process will emphasize them just enough to ensure that the ranking function will degrade to the baseline in the absence of higher-level linguistic and semantic constraints.

6.2 Selecting Constraints for a Type System

The rank-learning method proposed in this chapter relies on features derived from atomic linguistic and semantic constraints to rank passages. These features come from a decomposition of the annotation graph representing the information need according to a set of constraints specific to an individual type system. The QA system developer selects the constraints using a semi-automatic process described in this section, which must be performed once for every QA problem instance that relies on a new type system.

6.2.1 Automatic Enumeration of Constraints

The first step of the process is an algorithm that enumerates all possible constraints based on the definitions of element and relation types in the type system. A constraint can be thought of as a snippet of an annotation graph containing elements and relations between them. A first-order constraint contains two elements joined by a single relation. The first order constraints that a type system supports come directly from the relation type definitions in the type system. The algorithm, shown in Table 6.1, builds higher-order constraints by extending lower-order constraints, adding a single element and relation each time.

Consider the type system $T_{bbn+assert}$, and its definition, which is given in Section A.6. There are eight relation types defined, so step 2 of the algorithm initializes set C_1 with 56 first-order constraints by substituting leaf element types for *annotation*, *argument* and *entity*.

The main loop of the algorithm begins at step 3. At each iteration, the algorithm builds a set of order-i constraints based on the constraints of order $i-1$. The algorithm iterates through the set of relation type definitions Tr, and then loops through the constraints in C_{i-1}. For each element in the constraint of order $i-1$ that is equal to or inherits from the domain or range element type of the current relation type definition, the constraint is augmented by the relation to form a new order-i constraint. In addition, a new element, having a leaf type that inherits from the respective range or domain element type for that relation, is introduced as the appropriate target or source for the new relation.

To illustrate this process by example, consider the following first-order constraint, which contains a single *enclosure* relation between an `arg0` and a `person`:

$$(E = \{(a_0, \texttt{arg0}), (p, \texttt{person})\}, R = \{(enclosure, a_0, p)\}, T_{bbn+assert})$$

To build a second-order constraint, the algorithm can introduce an *attachment* relation with a `target` as its source:

$$\begin{aligned}&(E = \{(t, \texttt{target}), (a_0, \texttt{arg0}), (p, \texttt{person})\},\\ &R = \{(attachment, t, a_0), (enclosure, a_0, p)\}, T_{bbn+assert})\end{aligned}$$

The algorithm also introduces a constraint for each element e the `arg0` can enclose, which can be a `target`, as shown below, or any sub type of *argument* or *entity*, for a total of 15 new second-order constraints:

$$\begin{aligned}&(E = \{(a_0, \texttt{arg0}), (p, \texttt{person}), (e, \texttt{target})\},\\ &R = \{(enclosure, a_0, p), (enclosure, a_0, e)\}, T_{bbn+assert})\end{aligned}$$

Two new second-order constraints can be constructed because both the `arg0` and the `person` element can enclose a *keyterm*. Constraints are only lexicalized given a specific information need, so during the enumeration process, *keyterm* is considered a leaf type. It can be thought of as a wildcard having the semantics that it matches any sub-element type of *keyterm* found in the information need. Here, `arg0` is shown participating in an *enclosure* relation with a *keyterm*:

$$\begin{aligned}&(E = \{(a_0, \texttt{arg0}), (p, \texttt{person}), (k, \mathit{keyterm})\},\\ &R = \{(\mathit{enclosure}, a_0, p), (\mathit{enclosure}, a_0, k)\}, T_{bbn+assert})\end{aligned}$$

The final group of 22 new second-order constraints are built by introducing an enclosing element for `arg0` or `person`, both of which can be enclosed by a `sentence` or any leaf element type inheriting from *argument*. This constraint enforces *enclosure* of the `arg0` within a `sentence`:

$$\begin{aligned}&(E = \{(s, \texttt{sentence}), (a_0, \texttt{arg0}), (p, \texttt{person})\},\\ &R = \{(\mathit{enclosure}, s, a_0), (\mathit{enclosure}, a_0, p)\}, T_{bbn+assert})\end{aligned}$$

The example shows that just one of the 56 first-order constraints allowable under $T_{bbn+assert}$ yields a 40 second-order constraints constructed by adding a single relation and element to the original constraint. The algorithm repeats this process for the remaining 55 first-order constraints before turning its attention to the second-order constraints. Clearly, for all but the smallest type systems, in terms of number of element and relation type definitions, this algorithm will enumerate a large number of higher-order constraints.

6.2.2 Generalized Constraints

An additional, but potentially powerful, class of constraints not enumerated by the algorithm described above are known as generalized constraints. The constraints discussed thus far enforce specific types of elements with specific types of relations between them, with the exception of *keyterm*. In contrast, generalized constraints specify two or more *keyterms*, but refer to the information need for the relations and intermediate elements that associate them. One example of a generalized constraint is **Paths(*N*)**, described in the Section 6.3. This constraint is satisfied if a passage annotation graph contains a pair of *keyterms* from the information need, such that they are related by the same path of N relations in the passage as they are in the information need.

Table 6.1: Algorithm for Enumeration of Constraints for a Type System

Input:
Type system $T = (Te, Tr)$
Maximum order n (to generate constraints of order 1 through order n)

Output:
Set of sets of constraints, one for each order $\{C_1, \ldots, C_n\}$

```
1. Initialize $C_{1 \ldots n} = \emptyset$

2. For each $tr = (name, domain, range) \in Tr$:

        For each $d, r \in Te$ such that $d$ and $r$ are concrete leaf types inheriting from
        $domain$ and $range$, respectively:
            Add $(E = \{(e_1, d), (e_2, r)\}, R = \{(name, e_1, e_2)\}, T)$ to $C_1$

3. For $i = 2, \ldots, n$:

        For each $tr = (name, domain, range) \in Tr$:
            For each constraint $c = (E_c, R_c, T) \in C_{i-1}$:
                For each element $e \in E_c$ of type $domain$ or a subtype:
                    For each $r \in Te$ such that $r$ is a concrete leaf type inheriting from
                    $range$:
                    Add to $C_i$: $(E = E_c \cup \{(x, r)\}, R = R_c \cup \{(name, d, x)\}, T)$.
                For each element $r \in E_c$ of type $range$ or a subtype:
                    For each $d \in Te$ such that $d$ is a concrete leaf type inheriting from
                    $domain$:
                    Add to $C_i$: $(E = E_c \cup \{(x, d)\}, R = R_c \cup \{(name, x, r)\}, T)$.
```

Note: *keyterm* is always considered a leaf type for the purposes of this algorithm.

6.2.3 Constraint Selection

Once the universe of constraints has been enumerated using the automatic process and generalized constraints, if desired, have been defined, the final task remaining for the QA system developer is to decide which of the constraints to use. For the smallest type systems, it may be appropriate to use all of the constraints, but for larger type systems, it may be advantageous to reduce the space of constraints by, for example, removing constraints that are redundant or sparsely predictive. There are a number of well-known methods for feature selection, such as Chi-square or mutual information, that apply naturally to this problem. Intuition plays a role as well. For example, under $T_{bbn+assert}$, every sentence has a `sentence` tag. Therefore, any higher-order constraint checking any type of *enclosure* relation involving the `sentence` can be removed immediately because it is redundant; there is another constraint in the universe, of one degree lower order, that is always satisfied if and only if the redundant constraint is satisfied.

6.3 Linguistic and Semantic Constraints for $T_{bbn+assert}$

This section describes the constraint types selected for $T_{bbn+assert}$ for use in the experiments presented in this chapter. The procedure described in the previous section was employed, automatically enumerating all constraints up to order 3, then applying the following principles, based on intuition, to narrow the result down:

1. **An element may participate in at most one *enclosure* relation with another non-*keyterm* element.**

   This principle eliminates the redundancy inherent in constraints that specify an element enclosing, or being enclosed by, more than one other element. The principle also prohibits transitive enclosure, which is a reasonable thing to do because the NLP tools underlying $T_{bbn+assert}$ enforce transitivity of enclosure. Therefore, all enclosure information is captured sufficiently by the first-order constraints consisting of a single *enclosure* relation between two elements. The exclusion for *keyterms* is required for proper *keyterm precedence* constraint-checking, which requires that a pair of *keyterms* be mutually enclosed in an *annotation.*

2. **When *attachment* relations are present between `targets` and `arguments`, consider only *keyterm enclosure.***

Intuition suggests that the *attachment* relations and the `target` and `argument` *annotations* can be used to understand the long-distance semantic dependencies between *keyterms*. While there may also be interesting semantic dependencies between *keyterms* and *entities*, the decision was made to avoid proliferation of constraints mixing *attachment* relations and *entity enclosure* under the assumption that first-order constraints checking *argument-entity* and *entity-keyterm enclosure* would be sufficient to capture these dependencies.

3. **No nesting of predicate-argument structures.**

   Though this is allowed by the underlying NLP tools, the experiments in this chapter ignore the possibility of *enclosure* of `targets` and other *arguments* inside *arguments*. Nested `targets` and *arguments* are treated as if they are at the `sentence` level.

4. **For *keyterm precedence*, *keyterms* must be enclosed in the same *annotation*.**

   Even though the type system technically admits this, it would not make sense to check *precedence* for *keyterms* enclosed in different *annotations*, or not enclosed in any *annotation*.

After narrowing the space of constraints, the following first-order constraints remained:

- **KEnc(*annotation*)**: Keyterm Enclosure within an Annotation

  $$(E = \{(a, annotation), (k, keyterm)\}, R = \{(enclosure, a, k)\}, T_{bbn+assert})$$

  Satisfaction of this constraint by a passage involves matching an *enclosure* relation between an instance of the specified *annotation*, and any *keyterm* in the information need also enclosed by an instance of that same *annotation*. There are 16 constraints of this type, one corresponding to each leaf sub-element type of *annotation*.

- **AEnc(*annotation*, *annotation*)**: Annotation-Annotation Enclosure

  $$(E = \{(a_1, annotation), (a_2, annotation)\}, R = \{(enclosure, a_1, a_2)\}, T_{bbn+assert})$$

This constraint checks for *enclosure* relations between instances of the specified *annotation* types in the passage annotation graph, and can be checked for any pair of *annotations* for which an *enclosure* relation is defined in $T_{bbn+assert}$. There are 15 **AEnc(** `sentence`**,** ***annotation*** **)** constraints, as `sentence` can enclose any other type of *annotation*. There are 40 **AEnc(** ***argument*****,** ***entity*** **)** constraints.

- **Att(** ***annotation*****,** ***annotation*** **)**: Attachment between Annotations

$$(E = \{(t, \texttt{target}), (a, argument)\}, R = \{(attachment, t, a)\}, T_{bbn+assert})$$

  Under $T_{bbn+assert}$, *attachment* can only hold between a `target` and an *argument*. This constraint is important because there can be multiple `targets` in a passage, and it is also possible for a `target` to have multiple *arguments* attached to it with the same role label. There are 10 of these constraints, one for each leaf sub-element type of *argument*.

The automatic procedure enumerated several interesting candidates higher-order constraints, though several were rejected on the basis of intuition that they would be redundant. One example of a redundant higher-order constraint is the order-n constraint consisting solely of *attachment* relations. This was rejected on the grounds that the first-order single *attachment* constraints could capture this information. Some additional second-order constraints were not used because they are subsumed by the third-order constraints shown below:

- **KPrec(** ***annotation*** **)**: Keyterm Precedence within an Annotation

$$(E = \{(e, annotation), (k_1, keyterm), (k_2, keyterm)\},\\ R = \{(enclosure, e, k_1), (enclosure, e, k_2), (precedence, k_1, k_2)\}, T_{bbn+assert})$$

  This constraint is composed of two *enclosure* relations between the *annotation* and each *keyterm*, and a single *precedence* relation between the *keyterm* pair. There are 16 constraints of this type.

- **Att-KEnc2(** ***annotation*****,** ***annotation*** **)**: Attachment, with double Keyterm Enclosure

$$(E = \{(t, \texttt{target}), (a, argument), (k_t, keyterm), (k_a, keyterm)\},$$
$$R = \{(attachment, t, a), (enclosure, t, k_t), (enclosure, a, k_a)\}, T_{bbn+assert})$$

This constraint represents a specific type of relationship between two *keyterms*, in which each participates in an *enclosure* relation with an instance of one each of the specified *annotation* types, which are connected through an *attachment* relation. Note that a passage can satisfy all of **KEnc(** `target` **)**, **KEnc(** `arg1` **)** and **Att(** `target, arg1` **)** without satisfying **Att-KEnc2(** `target, arg1` **)**, which additionally requires that the `target` and `arg1` instances be the same throughout.

The following generalized constraints were introduced to capture features specific to a particular information need.

- **Ans**: Expected Answer Type

  $$(E = \{(s, \texttt{sentence}), (e, entity)\}, R = \{(enclosure, s, e)\}, T_{bbn+assert})$$

  This constraint is similar to **AEnc(** `sentence`, ***entity*** **)**, except that the *entity* type is not specified. Instead, the expected answer type of the question, as specified in the information need, is used. This constraint will fail to match if information need does not specify an expected answer type that is a type of *entity*. The process that fills in the particular *entity* type for this constraint is similar to the lexicalization of the *keyterm* placeholders in the other constraints that occurs when a constraint meets an information need annotation graph for a given question.

- **ExpAtt(** ***annotation*, *N*** **)**: Coverage of Expected Attached Annotations

  $$(E = \{(t, \texttt{target}), (a_1, argument), \ldots, (a_N, argument)\},$$
  $$R = \{(attachment, t, a_1), \ldots, (attachment, t, a_N)\}, T_{bbn+assert})$$

  This constraint checks for the presence of a `target` with the same N *arguments* attached as are specified in the information need. The specific argument roles are only specified given an information need corresponding to a particular question.

- **Paths(*N*)**: Coverage of Expected Keyterm Paths

  $(E = \{(a_1, annotation), (a_2, annotation), (k_1, keyterm), (k_2, keyterm), \ldots\},$
  $R = \{(enclosure, a_1, k_1), (enclosure, a_2, k_2), \ldots\}, T_{bbn+assert})$

  This constraint is the most powerful among those that represent long-distance semantic relationships between *keyterms* because it considers all paths through the annotation graph between a pair of *keyterms*. A path in this case refers to an acyclic traversal of *enclosure* and *attachment* relations and *entity*, *argument* and `target` elements between a_1 and a_2, which enclose the respective *keyterms*. The parameter N represents the length of the path in terms of *enclosure* and *attachment* relations, not including *keyterm enclosure*. Note that a path of length zero means $a_1 = a_2$. There may be more than one path of any given length that relates a pair of *keyterms*.

The following additional fifth-order constraint was suggested by intuition, because the automatic process was not run to order 5:

- **Att2-KEnc3(*annotation*, *annotation*, *annotation*)**: Double Attachment, with triple Keyterm Enclosure

  $(E = \{(t, \texttt{target}), (a_1, argument), (a_2, argument),$
  $(k_t, keyterm), (k_1, keyterm), (k_2, keyterm)\},$
  $R = \{(attachment, t, a_1), (attachment, t, a_2), (enclosure, t, k_t),$
  $(enclosure, a_1, k_1), (enclosure, a_2, k_2)\}, T_{bbn+assert})$

  This constraint is based on **Att-KEnc2(*annotation*, *annotation*)**, except that it adds a second `attachment` relation to a different *annotation* type, which must enclose a *keyterm* from the information need.

The selected linguistic and semantic constraint types can be used to decompose an annotation graph representing an information need and extract features useful for ranking. It is at this point that *keyterm* placeholders in the constraint types become lexicalized. As an example of this process, Figure 6.1 shows how the information need annotation graph for question 1398, *What year was Alaska purchased?*, decomposes into constraints. For clarity, the figure shows only one constraint of each type, identified by number below:

1. **KEnc(** `sentence` **)**: This constraint is satisfied by any passage having a `sentence` that participates in an *enclosure* relation with a *keyterm year*, *Alaska* or *purchased*.

2. **KPrec(** `sentence` **)**: For a passage to satisfy this constraint, it must have at least two of the three *keyterms year*, *Alaska* or *purchased*, occurring in the same order and enclosed within the same `sentence` instance.

3. **AEnc(** `sentence`**,** `location` **)**: This constraint is satisfied if the passage contains a `sentence` participating in an *enclosure* relation with a `location`.

4. **Att(** `target`**,** `arg1` **)**: This constraint checks for an *attachment* relation holding between a `target` and an `arg1`.

5. **Ans**: This constraint is equivalent to **AEnc(** `sentence`**,** `date` **)**, enforcing *enclosure* between the `sentence` and the *entity* type that is the expected answer type specified in the information need, which in this case, is `date`.

6. **ExpAtt(** `target`**,** 2 **)**: This constraint checks for a `target` having the same two *arguments* attached as in the information need, which in this case, are `arg1` and `argm-tmp`.

7. **Att-KEnc2(** `target`**,** `arg1` **)**: For a passage to satisfy this constraint, it must contain a `target` attached to an `arg1`, each of which must enclose a *keyterm* that is enclosed in the same *annotation* in the information need. Here, the only options are *purchased* and *Alaska*, so this constraint enforces that *Alaska* is the thing purchased, as opposed to the buyer.

8. **Att2-KEnc3(** `target`**,** `arg1`**,** `argm-tmp` **)**: Similar to the above, this constraint is checking for a `target` attached to both an `arg1` and an `argm-tmp`, each of which must match a *keyterm* against the information need. Note that any passage that satisfies this constraint also satisfies **Att-KEnc2(** `target`**,** `arg1` **)** and **Att-KEnc2(** `target`**,** `argm-tmp` **)**.

9. **Paths(** 2 **)**: This generalized constraint is satisfied by any path traversing *entity*, *argument* and `target` elements along *enclosure* and *attachment* relations between a pair of *keyterms*, so long as it matches the information need. The constraint shown in Figure 6.1(9) is just one of many paths not covered by other constraints.

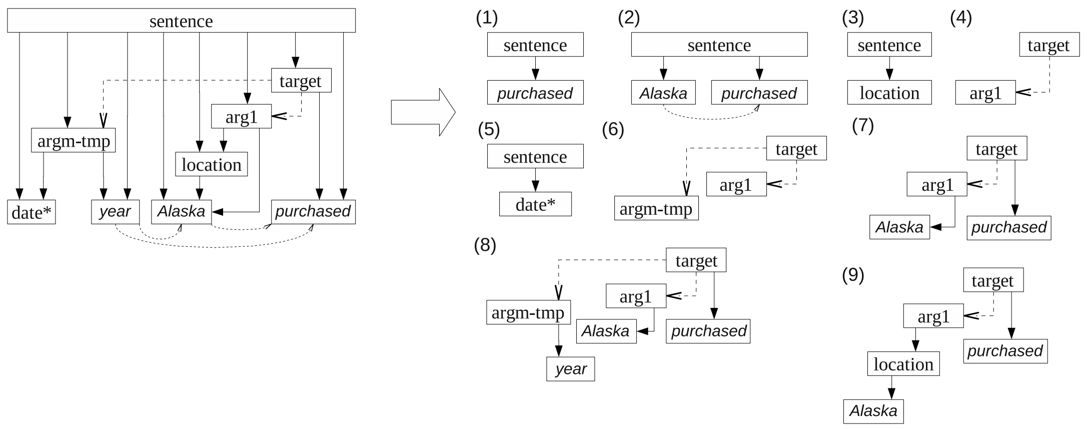

Figure 6.1: Example of decomposition of the information need for the question, *What year was Alaska purchased?*, into annotation graph snippets representing atomic linguistic and semantic constraints. One example constraint of each type is shown.

6.4 Constraint Satisfaction and Feature Extraction

To determine the degree to which a passage satisfies the linguistic and semantic constraints expressed in an information need, the information need is decomposed according to the constraint types selected for the type system in use. All of the linguistic and semantic constraints shown in Figure 6.1 consist of small annotation graph components. Each constraint is compared against the passage annotation graph to determine if it is satisfied. Despite the fact that the constraint types are query-independent, some of the matching is query-specific; *keyterm* elements are lexicalized and normalized morphologically, and generalized constraints are instantiated, according to the specific elements and relations in the information need. A passage either wholly satisfies a constraint or does not satisfy it at all. In aggregate, the count of satisfied constraints can be used to determine the degree of similarity between the passage and the information need.

Satisfaction of a constraint is based on the concept of *annotation sub-graph alignment*; a constraint is considered satisfied by a passage represented as an annotation graph if there exists an alignment of the constraint sub-graph to the passage annotation graph[1]. In this case, an alignment consists of a mapping between elements

[1]Sub-graph alignment is known to be an NP-complete problem [2], but the problem instances

of the constraint sub-graph and elements of the passage annotation graph such that all mapped elements are of the same type, and all relations that hold between elements in the constraint sub-graph also hold between the mapped elements in the passage graph. Satisfied constraints become features useful for ranking by counting the number of distinct alignments of the constraint to the passage that exist. See Figure 6.2 for an illustration of a constraint graph aligning to a passage annotation graph. In the example shown, there are two distinct sub-alignments, so the feature value according to the constraint **Att(** `target`**,** `arg1` **)** is 2. This method of feature extraction can be thought of as similar to a Tree Kernel, which uses piecewise tree comparison to construct feature vectors [39].

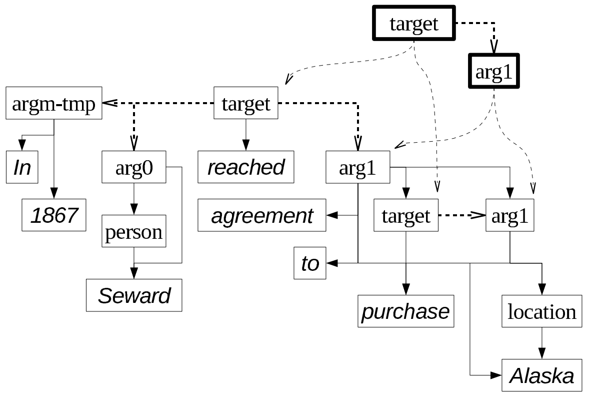

Figure 6.2: Constraint annotation sub-graph (in bold) aligning to the annotation graph for the answer-bearing passage, *In 1867, ... Seward reached agreement ... to purchase Alaska.* The constraint is **Att(** `target`**,** `arg1` **)**, as shown in Figure 6.1(4). Note that there are two distinct alignments of the constraint to the passage. The enclosing `sentence` and the *precedence* relations that exist between *keyterm* pairs are not shown to increase legibility.

6.5 Experimental Methodology

When applying rank-learning methods to the task of passage retrieval for QA, the approach is to use the trained model to re-rank the top results retrieved by a baseline retrieval strategy. The improvement in passage retrieval quality can then be measured by comparing the Mean Average Precision of the baseline approach to that of

are small enough to be tractable for this application.

the re-ranked results. The test collection for these experiments will be the TREC 2002 "MIT 109" test collection, as defined in Section 4.1.1

As in the previous chapter, the baseline passage retrieval approach consists of Indri queries, which are used to retrieve an initial set of up to 1000 sentences for each question. The Indri queries are formulated under T_{bbn} according to the procedure in Section 5.4.1, which includes *keyterms* from the question, as well as the expected named *entity* answer type and any overt *entities* in the question. See Section A.4 for a definition of the T_{bbn} type system, and the upper portion of Figure 5.4 for an example of a query formulated under T_{bbn}.

The learning-to-rank approach evaluated in these experiments consists of a Committee Perceptron model [16] that uses 162 features based on sub-graph alignment counts for the constraint types described above, as well as the baseline Indri score, for a total of 163 features. The complete list of features are given in Table 6.2. Feature definitions are type-system dependent, but do not vary on a per-question basis. Despite this, the range of values a feature may take on can vary from question to question[2], so all feature values are scaled to zero-mean, unit-variance on a per-question basis.

Experiments used 5-fold cross validation, in which the Committee Perceptron model was trained on 4 of the folds, and the trained model was used to re-rank the passages for the questions in the remaining fold. Mean Average Precision values reported are averaged across the five folds. For more information about the Committee Perceptron, including information about training and parameter settings, see Section 4.4.2.

With respect to the baseline T_{bbn} Indri queries, the improvement in passage retrieval quality afforded by the rank-learning strategy was measured separately for eight different groups of features to determine the relative utility of the different types of linguistic and semantic constraints. These eight feature groups are given below, arranged in order of increasing size. The last column of Table 6.2 identifies the features used in each of the following tests.

1. **Keyterms Only** evaluates the effect of the *enclosure* of *keyterms* within the `sentence`.

2. **Surface Patterns** evaluates the effect of the *enclosure* of, and pairwise *precedence* between, *keyterms* within the `sentence`.

[2]One reason for this is that questions have different numbers of *keyterms*.

3. **Named Entities + Keyterms** evaluates the effect of *enclosure* of both *keyterms* and *entities* within the `sentence`.

4. **Named Entities + Surface Patterns** evaluates the effect of *enclosure* of both *keyterms* and *entities*, and of pairwise *keyterm precedence*, within the `sentence`.

5. **Semantic Roles Only** evaluates the effect of the *enclosure* of `targets` and *arguments* within the `sentence`, as well as the *attachment* of *arguments* to their `targets`, without considering *keyterm* information.

6. **Semantic Roles + Keyterms** combines the features in the **Semantic Roles Only** and **Keyterms Only** groups with the *enclosure* relations that occur between `targets` and *keyterms*, and between *arguments* and *keyterms*.

7. **Semantic Roles + Surface Patterns** adds to the previous feature group *precedence* relations between *keyterm* pairs occurring within `sentence`, `target` and *argument annotations.*

8. **Semantic Roles + Named Entities + Surface Patterns** encompasses all constraints described above, in addition to arbitrary long-distance semantic relationships between *keyterm* pairs represented by paths through the annotation graph.

6.6 Experimental Results and Discussion

This section presents and analyzes the results of experiments designed to measure the improvement in passage retrieval quality when rank-learning techniques are used to re-rank baseline retrieval output. Results are given in terms of Mean Average Precision (MAP), averaged over five folds, and are reported for each of the feature groups numbered 1 through 8, individually. For each test, the *p*-value for Fisher's randomization test [55] is given.

Only 48 of the 109 questions in the TREC 2002 "MIT 109" test collection (44.03%) turned out to contain verbs analyzable by ASSERT. ASSERT does not cover certain verbs, including *is*, *do* and *become*, not included in its training data. Said in another way, the information needs corresponding to these questions under T_{bbn} and $T_{bbn+assert}$ are identical, so there is little expectation that re-ranking would

Table 6.2: Learning-to-Rank Features, with group membership. Features are parameterized by leaf element types defined in Section A.6. N identifies the length of the path through the annotation graph

Feature Name	Feature Count	Feature Groups
Baseline Indri Score	1	1-8
KEnc(`sentence` **)**	1	1,2,6-8
KPrec(`sentence` **)**	1	2,4,7,8
KEnc(***entity*** **)**	4	3,4,8
AEnc(`sentence`, ***entity*** **)**	4	3,4,8
Ans	1	3,4,8
KPrec(***entity*** **)**	4	4,8
Att(***argument*** **)**	10	5-8
AEnc(`sentence`, `target` **)**	1	5-8
AEnc(`sentence`, ***argument*** **)**	10	5-8
ExpAtt(`target`, ***N*** **)**	4	5-8
KEnc(`target` **)**	1	6-8
KEnc(***argument*** **)**	10	6-8
Att-KEnc2(`target`, ***argument*** **)**	10	6-8
Att2-KEnc3(`target`, ***argument***, ***argument*** **)**	45	6-8
KPrec(`target` **)**	1	7,8
KPrec(***argument*** **)**	10	7,8
AEnc(***argument***, ***entity*** **)**	40	8
Paths(***N*** **)**	6	8
Total	163	8

improve retrieval quality for these questions. To get an accurate assessment of the impact of re-ranking, it may be instructive to take a closer look at this subset of 48 questions. In the analysis below, this subset will be referred to as the *Deep Structure Questions* when discussing results and observations specific to these questions. The remaining 61 questions will be referred to as *Shallow Structure Questions*.

6.6.1 Full Question Set

For the full set of 109 questions, 5-fold cross validation is performed, with approximately 88/22 train/test queries in each fold. Table 6.3 shows the results on this question set.

Table 6.3: Passage Retrieval Quality using Rank-Learning Techniques on the TREC 2002 "MIT 109" test collection

	MAP	% over Indri T_{bbn}	p-value ($N = 109$)
Indri T_{bbn}	0.1901	–	–
Feature Group 1	0.2076	9.21	0.0057
2	0.2134	12.26	0.0142
3	0.2142	12.68	0.0036
4	0.2170	14.15	0.0582
5	0.2000	5.21	0.0560
6	0.2157	13.47	0.1171
7	0.2156	13.41	0.1061
8	**0.2329**	22.51	0.0332

The learning-to-rank techniques compare favorably against the Indri T_{bbn} baseline, demonstrating the feasibility and effectiveness of a rank learning approach to passage retrieval for QA. The structured retrieval approach discussed in the previous chapter, however, shows a noticeably larger improvement over the same baseline. The reason for the disparity is that the rank-learning method does not have access to the same high-quality question analysis used in the structured retrieval experiments.

The rank-learning method uses features of the question and information need only, without trying to predict features of likely answer-bearing passages, yet still shows improvement over the Indri baseline that is statistically significant at the 0.05 level for feature groups 1 through 3, as well as feature group 8. It can therefore be concluded that the rank-learning approach is robust to the specific types of features used, managing to show some improvement regardless of which features are available.

Of particular interest is how the method is able to use bag-of-words features alone to improve over the baseline. These features tend to moderate the IDF penalty for common terms that is captured by the Indri score, so it may be possible to do parameter tuning for these short queries to mitigate the difference. The best-performing feature group is group 8, which benefits from the inclusion of features based on the powerful **Paths(** $\boldsymbol{N}$ **)** family of generalized constraints.

The semantic role features (groups 5 through 7) show large gains, but the improvements over the baseline are not statistically significant for the full set of questions. This result questions the effectiveness of the semantic role features. In these tests, fewer questions are helped by the trained model, resulting in improvements that are not significant. In feature group 6 for example, the baseline model outperforms the trained model on 26% of the questions, and the trained model outperforms the baseline on 30% of the questions. For the questions that are helped by the trained model, performance is improved by more than 430% on average. For the questions hurt by the trained model, performance is decreased by 35% on average.

Because this net gain in performance is not consistent across questions, an elevated *p*-value is observed. Significance testing attempts to predict whether results from the experimental method and the baseline are drawn from the same, rather than two different, underlying distributions. Significant differences between the two methods are more likely when the experimental method improves performance for more questions by a smaller amount, rather than few questions by a larger amount. Observing this latter behavior, it could be that the difference in the two methods may be a chance occurrence.

An alternative hypothesis is that there exist two distinct sub-populations of questions, those for which re-ranking is helpful, and those for which it is not. As described above, there are many questions for which the ASSERT semantic role labeler fails to provide `target` and *argument annotations*, and as a result, the semantic role feature values are all zeroes. These features, although potentially predictive for some questions, are not predictive for all questions. Combining these two different question populations in the training set may have detrimental effects on the learning algorithm by diluting the "signal" provided by the semantic role features.

To confirm this suspicion, the learned feature weights for the model trained on the full question set can be inspected. Table 6.4 shows the 15 largest (in terms of absolute value) features learned on the full question set, averaged across all five

Table 6.4: Top 15 (in absolute value) mean feature weights across folds, trained on the full feature set and full question set.

Feature Name	Mean Weight
Att-KEnc2(`target`, `arg1` **)**	203.99
Paths(5 **)**	161.90
Paths(2 **)**	138.04
AEnc(`sentence`, `date` **)**	128.02
Ans	113.67
KPrec(`sentence` **)**	94.82
Paths(4 **)**	82.90
Baseline Indri T_{bbn} score	74.20
KEnc(`date` **)**	57.32
KEnc(`org` **)**	55.52
Paths(1 **)**	-69.22
Att2-KEnc3(`target`, `arg1`, `arg2` **)**	-70.28
KEnc(`person` **)**	-78.54
KPrec(`person` **)**	-96.54
Paths(3 **)**	-180.33

cross validation folds. Though some of the semantic role features claimed the largest magnitude weights, 6 of the 15 most influential features are drawn from the surface patterns and named entity feature groups. This is a portrait of the model's difficulty deciding between semantic role features, which are key for certain questions and irrelevant for many, and surface patterns and named entity features, which provide modest help for all questions.

6.6.2 Deep vs. Shallow Question Structure

Across the full question set, disappointing performance for the semantic role features was observed, and the hypothesis was that the behavior could be explained by the fact that relatively few of the questions (48 of 109, or 44.03%) are *Deep Structure Questions* as described above. Because the *Shallow Structure Questions* have no `target` or *arguments*, the semantic role features are always all zeroes, and are useless for ranking passages with respect to the information needs corresponding to these questions. To be able to accurately measure the power of the semantic role features, it is necessary to take a more careful look at the *Deep Structure Questions*.

Table 6.5: Passage Retrieval Quality using Rank-Learning Techniques on the sets of *Deep* and *Shallow Structure Questions*.

	Deep Structure Questions		
	MAP	% over Indri T_{bbn}	p-value ($N = 48$)
Indri T_{bbn}	0.1978	–	–
Feature Group 1	0.2319	17.24	0.0790
2	0.2176	10.01	0.2167
3	0.2067	4.50	0.3605
4	0.2366	19.62	0.0152
5	0.2159	9.15	0.1269
6	0.2694	36.20	0.0723
7	0.2717	37.36	0.0573
8	**0.2788**	40.95	0.0194
	Shallow Structure Questions		
	MAP	% over Indri T_{bbn}	p-value ($N = 61$)
Indri T_{bbn}	0.1845	–	–
Feature Group 1	0.1835	-0.60	0.5039
2	**0.2014**	9.10	0.0951
3	0.1902	3.03	0.2761
4	0.1864	0.98	0.4619
5	0.1846	0.00	1.0000
6	0.1831	-0.81	0.5097
7	0.1858	0.65	0.4691
8	0.1869	1.25	0.4531

Table 6.5 reports the results for the model trained and tested on the *Deep Structure* (top half) and *Shallow Structure* (bottom half) sets separately. As before, these tests report Mean Average Precision figures averaged across the five folds.

From the top half of Table 6.5, it can clearly be seen that, for the *Deep Structure Questions*, the semantic role features have a noticeable positive impact on passage retrieval performance, realizing over 35% improvements in Mean Average Precision when those features are used in combination with keyterm information. For feature group 8, the improvement is statistically significant at the 0.05 level, but is not able to reach the 0.01 level of significance, likely due to the reduced sample size. Feature group 5, which uses semantic role features alone, shows less of an improvement because the semantic role features are most useful when describing linguistic and semantic relationships among keyterms that are not implied by surface patterns alone.

For the *Shallow Structure Questions* the picture is vastly different. In the bottom half of Table 6.5, the only features that result in moderate improvement are the surface patterns. As expected, feature group 5 shows zero improvement, because for these questions, the feature values in this group other than the Baseline Retrieval Score are always zeroes. What is interesting about these questions is that named entities are of limited use in re-ranking the passages, as the baseline Indri queries already capture much of the information provided by these features. Much of the small variations in performance across feature groups 4-7 are attributable to random sampling of the passage-pairs during the training process.

Looking at the features that are assigned the highest weight by the learning algorithm, there is a distinct shift towards strongly favoring semantic role features for the *Deep Structure Questions*. Table 6.6 shows the top 15 (in absolute value) feature weights learned with the full feature group (8) on the *Deep Structure Questions*, averaged across all five cross validation folds. The top two of these top 15 features encode long-distance linguistic and semantic relationships between keyterms that are not implied by the surface representation. Ten of the top 15 make use of some semantic role information. It is interesting to note that the most basic surface pattern features, **KEnc(** `sentence` **)** and **KPrec(** `sentence` **)**, are not a part of the top 15 most useful features, all of which make use of some named entity and/or semantic role information. This fact suggests that semantic role features are indeed powerful for ranking passages with respect to *Deep Structure Questions*.

6.7 Comparison to the Structured Retrieval Method

Table 6.7 compares the rank-learning passage retrieval strategy proposed in this chapter, and the approach based on structured retrieval techniques discussed in Chapter 5. The approaches are compared against the common Indri T_{bbn} baseline method, on the 54 questions drawn from the TREC 2002 "MIT 109" test collection that were not used for training the structured retrieval approach. The learning-to-rank method can not rank sentences not retrieved by the baseline, unlike the structured retrieval approach. To compare the two approaches on ranking quality alone, the relevant sentences not retrieved from by the baseline were removed from the judgment set.

Table 6.6: Top 15 (in absolute value) mean feature weights across folds, trained on the full feature set and the *Deep Structure Questions*.

Feature Name	Mean Weight
Paths(5)	264.45
Paths(4)	211.60
Ans	183.01
Att-KEnc2(`target`, `arg1`)	174.95
AEnc(`sentence`, `date`)	99.87
Baseline Indri T_{bbn} score	98.42
KEnc(`date`)	91.33
KPrec(`arg2`)	81.71
Att-KEnc2(`target`, `arg0`)	69.97
KEnc(`org`)	65.65
KPrec(`arg0`)	55.12
Att(`target`, `argm-mnr`)	-54.89
AEnc(`sentence`, `argm-mnr`)	-54.89
ExpAtt(3)	-66.46
Att2-KEnc3(`target`, `arg1`, `arg2`)	-96.68

The table reports retrieval quality for a flavor of the structured retrieval approach referred to as *every structure*, in which the method round-robins over a set of structured queries for each question to construct the overall ranking. The Mean Average Precision score for this approach is sensitive to the order in which the per-structure queries are considered by the round-robin algorithm; *optimal* and *worst* are the best and worst orders possible, while *as published* is the order that was used in the experiments reported in [8].

The table shows that both methods statistically significantly outperform the Indri T_{bbn} baseline at the 0.01 level. Additionally, the structured retrieval can statistically significantly outperform the rank-learning approach at the 0.05 level, but only when the round-robin merging of the per-structure results is performed optimally. It is not inconceivable that a QA system using the structured retrieval approach would be able to guide the round robin process in this manner, perhaps through accurate confidence values for predicted answer-bearing structures, but for general purpose applications in which this type of information is not necessarily available, the learning-to-rank approach seems robust.

Table 6.7: Comparison of Learning-to-Rank and Structured Retrieval Methods for Linguistic and Semantic Passage Retrieval for QA

Method	MAP	% over Indri T_{bbn}	p-value ($N = 54$)	% over LTR	p-value ($N = 54$)
Structured Retrieval					
- *optimal*	0.4764	160.04%	< 0.0001	90.04%	0.0034
- *as published*	0.3837	109.44%	0.0002	53.54%	0.0670
- *worst*	0.3096	68.99%	0.0053	23.88%	0.3322
Learning-to-Rank	0.2499	36.40%	0.0486	-	-
Indri T_{bbn} baseline	0.1832	-	-	-	-

6.8 Comparison to Passage Ranking based on Dependency Path Similarity

As introduced in Section 2.2, Cui, *et al.*, proposed in 2005 a method of re-ranking passages for QA based on dependency parse similarity [14]. The method has been referred to as the state-of-the art in linguistic passage ranking for QA applications, so this section compares the proposed rank-learning approach against it. The primary finding is that the Cui method is not competitive with the ranking strategy proposed in this chapter, but instead, is much more comparable to the Indri T_{bbn} baseline. The Cui method is composed of two scores, a lexical score based on term overlap identified as "MITRE" in the text, as well as a dependency path match score.

It turns out that the dependency path match score correlates highly with the Indri T_{bbn} score; the average per-question Pearson correlation coefficient is 0.7083, with variance 0.0593, where 1.0 indicates perfect correlation, -1.0 indicates perfect inverse correlation, and 0.0 indicates a lack of correlation. Despite this correlation, it seems that the Indri score is able to capture finer distinctions in the data. For fixed values of the path match score, the Indri score, which is measured in logarithmic scale, has a variance of 7.5071. For fixed values of the Indri score, the Cui path match score has a variance of 0.0326.

While the path match score does a reasonable job of tiering the data, Indri is able to provide a better ranking within the tiers. The observation that Indri is capturing so much of the information in the path match score could be explained if the path match score is dominated by matching the named entity expected answer type, which is captured in the Indri T_{bbn} queries through the use of the `#any:`*entity* operator.

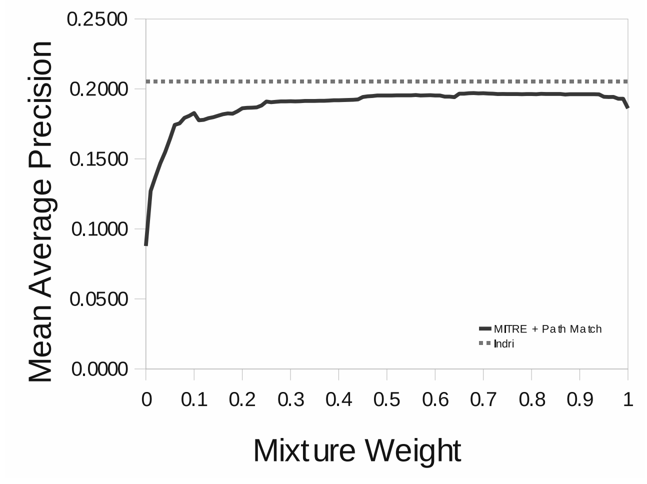

Figure 6.3: Parameter sweep for the mixture weight combining the lexical (MITRE) score with the path match score for the passage ranking method of Cui, *et al.*, based on dependency path similarity [14]. For no value of the mixture weight is the method statistically significantly better than the Indri T_{bbn} baseline.

From the paper, it is not clear what mixture weight was used to combine the lexical score and the path match score in the Cui method, but from inspection of the code available online[3], it appears that 0.5 was chosen. For the TREC 2002 "MIT 109" test collection studied in this chapter, it turned out that there was no value of the mixture parameter for which the Cui method could statistically significantly outperform the Indri T_{bbn} baseline. A plot of the parameter sweep is shown in Figure 6.3.

6.9 Guidance for QA System Developers

At this point, the reader has encountered two strategies for linguistic and semantic passage retrieval that can be incorporated into a QA system. Both strategies improve retrieval quality in terms of Mean Average Precision, retrieving more answer-bearing passages and ranking them more highly, while reducing the occurrence of false positives. This section is aimed at the developer trying to understand which method is more appropriate for his or her QA system.

[3]See: `http://www.cuihang.com/software.html`

The comparison between the methods given in Section 6.7 can be misleading if it is not carefully interpreted. As it stands, the structured retrieval method appears to be able to achieve a noticeably better MAP than the rank-learning approach, regardless of how round-robining is performed. For the worst possible round robin ordering, however, structured retrieval can not be shown to be statistically significantly better than the rank-learning method.

All of the results attributed to structured retrieval in this thesis need to be viewed from the perspective that they represent an optimistic upper bound on performance that may be difficult to replicate in real systems. For the structured retrieval approach to work well, a QA system must be able to accurately predict the linguistic and semantic features of likely answer-bearing passages. The structured retrieval results in this thesis assume that this is possible with 100% accuracy. The results for the rank-learning approach make no such assumption, using features of the question only. The recommendation, therefore, is that the learning-to-rank strategy be preferred until high-accuracy answer structure prediction becomes available.

Finally, a note of caution about linguistic and semantic passage retrieval. Though it may reduce the workload shouldered by the Answer Generation module, and improve Answer Generation accuracy[4], it can never replace Answer Generation. Consider the following sentence, *In 1867, Russia proposed that the U.S. purchase Alaska* For the question about the purchase of Alaska, the rank-learning passage retrieval strategy proposed in this chapter might rank this sentence highly because it satisfies many of the linguistic and semantic constraints expressed in the information need. There is a *purchase* event, and *Alaska* is the `arg1`. There is also an `argm-tmp` temporal adjunct containing an instance of `date`.

It is not clear whether this sentence is relevant or not. In terms of formal semantics, it does not indicate that the purchase has yet taken place. Despite containing the correct answer, the sentence does not appear to support the answer. For certain applications, however, the sentence might be relevant. A QA system capable of aggregating information from multiple passages may be able to combine the sentence with information indicating that the purchase was actually made. The sentence may be unconditionally relevant for a system tasked with summarizing historical events or compiling timelines. It is up to the individual developer to determine what relevance means in the context of his or her own QA system, and to pair linguistic and semantic passage retrieval with the appropriate Answer Generation strategy.

[4]See Section 7.7 for an analysis of Answer Generation.

6.10 Conclusions

Learning-to-rank techniques have demonstrated themselves to be highly effective as an approach to linguistic and semantic passage retrieval for QA systems. The approach outlined in this chapter offers several advantages:

1. **Partial Matching**: The rank-learning linguistic and semantic passage retrieval method supports fine grained partial matching between retrieved passages and the information need by checking constraints individually and in small groups as opposed to matching linguistic and semantic structures as a whole.

2. **Automatic Constraint Weighting**: Rank-learning techniques also free the developer from needing to specify the order of constraint relaxation *a priori*, as the relative importance and predictive power of the various component constraints is learned automatically through the training process.

3. **Robustness**: The approach was shown to improve over the Indri T_{bbn} baseline for several different feature groups.

4. **Degradation to the Baseline**: Finally, this rank-learning approach incorporates information about the baseline retrieval method that allows it to degrade gracefully to the baseline in the absence of useful linguistic and semantic information.

5. **Applicability to Other Tasks**: The learning-to-rank passage retrieval method proposed here is fundamentally a method of computing linguistic and semantic similarity between two pieces of text. The technique could easily be applied to other appropriate tasks, such as Recognizing Textual Entailment [15].

Chapter 7

Generalizing to New Problem Instances

Building on the success of the learning-to-rank approach to linguistic and semantic passage retrieval for Question Answering (QA) detailed in the previous chapter, the next task is to demonstrate the generality of the approach, and its adaptability to new domains and problem instances. This chapter presents a set of passage retrieval experiments set within the context of an Italian-language QA task similar to those presented for English in the previous chapter. These experiments represent a point of generalization along two dimensions; in addition to the corpus language, there is a new type system in use, based on the Natural Language Processing (NLP) tools available for Italian.

The discussion in this chapter is guided by the hypothesis that, for the specific test collection studied, the rank-learning-based linguistic and semantic passage retrieval approach, using the proposed type system, provides a *better quality* passage ranking compared to a standard baseline, where *better quality* entails more relevant passages retrieved, more highly ranked. Furthermore, using a standard Answer Generation baseline, this *better quality* passage ranking translates to *better quality* end-to-end system performance, in terms of accuracy and Mean Reciprocal Rank of the correct answer.

[In quale anno] [è stata lanciata] [su Hiroshima] [la bomba atomica?]
[È stata lanciata] [la bomba atomica] [su Hiroshima] [in quale anno?]
[È stata lanciata] [su Hiroshima] [la bomba atomica] [in quale anno?]
[La bomba atomica] [è stata lanciata] [su Hiroshima] [in quale anno?]
[In quale anno] [la bomba atomica] [è stata lanciata] [su Hiroshima?]

Figure 7.1: Five of the equivalent, grammatical permutations of the phrases in the question, *In quale anno è stata lanciata la bomba atomica su Hiroshima?*

7.1 Phrase Ordering in Italian

Chapter 3 argued that, despite the fact that surface term ordering often implies semantics in English, surface patterns are not necessarily the most robust approach for passage retrieval because of passive voice, topicalization and other types of constructions involving clause movement, which violate the implication. This argument is supported by the English-language result described in the previous chapter, which showed that, when more predictive features are available, the rank learning algorithm marginalizes sentence-level term ordering constraints, though ordering of terms within arguments is still useful.

The Italian language is an interesting case study for abstracting away from surface term ordering, as it has relatively weak phrasal ordering constraints, much weaker than those of the English language. Consider CLEF question 20040016, *In quale anno è stata lanciata la bomba atomica su Hiroshima?* (*In what year was the atomic bomb dropped on Hiroshima?*) Figure 7.1 shows just a few of the permutations of the arguments of this question that are considered grammatical according to a native speaker of the language. For a language with relatively free phrase ordering constraints, one would expect surface patterns to be an even weaker predictor of relevance than for English and, as a consequence, the deeper linguistic and semantic relations between keyterms would become even more important.

7.2 Type Systems

As in the English-language passage retrieval experiments, those described in this chapter rely on two different type systems. The simpler type system, T_{chaos_ne}, which has fewer element and relation types, is used to formulate Indri queries for the baseline method. The other type system, T_{chaos}, supports richer linguistic relations, which can be used as features in a trained model that can re-rank the baseline results to obtain a *better quality* ranking. Section A.8 defines T_{chaos_ne}, the baseline type system for this task, while Section A.9 gives the type system used for re-ranking, T_{chaos}, which unlike the corresponding type system in the English-language experiments, does not model semantics. This limitation is a result of the fact that the Chaos parser, described in Section 4.3.2, does not perform semantic role labeling, and there are no semantic role labeling tools known to be available for Italian.

$$G_{20040016} = (E,\, R,\, T_{chaos})$$

$$E = \left\{ \begin{array}{ll} e_1 = (\texttt{sentence}), & e_8 = (\text{anno}: \textit{year}), \\ e_2 = (\texttt{verb}), & e_9 = (\text{lanciare}: \textit{to drop}), \\ e_3 = (\texttt{V\_PP}), & e_{10} = (\text{bomba}: \textit{bomb}), \\ e_4 = (\texttt{V\_Sog}), & e_{11} = (\text{atomico}: \textit{atomic}), \\ e_5 = (\texttt{V\_PP}), & e_{12} = (\text{Hiroshima}) \\ e_6 = (\texttt{location}), & \end{array} \right\}$$

$$R = \left\{ \begin{array}{ll} (enclosure,\, e_1,\, e_2), & (enclosure,\, e_2,\, e_9), \\ (enclosure,\, e_1,\, e_3), & (enclosure,\, e_3,\, e_8), \\ (enclosure,\, e_1,\, e_4), & (enclosure,\, e_4,\, e_{10}), \\ (enclosure,\, e_1,\, e_5), & (enclosure,\, e_4,\, e_{11}), \\ (enclosure,\, e_1,\, e_6), & (enclosure,\, e_5,\, e_6), \\ (enclosure,\, e_1,\, e_7), & (enclosure,\, e_5,\, e_{12}), \\ (enclosure,\, e_1,\, e_8), & (enclosure,\, e_6,\, e_{12}), \\ (enclosure,\, e_1,\, e_9), & (attachment,\, e_2,\, e_3), \\ (enclosure,\, e_1,\, e_{10}), & (attachment,\, e_2,\, e_4), \\ (enclosure,\, e_1,\, e_{11}), & (attachment,\, e_2,\, e_5) \\ (enclosure,\, e_1,\, e_{12}), & \end{array} \right\}$$

Figure 7.2: Information need corresponding to the input question *In quale anno è stata lanciata la bomba atomica su Hiroshima?* (*What year was the atomic bomb dropped on Hiroshima?*) expressed under T_{chaos}.

Figure 7.2 shows a depiction of the information need annotation graph corresponding to our example question, *In quale anno è stata lanciata la bomba atomica su Hiroshima?*, under T_{chaos}. Note that the keyterms in the question have been lemmatized. The figure reveals an important distinction between T_{chaos} and $T_{bbn+assert}$, which was used in the previous chapter's experiments. Chaos is a syntactic parser, so it identifies the phrases for grammatical functions such as subject (`V_Sog`) and oblique or prepositional (`V_PP`). These are syntactic distinctions and are not to be confused with the deeper semantic roles produced by ASSERT. Because the question is in passive voice, the thing that is dropped becomes the grammatical subject, as opposed to the agent of the dropping action, which is unspecified.

Use of a syntactic representation rather than a semantic one can also result in a wider variation among answer-bearing passages. Consider the following passages judged relevant for question 20040016, shown below, and again in Figure 7.3. The figure shows the three answer-bearing passages, each having different syntactic forms, mapping into a unified semantic representation modeled after what an ASSERT-like tool would produce if one were available for the Italian language.

1. ... [`V_Sog` *il bombardiere B-29 'Enola Gay'*] *che* [`V_Obj` *il 6 agosto 1945*] [`verb` *sganciò*] [`V_PP` *su Hiroshima*] [`V_Obj` *la prima bomba atomica della storia*] ...

   ... the B-29 bomber 'Enola Gay,' which on August 6, 1945 released over Hiroshima the first atomic bomb in history ...

2. ... [`V_Sog` *bomba atomica*] [`verb` *lanciata*] [`V_PP` *su Hiroshima*] [`V_PP` *dagli americani*] [`V_PP` *nel 1945.*]

   ... atomic bomb dropped on Hiroshima by the Americans in 1945.

3. ... [`V_Sog` *la decisione di 50 anni fa del presidente Harry Truman*] *di* [`verb` *lanciare*] [`V_Obj` *la bomba atomica*] [`V_PP` *sulle città giapponesi di Hiroshima e Nagasaki*]

   ... the decision 50 years ago of President Harry Truman to drop the atomic bomb on the Japanese cities of Hiroshima and Nagasaki.

7.3 Linguistic Constraints for T_{chaos}

Many of the methods for decomposing an annotation graph representing an English-language question under $T_{bbn+assert}$ into atomic constraints apply equally well to the task of Italian-language passage retrieval for QA. All of the constraints defined in Section 6.3 for the English-language experiments and repeated below for the reader's convenience are re-used in the experiments in this chapter. Note that, when used below, element type *annotation* refers to the T_{chaos} type system instead of the $T_{bbn+assert}$ used in the previous chapter.

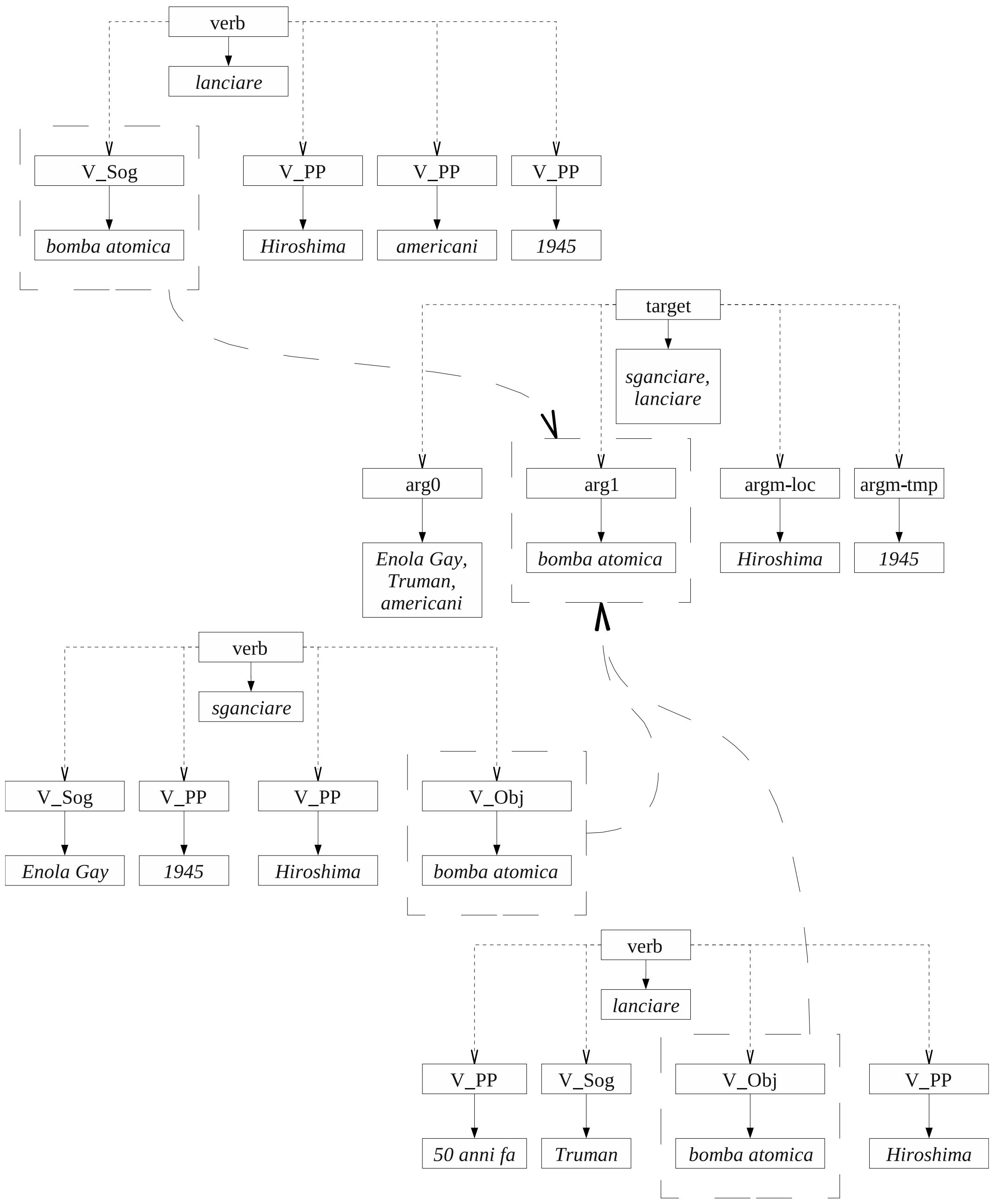

Figure 7.3: Syntactic representations for three answer-bearing sentences for the question, *In quale anno è stata lanciata la bomba atomica su Hiroshima?*, shown mapping into an idealized semantic representation similar to that of ASSERT.

- **KEnc(*annotation*)**: Keyterm Enclosure within an Annotation

- **KPrec(*annotation*)**: Keyterm Precedence within an Annotation

- **AEnc(*annotation*, *annotation*)**: Annotation-Annotation Enclosure

- **Att(*annotation*, *annotation*)**: Attachment between Annotations

  Under T_{chaos}, the *attachment* is defined to hold over domain type `verb` and range type *function*, which has 6 subtypes: `V_Sog`, subject; `V_Obj`, direct object; `V_NP`, indirect object; `V_PP` prepositional or oblique; `V_Adv`, adverbial; and `V_PrRel`, relative clause.

- **Ans**: Expected Answer Type

  Under T_{chaos}, the *entity* types that can serve as the expected answer type are: `date`, `location`, `organisation`, and `person`.

- **ExpAtt(*annotation*, *N*)**: Coverage of Expected Attached Annotations

  For Italian, $N \in \{1, \ldots, 6\}$.

- **Att-KEnc2(*annotation*, *annotation*)**: Attachment, with double Keyterm Enclosure

- **Att2-KEnc3(*annotation*, *annotation*, *annotation*)**: Double Attachment, with triple Keyterm Enclosure

- **Paths(*N*)**: Coverage of Expected Keyterm Paths

  For Italian, $N \in \{0, \ldots, 10\}$.

Additionally, there is a new feature type introduced for Italian: **Att2-KEnc2(*annotation*, *annotation*, *annotation*)**. It is similar to the **Att2-KEnc3(*annotation*, *annotation*, *annotation*)**, with the exception that it does not enforce an *enclosure* relation between the *annotation* that is the domain of the *attachment* relations and a *keyterm*. Operationally, this constitutes checking that two *keyterm* instances are related through a `verb` that is not specified. Table 7.1 shows the full set of features used in the experiments in this chapter.

Table 7.1: Learning-to-Rank Features for Italian, with group membership. Features are parameterized by leaf element types defined in Section A.9. N identifies the length of the path through the annotation graph

Feature Name	Feature Count	Feature Groups
Baseline Indri Score	1	1-8
KEnc(`sentence` **)**	1	1,2,6-8
KPrec(`sentence` **)**	1	2,4,7,8
KEnc(***entity*** **)**	4	3,4,8
AEnc(`sentence`, ***entity*** **)**	4	3,4,8
Ans	1	3,4,8
KPrec(***entity*** **)**	4	4,8
Att(***function*** **)**	6	5-8
AEnc(`sentence`, `verb` **)**	1	5-8
AEnc(`sentence`, ***function*** **)**	6	5-8
ExpAtt(`verb`, ***N*** **)**	6	5-8
KEnc(`verb` **)**	1	6-8
KEnc(***function*** **)**	6	6-8
Att-KEnc2(`verb`, ***function*** **)**	6	6-8
Att2-KEnc3(`verb`, ***function***, ***function*** **)**	15	6-8
Att2-KEnc2(`verb`, ***function***, ***function*** **)**	15	6-8
KPrec(`verb` **)**	1	7,8
KPrec(***function*** **)**	6	7,8
AEnc(***function***, ***entity*** **)**	24	8
Paths(***N*** **)**	11	8
Total	119	8

7.4 Experimental Methodology

This chapter's experiments will evaluate the learning-to-rank approach using the Italian-language CLEF 2004/2006 test collection, defined in Section 4.1.2. As in the English-language experiments, the rank-learning approach is compared against baseline Indri queries, in terms of Mean Average Precision, using a 5-fold cross validation approach. The baseline queries, formulated under T_{chaos_ne} according to the procedure described in Section 5.4.1, are used to retrieve 1000 results, which are then re-ranked using the trained model.

In contrast to the experiments in the previous chapter, in which fewer than half of the questions in the test collection had an interesting interpretation under the linguistic and semantic type system in use ($T_{bbn+assert}$), nearly all of the questions in the CLEF 2004/2006 test collection have information needs with constraints representable by T_{chaos}. These constraints become 119 count-based features[1], shown in Table 7.1, that, along with the baseline Indri score, are used by the trained Committee Perceptron model [16] to re-rank the baseline query results. See Section 4.4.2 for more information about the Committee Perceptron algorithm.

As in the previous chapter, improvement in passage retrieval quality is measured separately for eight different feature groups, which are described below. They correspond to the groups used in the English-language experiments, except for the fact that semantic roles have been replaced by syntactic chunking and grammatical functions. The right-hand column of Table 7.1 lists which of the following groups contain which features.

1. **Keyterms Only** uses bag-of-words constraints only.

2. **Surface Patterns** adds pairwise *precedence* between *keyterms* in the `sentence` to the above.

3. **Named Entities + Keyterms** consists of *enclosure* relations between the `sentence` and both *keyterms* and *entities*.

4. **Named Entities + Surface Patterns** all of the above, plus *precedence* relations between *keyterm* pairs within *entities*.

[1]All questions share these 119 features, but the range of values they may take is defined on a per-question basis. See Section 6.5 for more information.

5. **Grammatical Functions Only** consists of *enclosure* relations between the `sentence` and `verbs` and *functions*, as well as *attachment* relations between `verbs` and `functions`. No information about *keyterms* is used.

6. **Grammatical Functions + Keyterms** contains *enclosure* relations between `sentence`, `verb` and *function annotations* and *keyterms* in addition to the features in the **Grammatical Functions Only** group.

7. **Grammatical Functions + Surface Patterns** uses all features in the previous group, with the addition of *precedence* relations between *keyterm* pairs enclosed in the `sentence`, `verbs` and *functions.*

8. **Grammatical Functions + Named Entities + Surface Patterns** consists of the union of all above-described feature groups, plus *enclosure* of *entities* in *functions* and the **Paths(*N*)** feature.

7.5 Experimental Results and Discussion

Five-fold cross validation was performed on the 400 questions in the CLEF 2004/2006 test collection, resulting in 320/80 training/test questions in each fold. Table 7.2 summarizes the results in terms of Mean Average Precision (MAP) for each feature group, with *p*-values given for Fisher's randomization test [55].

Table 7.2: Passage Retrieval Quality using Rank-Learning Techniques on the CLEF 2004/2006 test collection

	MAP	% over Indri T_{chaos_ne}	p-value ($N = 400$)
Indri T_{chaos_ne}	0.2140	–	–
Feature Group 1	0.2314	8.13	0.0005
2	0.2389	11.63	< 0.0001
3	0.2519	17.71	< 0.0001
4	**0.2568**	20.00	< 0.0001
5	0.2169	0.93	0.0989
6	0.2014	-5.88	0.7922
7	0.2177	1.72	0.3662
8	0.2405	12.38	0.0231

Table 7.2 shows that the rank-learning approach can provide an improvement in passage retrieval quality over the baseline, with feature groups 1 through 4 and 8 showing statistical significance at the 0.05 level[2]. Feature groups 1 and 2 show a percentage improvement over the baseline similar to what was observed in the English-language experiments, and groups 3 and 4 show better improvements than the corresponding English feature groups. These feature groups are directly comparable to the English-language experiment, because they contain almost identical features, regardless of the language.

Groups 5 through 8, however, are not comparable to the English experiment because they contain no semantic information, relying instead on syntactic chunking with grammatical functions produced by the Chaos parser. It is not surprising that the syntactic analysis is less useful for ranking than the semantic analysis provided by ASSERT was for English data. One reason, illustrated in Figure 7.3, is that there is simply more variation between questions and answer-bearing passages under T_{chaos}. If an ASSERT-like representation were to be available for Italian, it could abstract over much more of the question-answer variation and would likely perform more like ASSERT does for English.

Another reason that the Chaos output is of reduced effectiveness as a passage ranking feature is that there are often many, many more analyses on a piece of text than ASSERT produces for the English-language data. There are three contributing factors for this. First, Chaos provides analyses for verbs such as *be* and *have*, which ASSERT explicitly does not cover. Second, at indexing time, as described above, when Chaos produces analyses with packed ambiguities, an index representation is produced for each hypothesis. The third cause is the characteristics of the language. In Italian, some adjectives are difficult to distinguish from the past participles of verbs. Often, the only difference is the presence of an auxiliary. Consider the difference between *la donna spaventata* (the scared woman) and the *la donna si è spaventata* (the woman got scared). Often, Chaos will produce a spurious `verb-V_Sog` relation for the former.

One interesting result revealed by Table 7.2 is that feature group 5, which considers `verb`-*function* structure in the absence of lexical information, outperforms feature group 6, which adds in that information about *keyterms*. This is the opposite conclusion drawn from the English-language result, which indicates that enforcing specific linguistic and semantic relationships between keywords is quite effective for passage ranking. It seems that the Chaos analysis, which as described above overgenerates in

[2]Significance for group 8 is reduced when fewer stopwords are filtered. See Section 4.1.2 for more information about the stopwords filtering used in these experiments.

some respects, serves two purposes. Without *keyterms*, it actually supports a greater degree of matching and manages to outperform the baseline slightly. It is, however, interesting to note that the p-value is rather low for a less than 1% increase in MAP. When *keyterm* information is introduced in feature group 6, performance suffers by 5%, likely a result of the increased number of false positive linguistic relationships *keyterms* are participating in. Despite this, the model is able to statistically significantly (at the 0.05 level) outperform the baseline for feature group 8, which shows that the model is robust enough to emphasize the features most useful for ranking even in the face of noisy data.

Table 7.3 shows the top 15 greatest magnitude feature weights, drawn from feature group 8, averaged across the 5 folds. As in the English language experiments, the **Paths(*N*)** feature turns out to be effective for passage ranking. It can also be seen that the model is balancing bag-of-words, named entity, and `verb`-*function* constraints along with *keyterm precedence* constraints, which would not be expected to perform well on Italian owing to its relatively freer phrasal ordering rules. This can be explained by the fact that many of the questions have *keyterm precedence* relations within phrases that are likely to be preserved, as in, for example, question 20040094, *Qual è un fattore di rischio per le malattie cardiovascolari?* (What is a risk factor for cardiovascular disease?). The model puts weight on **KPrec(** `sentence` **)** not because ordering is preserved between phrases, but within phrases, such as *fattore di rischio* and *malattie cardiovascolari*. This can be seen by the fact that **KPrec(*annotation*)** is weighted highly for the `V_Sog`, `person` and `V_Adv` *annotation* types.

As described in Section 4.1.2, the questions in the CLEF 2004/2006 test collection are different from those in the TREC 2002 "MIT 109" test collection, in that they contain some definition questions and list questions in addition to factoid questions. Table 7.4 shows results reported separately for each question type. Results for the factoid questions closely mirror the results for the population as a whole, with the exception that feature group 8 no longer shows statistically significant improvements, possibly owing to the loss of the definition questions, for which the features in group 8 appear to be quite effective, showing an almost 15% improvement with statistical significance at the 0.01 level.

The definition questions for which the group 8 feature set is particularly well-suited fall into a category of *who is* questions that ask about specific people by first and last name. The information needs corresponding to these questions label the person of interest as the `V_Sog` of some `verb`, as well as an instance of the `person` named entity type. This leads to a number of interesting paths between the *keyterms* that make up the person's name, which are reflected in the **Paths(N)** family of

Table 7.3: Top 15 (in absolute value) mean feature weights across folds for the CLEF 2004/2006 test collection and feature group 8.

Feature Name	Mean Weight
KPrec(`V_Sog` **)**	536.06
KEnc(`V_Sog` **)**	465.09
Att2-KEnc2(`verb`, `V_NP`, `V_Sog` **)**	407.83
KEnc(`V_Obj` **)**	380.60
Att-KEnc2(`verb`, `V_PP` **)**	374.91
Paths(6 **)**	305.33
Att-KEnc2(`verb`, `V_Obj` **)**	305.26
KEnc(`person` **)**	292.26
KEnc(`V_PP` **)**	289.83
KPrec(`sentence` **)**	280.41
KPrec(`person` **)**	249.04
Paths(0 **)**	234.17
Paths(4 **)**	215.56
KPrec(`V_Adv` **)**	205.37
KEnc(`verb` **)**	-219.51

features in group 8. Additionally, the surface pattern features, present in groups 7 and 8, are able to make some use of the fact that the person of interest is given by both first and last name. The *precedence* relations between these *keyterms* tend to be preserved in the answer-bearing passages.

It bears mention that CLEF definition questions are different from the TREC notion of a definition question, with which the reader might be more familiar. Because CLEF takes a single-answer approach to definition question evaluation instead of adopting a nugget-based approach, the answers to *who* or *what is* types of questions tend to be phrased as parentheticals or appositives, which are not as well-represented by the type system as copulae are. This fact likely accounts for the reason why the baseline bag-of-words plus named entity queries perform better on the definition questions than the factoid questions. In contrast, CLEF list questions are not evaluated using a single-answer methodology, but there were not enough of them in the test collection to be able to learn much from them.

Table 7.4: Passage Retrieval Quality using Rank-Learning Techniques on the CLEF 2004/2006 test collection, by question type. Count of questions with at least one relevant item given in parentheses.

Type (N)	Feature Group	MAP	% over Indri T_{chaos_ne}	p-value
Factoid	Indri T_{chaos_ne}	0.1897	–	–
($N = 263$)	1	0.2100	10.70	0.0009
	2	0.2187	15.28	< 0.0001
	3	0.2342	23.45	< 0.0001
	4	**0.2397**	26.35	< 0.0001
	5	0.1919	1.15	0.2164
	6	0.1754	-7.53	0.8109
	7	0.1902	0.26	0.4696
	8	0.2121	11.80	0.0776
Definition	Indri T_{chaos_ne}	0.3306	–	–
($N = 53$)	1	0.3325	0.57	0.0355
	2	0.3325	0.57	0.0629
	3	0.3343	1.11	0.0168
	4	0.3347	1.24	0.0145
	5	0.3372	1.99	0.0107
	6	0.3397	2.75	0.3319
	7	0.3658	10.64	0.0537
	8	**0.3800**	14.94	0.0039
List	Indri T_{chaos_ne}	0.4725	–	–
($N = 4$)	1	0.4707	-0.38	0.7483
	2	0.4861	2.87	0.4885
	3	0.4717	-0.16	0.7442
	4	0.4858	2.81	0.5040
	5	0.4799	1.56	0.1248
	6	0.3217	-31.91	0.6854
	7	0.3228	-31.68	0.6843
	8	**0.5039**	6.64	0.2494

7.6 Varying the Size of the Retrieval Unit

The judgment set for the Italian-language CLEF 2004/2006 QA test collection, described in Section 4.1.2, contains not only sentence-level judgments, but also block-level judgments, where block is construed to mean a contiguous three-sentence window. Any block that contains a relevant sentence is relevant, but there are also some blocks that are relevant even though no single one of the individual sentences comprising it is answer-bearing. For these blocks, the answer is spread across two or three sentences, usually as a result of anaphora. Many of these blocks contain sentences that are "near misses," in that they satisfy most of the linguistic and semantic constraints laid out in the information need, yet are not answer-bearing owing in many cases to unresolved anaphora. With the three-sentence block as the unit of retrieval, many of these cases become relevant because, within a two-sentence horizon, the anaphora are resolved.

Table 7.5: Block-level Passage Retrieval Quality using Rank-Learning Techniques on the CLEF 2004/2006 test collection

	MAP	% over Indri T_{chaos_ne}	p-value ($N = 400$)
Indri T_{chaos_ne}	0.1913	–	–
Feature Group 1	0.2068	8.10	< 0.0001
2	0.2102	9.87	0.0001
3	0.2313	20.90	< 0.0001
4	**0.2392**	25.03	< 0.0001
5	0.1298	-32.14	1.0000
6	0.1415	-26.03	1.0000
7	0.1633	-13.06	0.9871
8	0.1903	-0.52	0.5095

Table 7.5 shows the results when the experiment is repeated using the block as a unit of retrieval. It can immediately be seen that the Mean Average Precision scores are depressed, in part due to the fact that there are more relevant items to retrieve. Moving to the block unit of retrieval also renders the `verb`-*function* features much less useful for ranking; the training process is confused by linguistic relations matching within blocks, yet not contributing to relevance. This issue is particularly pronounced when training on blocks that contain a single relevant sentence, and two non-relevant sentences, which comprise the majority of blocks. Inspecting the

learned feature weights in this experiment confirm that a majority of the top fifteen feature weights are either **KEnc(** ***annotation*** **)** or **KPrec(** ***annotation*** **)**.

7.7 Downstream Answer Generation Performance

This analysis attempts to answer the question as to whether or not an unchanged downstream Answer Generation module, and the end-to-end QA system as a whole, can benefit from a *better quality* passage ranking coming from the embedded IR component of a QA system. To show that such a benefit is possible, it suffices to demonstrate a single example, though the same methodology could easily be applied to the English-language data. This section studies an accepted Answer Generation baseline, which extracts *entities* of the expected answer type from the retrieved passages, and ranks them according to the passage ranking. This approach debuted in early TREC QA systems [51, 1], and is still in use today [53].

End-to-end QA system performance is measured in terms of accuracy, or the proportion of questions for which the top-ranked answer is correct, as well as Mean Reciprocal Rank (MRR) in which the first correct answer encountered at ranks 1 through 5 contributes to the overall score by a factor of one over the rank. Very good or very poor Answer Generation quality can hide retrieval effectiveness. Perfect Answer Generation will find the correct answer regardless of the rank at which the answer-bearing passage is retrieved. The worst possible Answer Generation module will never find the correct answer regardless of the passage retrieval output. Realistic Answer Generation modules fall somewhere between these two extremes.

Table 7.6 summarizes accuracy and MRR for 223 of the questions in the CLEF 2004/2006 test collection, those questions for which the expected answer type is either `date`, `location`, `organisation` or `person`, as the passage ranking is improved from the baseline T_{chaos_ne} ranking to the rankings according to feature groups 4 (best quality) and 8 (all features). For each test, the p-value according to Fisher's randomization test [55] is given, along with N, the count of questions with having a given answer type.

Over all answer types, the passage ranking according to feature group 4 provides a statistically significant improvement in end-to-end QA system effectiveness, in terms of both accuracy (0.05 level) and MRR (0.01 level), when compared with the baseline passage ranking. For feature group 8, the improvements are not statistically significant, despite the fact that, in terms of accuracy, both feature groups provide an identical improvement in terms of the raw statistic. For feature group 4, there

Table 7.6: End-to-end QA system performance in terms of accuracy and MRR by expected answer type, for feature groups 4 (above) and 8 (below).

Group 4		Accuracy			MRR		
		Indri	Feature		Indri	Feature	
Answer Type	N	T_{chaos_ne}	Group 4	p-value	T_{chaos_ne}	Group 4	p-value
`date`	46	0.1739	0.1522	1.0000	0.2391	0.2609	1.0000
`location`	61	0.1803	0.3279	0.0122	0.4098	0.5738	0.0008
`organisation`	56	0.0179	0.0179	1.0000	0.0179	0.0179	1.0000
`person`	60	0.1667	0.2167	0.4581	0.3000	0.3333	0.2566
all	223	0.1345	0.1839	0.0189	0.1794	0.2330	0.0012

Group 8		Accuracy			MRR		
		Indri	Feature		Indri	Feature	
Answer Type	N	T_{chaos_ne}	Group 8	p-value	T_{chaos_ne}	Group 8	p-value
`date`	46	0.1739	0.1522	1.0000	0.2391	0.1957	1.0000
`location`	61	0.1803	0.3279	0.0344	0.4098	0.5082	0.0276
`organisation`	56	0.0179	0.0357	1.0000	0.0179	0.0357	1.0000
`person`	60	0.1667	0.2000	0.7735	0.3000	0.3500	0.3921
all	223	0.1345	0.1839	0.0783	0.1794	0.2225	0.0591

are more questions for which the accuracy is equal to the baseline, but the number of occurrences in which the baseline is better is reduced, compared to feature group 8.

For specific answer types, statistically significant improvement at the 0.05 level can be seen for `location` only, for both feature groups. For `date`, however, end-to-end accuracy for both feature groups, as well as MRR for feature group 8, decline. Though `date` is the worst-performing answer type in terms of Mean Average Precision of the original T_{chaos_ne} ranking, the trained model improves the ranking of answer-bearing passages. Despite this, there is a drop in Answer Generation performance, which can be explained by the fact that Chaos is not tagging *keyterms* representing years, such as *1945*, as `date`. It only writes a `date` tag when a date expression consisting of a month and a year, or a day, a month and a year, was recognized and canonicalized in free text, such as *6 agosto 1945* becoming *6/08/1945*. For all `date` questions in which performance declined, the explanation was that relevant passages containing the correct answer expressed as an untagged year were brought to the top of the ranking by the trained model, pushing other results down.

For `organisation`, accuracy and MRR are equivalent because the correct answer was either extracted at rank 1 or not at all. In fact, only one of the 56 questions was correctly answered by the baseline, and one additional question was correctly answered by the feature group 8 ranking. The feature group 4 ranking results in identical accuracy and MRR compared with the baseline. Performance is particularly poor for `organisation` because Chaos appears to be using a dictionary-based approach for this *entity* type. *NATO* is the answer most frequently proposed for an `organisation` question, followed by the single keyterms *club*, *corporation*, *bank* and *hotel*. These English words would be found in Italian-language newswire text as part of proper nouns.

7.8 Conclusions

This chapter demonstrated that the rank-learning approach to linguistic and semantic passage retrieval for QA is successful on an additional problem instance, with a different corpus language and a different type system. It can be concluded that the approach adapts to new domains having different language-specific and type system-specific characteristics.

This problem instance was considerably more challenging because of the level of abstraction of the type system; using a syntactic, as opposed to a semantic, type system reduces the chance that the information need will match answer-bearing text to a high degree. It was observed that the approach is sensitive to the accuracy of the text annotation under the type system in use. Accuracy of the Chaos parser, as well as the choice of the unit of retrieval, affected the quality of the overall ranking. There is clearly a minimum threshold for data quality, and more work will be needed to more accurately bound it.

Despite this, the approach demonstrated robustness in that it was able to emphasize the most useful features and provide a *better quality* ranking with respect to the baseline in many cases. This behavior of the rank-learning approach to passage retrieval will be useful in any application in which it is not known *a priori* which of the features will be most reliable or useful for ranking.

An additional result shown in this chapter is that it is possible for a QA system to be more effective in terms of end-to-end accuracy and answer MRR when the quality of the passage ranking coming out of the embedded IR component is improved, and the Answer Generation component is left unchanged. Intuition suggests that if the Answer Generation module is re-tuned after the introduction of linguistic and semantic passage retrieval in order to better take advantage of the new kinds of retried results, the end-to-end system improvement could be even greater.

Chapter 8

Contributions

This thesis has studied the problem of passage retrieval within the context of Question Answering (QA) systems. Guided by the observation that standard approaches to *ad hoc* retrieval are not necessarily well-suited to the task of passage retrieval for QA, this document set out to develop passage retrieval methods capable of query-time checking of the linguistic and semantic constraints of interest to a QA system. This chapter summarizes the contributions of this thesis.

8.1 Modeling Annotated Text as Annotation Graphs

Chapter 3 offered a methodology for converting any text annotation scheme based on the concept of typed text spans and typed pairwise relations between them into a type system, which can then be used to represent QA system information needs, as well as annotated text, as annotation graphs. This conversion into a common, unifying representation enables direct comparison retrieved text and the information need in terms of linguistic and semantic content.

Subsequent chapters of the thesis proposed specific linguistic and semantic passage retrieval approaches in terms of the annotation graph formalism. For a QA system to be able to make use of them, the only assumption is that its internal representation for its information need can be reduced to an annotation graph. This is a relatively weak assumption, so the applicability of the proposed passage retrieval methods is broad.

8.2 Structured Retrieval

In Chapter 5, annotation graph retrieval was implemented using structured retrieval techniques. The method was shown to be effective, particularly for complex information needs encoding large numbers of linguistic and semantic constraints, yet sensitive to the accuracy of the tools involved in corpus preparation. It was demonstrated that the structured queries do a good job of selecting passages that satisfy a large number of the linguistic and semantic constraints in the information need, but when only a few of them are not satisfied, ranking quality degrades sharply, worse than the baseline.

It is therefore important for any QA system employing the structured retrieval strategy as proposed to be able to confidently predict the form of answer-bearing passages likely to be found in the corpus. When accurate prediction is available, a comparison between the structured retrieval and learning-to-rank methods shows that the structured retrieval method is more effective in terms of Mean Average Precision. The same comparison shows that, if the system's predictions turn out to be poor, the structured retrieval method is not statistically significantly more effective than the rank-learning approach.

8.3 Learning-to-Rank

The rank-learning approach to linguistic and semantic passage retrieval for QA, explored in Chapters 6 and 7, was shown to be effective for two different languages and two different type systems, demonstrating that it is adaptable to a variety of domains. Unlike individual structured queries, the trained ranking model is able to gracefully decay to baseline ranking performance when linguistic and semantic constraints are not satisfied.

This method was also shown to be somewhat sensitive to the quality of the tools used to prepare the corpus, but the primary advantage of the rank-learning approach is its robustness. Through the training process, the learning-to-rank method can marginalize unreliable or noisy features, and emphasize those that are the best predictors of relevance. When the QA system does not have complete confidence in its ability to predict the form answer-bearing passages in the corpus will take, the rank-learning approach, which relies on features of the question alone, offers the best balance of effectiveness and robustness.

The *better quality* passage ranking provided by the learning-to-rank method was shown to be able to translate into improved end-to-end QA system performance in terms of answer accuracy for certain types of questions. This result can change owing to variations in the effectiveness of Answer Generation algorithms, and in the degree to which they incorporate features of the passage ranking.

With its novel decomposition of annotation graphs into atomic linguistic and semantic constraints, the learning-to-rank approach to linguistic and semantic passage retrieval for QA systems proposed in this thesis represents a step towards the rank-learning community's broad vision of applicability to "different types of information needs, more linguistic features, ... [and] different applications" [13].

8.4 Future Research Directions

The structured retrieval results in this thesis assume the existence of an accurate method of predicting the form of likely answer-bearing text given the question to estimate an upper bound for effectiveness of the retrieval method. Results indicate that the method can achieve high Average Precision when an accurate prediction capability exists, therefore, building such a predictor constitutes a natural avenue for future research. A possible approach might be to learn a set of structural transforms between question and predicted answer structure, but to accurately apply the transforms, the model would need to be lexicalized, which would require vast amounts of training data.

There is good reason to be optimistic about the learning-to-rank approach to linguistic and semantic passage retrieval for QA proposed in this thesis. Unlike the structured retrieval, this method is not burdened by the need to predict the structure of answer-bearing passages. Features of the question alone were shown to be sufficient to provide an improved passage retrieval baseline. The results in this thesis were obtained using the Committee Perceptron learning algorithm [16] with little tuning. The next step for the rank-learning method is to apply techniques from the machine learning toolbox to optimize performance, including feature selection, tuning the model's parameters, trying different learners, or even ensemble learning techniques such as bagging and boosting.

Another interesting direction for future work would be to combine the two approaches. One of the issues with the structured retrieval approach is that it is not clear which of the constraints is most important, so the queries contain all of them. These queries can result in high-precision, low-recall passage retrieval. As a result of the training process of the rank-learning approach, the various constraints are individually weighted according to the degree to which they contribute to a quality ranking. Without any guidance as to which constraint components to use, earlier attempts to construct partial structured queries had failed. With these weights, it may be possible to construct a structured query with the minimum constraints required to achieve a ranking with a good balance of precision and recall.

8.5 Conclusions

This thesis proposed two linguistic and semantic passage retrieval methods that address the problem of poor quality passage retrieval in Question Answering systems, which is recognized among the QA research community as one of the sources for poor answer quality. Both strategies rely on the principle of integrating lingusitic and semantic features obtained through Question Analysis into the retrieval process. The proposed methods were shown to noticeably reduce false positives, which can, in turn, improve end-to-end system accuracy and system latency by reducing the burden on post-retrieval Answer Generation processes.

In the future, the increasing quality of NLP tools will allow QA systems to more reliably extract deeper linguistic and semantic features from questions and answer-bearing passages. As the system's understanding more closely approximates that of a human, the proposed linguistic and semantic methods will only become more effective. Substantial opportunities for future research exist in the adaptation of new features and linguistic and semantic representations for use with the proposed retrieval strategies. In time, the goal of fast and accurate passage retrieval will be achieved, bringing the field one step closer to realizing the vision of real-time Question Answering.

Appendix A

Type System Reference

This appendix collects all of the specific type system definitions referred to throughout this thesis in one place for convenient reference. Recall from Chapter 3 that a type system consists of a tuple that defines a set of element type definitions Te and a set of relation type definition Tr. Each element type definition consists of a name, and an optional pointer to a parent type within the same type system. Each relation type definition consists of a name, and domain and range element types drawn from the same type system. Every instantiated annotation graph is associated with a specific type system.

Type systems are covered in the order in which they are introduced in the text, and relationships between type systems are highlighted where appropriate. The type system definitions in this appendix adhere to the following conventions:

- In every type system, there is an element type corresponding to each keyterm k in the vocabulary V. This is written as a set of element types $k \in V$, all of which inherit from the element type *keyterm*.
- Element types in *italics*, including *keyterm* itself, can be considered to be abstract in that they exist solely to make the definition of element types more concise.
- Concrete, leaf element type other than *keyterms* are given in `monospace type`.
- Common set operator notation is used to explain the relationships between type systems. For example, the proper superset operator can be used to indicate that one type system extends another: $T_1 \supset T_2$ means that $Te_1 \supset Te_2$ and

$Tr_1 \supset Tr_2$. The union operator can be used to indicate a form of multiple inheritance in which one type system is the combination of two others: $T_1 = T_2 \cup T_3$ means that $Te_1 = Te_2 \cup Te_3$ and $Tr_1 = Tr_2 \cup Tr_3$.

Figure A.1 gives a graphical overview of the relationships between the type systems defined in this appendix. Arrows between type systems can be understood as a superset relation; the type system at the head of the arrow is a superset of the type system at the tail.

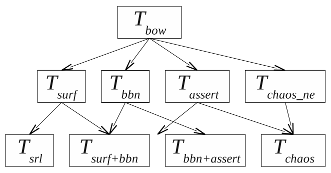

Figure A.1: Depiction of superset relationships between the type systems used in this thesis.

Table A.1 lists the type systems defined in this appendix and gives the section of the text where they are first referenced.

Table A.1: Type systems used in this thesis, and the section of the text where they are first referenced.

Type System	Section
T_{bow}	3.2
T_{surf}	3.2
T_{srl}	3.3
T_{bbn}	5.4.1
$T_{bbn+assert}$	5.2
$T_{surf+bbn}$	5.6
T_{assert}	5.7
T_{chaos_ne}	7.2
T_{chaos}	7.2

A.1 Bag-of-Words Type System T_{bow}

The simplest type system used in this thesis, T_{bow} supports the notion of *keyterm enclosure* within a `sentence`. It is introduced in Section 3.2 to serve as an example of how the annotation graph can serve as a representation for text.

$$T_{bow} = (Te, Tr)$$

$$Te = \left\{ \begin{array}{l} (annotation, \emptyset), \\ (\texttt{sentence}, annotation), \\ (keyterm, \emptyset), \\ (k \in V, keyterm) \end{array} \right\} ; Tr = \left\{ \ (enclosure, annotation, keyterm) \ \right\}$$

Figure A.2: Definition of a simple bag-of-words type system, supporting *keyterm enclosure* within a `sentence`.

A.2 Surface Pattern Type System T_{surf}

Surface patterns is a richer level of representation than is bag-of-words, because it is able to distinguish between passages containing the same *keyterms* in a different order. In English, *keyterm* ordering is quite predictive of meaning; as an example, consider the difference between the sentences *Alice hit Bob* and *Bob hit Alice.* This type system is the same as T_{bow}, except for the addition of the *precedence* relation, which is defined to hold over pairs of *keyterms*. T_{surf}, which is also first used in Section 3.2, is strictly more expressive than T_{bow}, which can be written as $T_{surf} \supset T_{bow}$.

$$T_{surf} = (Te,\, Tr)$$

$$Te = \left\{ \begin{array}{l} (annotation,\, \emptyset), \\ (\texttt{sentence},\, annotation), \\ (keyterm,\, \emptyset), \\ (k \in V,\, keyword) \end{array} \right\};\, Tr = \left\{ \begin{array}{l} (enclosure,\, annotation,\, keyterm), \\ (precedence, keyterm, keyterm) \end{array} \right\}$$

Figure A.3: Definition of a type system that supports bag-of-words constraints as well as surface patterns, implemented as *precedence* relations between pairs of *keyterms.*

A.3 Semantic Role Labeling Type System T_{srl}

In addition to bag-of-words constraints and surface patterns, this type system defines specific element types to model verbs and their semantic *arguments*. Verbs are referred to as `targets`. The element type `agent` is used to identify the actor or initiator of the event described by a `target` verb. The thing that is acted on is referred to as the `patient`. *Arguments* are associated with their respective `targets` using the *attachment* relation. Other element and relation types are the same as those defined in $T_{surf} \subset T_{srl}$.

$$T_{srl} = (Te, Tr)$$

$$Te = \left\{ \begin{array}{l} (annotation, \emptyset), \\ (\texttt{sentence}, annotation), \\ (\texttt{target}, annotation), \\ (argument, annotation), \\ (\texttt{agent}, argument), \\ (\texttt{patient}, argument), \\ (keyterm, \emptyset), \\ (k \in V, keyterm) \end{array} \right\} ; Tr = \left\{ \begin{array}{l} (enclosure, \texttt{sentence}, \texttt{target}), \\ (enclosure, \texttt{sentence}, argument), \\ (enclosure, annotation, keyterm), \\ (precedence, keyterm, keyterm), \\ (attachment, \texttt{target}, argument) \end{array} \right\}$$

Figure A.4: Definition of a type system that supports bag-of-words constraints, sentence segmentation, surface patterns for *keyterms*, and a simplified version of semantic role labeling.

A.4 BBN Identifinder Type System T_{bbn}

The BBN Identifinder [4] is a Named Entity Recognition tool described in Section 4.2.2. It is used in the English-language experiments in this thesis. First introduced in Section 5.4.1 The type system defined here, $T_{bbn} \supset T_{bow}$, adds an element type called *entity* and four element types that inherit from it: `date`, `location`, `org` and `person`. *Entities* are *annotations*, so they can participate in *enclosure* relations with *keyterms*. An additional *enclosure* relation is introduced to support containment of *entities* within the `sentence`. Note that, for brevity, this type system does not define all possible element subtypes of *entity* that can be recognized by the BBN Identifinder; the figure shows only those *entities* that occur in the English-language test collection used in this thesis. See Section 4.1.1 for more information about the test collection.

$$T_{bbn} = (Te, Tr)$$

$$Te = \left\{ \begin{array}{l} (\mathit{annotation}, \emptyset), \\ (\texttt{sentence}, \mathit{annotation}), \\ (\mathit{entity}, \mathit{annotation}), \\ (\texttt{date}, \mathit{entity}), \\ (\texttt{location}, \mathit{entity}), \\ (\texttt{org}, \mathit{entity}), \\ (\texttt{person}, \mathit{entity}), \\ (\mathit{keyterm}, \emptyset), \\ (k \in V, \mathit{keyterm}) \end{array} \right\}; Tr = \left\{ \begin{array}{l} (\mathit{enclosure}, \texttt{sentence}, \mathit{entity}), \\ (\mathit{enclosure}, \mathit{annotation}, \mathit{keyterm}) \end{array} \right\}$$

Figure A.5: Definition of a type system based T_{bow} that adds support for the output of the BBN Identifinder Named Entity Recognition tool.

A.5 ASSERT Type System T_{assert}

The ASSERT Shallow Semantic Parser [50], described in Section 4.2.3, identifies verb predicate-argument structures in text, and labels arguments with semantic roles. This section defines a type system, first referenced in Section 5.7, called $T_{assert} \supset T_{bow}$ that is similar in spirit to T_{srl}, but that supports many more types of *arguments*. $T_{assert} \not\supset T_{srl}$ because the element types `agent` and `patient` become their ASSERT equivalents, `arg0` and `arg1`, but the `target` verb element type and *attachment* relation type defined here are same as those defined in T_{srl}.

$$T_{assert} = (Te,\, Tr)$$

$$Te = \left\{ \begin{array}{l} (annotation,\, \emptyset), \\ (\texttt{sentence},\, annotation), \\ (\texttt{target}, annotation), \\ (argument,\, annotation), \\ (\texttt{arg0},\, argument), \\ (\texttt{arg1},\, argument), \\ (\texttt{arg2},\, argument), \\ (\texttt{arg3},\, argument), \\ (\texttt{arg4},\, argument), \\ (\texttt{argm-adv},\, argument), \\ (\texttt{argm-dir},\, argument), \\ (\texttt{argm-loc},\, argument), \\ (\texttt{argm-mnr},\, argument), \\ (\texttt{argm-tmp},\, argument) \\ (keyterm,\, \emptyset), \\ (k \in V,\, keyterm) \end{array} \right\} ;\, Tr = \left\{ \begin{array}{l} (enclosure,\, \texttt{sentence},\, \texttt{target}), \\ (enclosure,\, \texttt{sentence},\, argument), \\ (enclosure,\, argument,\, \texttt{target}), \\ (enclosure,\, argument,\, argument), \\ (enclosure,\, annotation,\, keyterm), \\ (attachment,\, \texttt{target},\, argument) \end{array} \right\}$$

Figure A.6: Definition of a type system based T_{bow} that adds support for the form of semantic role labeling produced by the ASSERT Shallow Semantic Parser.

A.6 BBN and ASSERT Type System $T_{bbn+assert}$

The type system defined in this section, $T_{bbn+assert} = T_{bbn} \cup T_{assert}$, is used in the English-language structured retrieval experiments in this thesis and is first referenced in Section 5.2. Note that, as used in Chapter 5, $T_{bbn+assert}$ does not define the *keyterm precedence* relation type; in Chapter 6, the *precedence* relation is used.

$$T_{bbn+assert} = (Te, Tr)$$

$$Te = \left\{ \begin{array}{l} (annotation, \emptyset), \\ (\texttt{sentence}, annotation), \\ (\texttt{target}, annotation), \\ (argument, annotation), \\ (entity, annotation), \\ (\texttt{date}, entity), \\ (\texttt{location}, entity), \\ (\texttt{org}, entity), \\ (\texttt{person}, entity), \\ (\texttt{arg0}, argument), \\ (\texttt{arg1}, argument), \\ (\texttt{arg2}, argument), \\ (\texttt{arg3}, argument), \\ (\texttt{arg4}, argument), \\ (\texttt{argm-adv}, argument), \\ (\texttt{argm-dir}, argument), \\ (\texttt{argm-loc}, argument), \\ (\texttt{argm-mnr}, argument), \\ (\texttt{argm-tmp}, argument) \\ (keyterm, \emptyset), \\ (k \in V, keyterm) \end{array} \right\}; Tr = \left\{ \begin{array}{l} (enclosure, \texttt{sentence}, \texttt{target}), \\ (enclosure, \texttt{sentence}, argument), \\ (enclosure, \texttt{sentence}, entity), \\ (enclosure, argument, \texttt{target}), \\ (enclosure, argument, argument), \\ (enclosure, argument, entity), \\ (enclosure, annotation, keyterm), \\ (attachment, \texttt{target}, argument), \\ (precedence, keyterm, keyterm) \end{array} \right\}$$

Figure A.7: Definition of a type system supporting both BBN Identifinder Named Entity Recognition and ASSERT Semantic Role Labeling: $T_{bbn+assert} = T_{bbn} \cup T_{assert}$.

A.7 Surface Patterns and BBN Type System $T_{surf+bbn}$

The type system defined in this section, $T_{surf+bbn} = T_{surf} \cup T_{bbn}$, can be considered a stronger baseline than T_{bbn}. It is used in place of T_{bbn} in a follow-up experiment described in Section 5.6, which compares the performance of structured queries formulated under $T_{bbn+assert}$ against this stronger baseline.

$$T_{surf+bbn} = (Te, Tr)$$

$$Te = \left\{ \begin{array}{l} (annotation, \emptyset), \\ (\texttt{sentence}, annotation), \\ (entity, annotation), \\ (\texttt{date}, entity), \\ (\texttt{location}, entity), \\ (\texttt{org}, entity), \\ (\texttt{person}, entity), \\ (keyterm, \emptyset), \\ (k \in V, keyterm) \end{array} \right\}; Tr = \left\{ \begin{array}{l} (enclosure, \texttt{sentence}, entity), \\ (enclosure, annotation, keyterm), \\ (precedence, keyterm, keyterm) \end{array} \right\}$$

Figure A.8: Definition of a type system based T_{surf} with support for the output of the BBN Identifinder Named Entity Recognition tool.

A.8 Chaos Named Entities Type System T_{chaos_ne}

First mentioned in Section 7.2, the type system $T_{chaos\_ne} \supset T_{bow}$ supports Italian-language Named Entity Recognition provided by the Chaos parser, described in Section 4.3.2. In the same way as T_{bbn}, defined in Section A.4, T_{chaos_ne} does not encode all of the named entity types recognizable by Chaos, but instead, only those referenced in the Italian-language test collection discussed in Section 4.1.2. In fact, the only difference between T_{chaos_ne} and T_{bbn} is that the *entity* type that BBN Identifinder calls `org`, is called `organisation` by Chaos.

$$T_{chaos\_ne} = (Te, Tr)$$

$$Te = \left\{ \begin{array}{l} (annotation, \emptyset), \\ (\texttt{sentence}, annotation), \\ (entity, annotation), \\ (\texttt{date}, entity), \\ (\texttt{location}, entity), \\ (\texttt{organisation}, entity), \\ (\texttt{person}, entity), \\ (keyterm, \emptyset), \\ (k \in V, keyterm) \end{array} \right\} \quad Tr = \left\{ \begin{array}{l} (enclosure, \texttt{sentence}, entity), \\ (enclosure, annotation, keyterm) \end{array} \right\}$$

Figure A.9: Definition of a type system supporting Italian-language Named Entity Recognition by Chaos.

A.9 Chaos Type System T_{chaos}

The type system defined here $T_{chaos} \supset T_{chaos\_ne}$ can represent verbs and phrase labeled with grammatical functions such as subject (`V_Sog`), direct object (`V_Obj`), indirect object (`V_NP`), oblique/prepositional (`V_PP`), adverbial (`V_Adv`), as well as via a relative pronoun (`V_PrRel`). These are syntactic designations not to be confused with semantic role labels. To clarify this distinction, verbs under T_{chaos} are labeled `verb`, as opposed to the `target` nomenclature that T_{srl} and T_{assert} uses. The label `verb` is an abstraction over the actual labels Chaos produces, such as `VerFin` for finite verbs and `VerGer` for gerunds. The type system is first referenced in Section 7.2.

$$T_{chaos} = (Te, Tr)$$

$$Te = \left\{ \begin{array}{l} (\mathit{annotation}, \emptyset), \\ (\texttt{sentence}, \mathit{annotation}), \\ (\texttt{verb}, \mathit{annotation}), \\ (\mathit{function}, \mathit{annotation}), \\ (\mathit{entity}, \mathit{annotation}), \\ (\texttt{V\_Sog}, \mathit{function}), \\ (\texttt{V\_Obj}, \mathit{function}), \\ (\texttt{V\_NP}, \mathit{function}), \\ (\texttt{V\_PP}, \mathit{function}), \\ (\texttt{V\_Adv}, \mathit{function}), \\ (\texttt{V\_PrRel}, \mathit{function}), \\ (\texttt{date}, \mathit{entity}), \\ (\texttt{location}, \mathit{entity}), \\ (\texttt{organisation}, \mathit{entity}), \\ (\texttt{person}, \mathit{entity}), \\ (\mathit{keyterm}, \emptyset), \\ (k \in V, \mathit{keyterm}) \end{array} \right\} \quad Tr = \left\{ \begin{array}{l} (\mathit{enclosure}, \texttt{sentence}, \mathit{entity}), \\ (\mathit{enclosure}, \texttt{sentence}, \texttt{verb}), \\ (\mathit{enclosure}, \texttt{sentence}, \mathit{function}), \\ (\mathit{enclosure}, \mathit{function}, \mathit{entity}), \\ (\mathit{enclosure}, \mathit{function}, \texttt{verb}), \\ (\mathit{enclosure}, \mathit{function}, \mathit{function}), \\ (\mathit{enclosure}, \mathit{annotation}, \mathit{keyterm}) \\ (\mathit{attachment}, \texttt{verb}, \mathit{function}) \end{array} \right\}$$

Figure A.10: Definition of a type system supporting Named Entity Recognition, as well as syntactic chunking with labeling of grammatical functions provided by the Chaos Parser.

Bibliography

[1] Steven Abney, Michael Collins, and Amit Singhal. Answer extraction. In *Proceedings of the Sixth Conference on Applied Natural Language Processing (ANLP)*, 2000. 7.7

[2] M. J. Atallah, editor. *Algorithms and Theory of Computation Handbook*. CRC Press LLC, 1999. 1

[3] Roberto Basili and Fabio Massimo Zanzotto. Parsing engineering and empirical robustness. *Journal of Natural Language Engineering*, 8(2–3), June 2002. 4.3.2

[4] D. Bikel, R. Schwartz, and R. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, 34(1–3):211–231, 1999. 4.2.2, A.4

[5] M. Bilotti. Query expansion techniques for question answering. Master's thesis, Massachusetts Institute of Technology, 2004. 4.1.1

[6] M. Bilotti, J. Elsas, J. Carbonell, and E. Nyberg. Passage ranking for question answering
using linguistic and semantic features. In *Proceedings of the 32nd European Conference on Information Retrieval (ECIR)*, 2010. (Submitted). (document), 4.4.2, 4.1

[7] M. Bilotti, B. Katz, and J. Lin. What works better for question answering: Stemming or morphological query expansion? In *Proceedings of the Information Retrieval for Question Answering (IR4QA) Workshop at SIGIR 2004*, 2004. 1.2, 1.3, 4.1.1

[8] M. Bilotti, P. Ogilvie, J. Callan, and E. Nyberg. Structured retrieval for question answering. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007. 4.4.1, 5, 5.1, 5.4, 2, 5.4.2, 6.7

[9] J. Callan. Passage-level evidence in document retrieval. In *Proceedings of SIGIR'94*, 1994. 2.2

[10] J. P. Callan, W. B. Croft, and S. M. Harding. The INQUERY retrieval system. In *Proceedings the 3rd International Conference on Database and Expert Systems Applications*, 1992. 4.4.1

[11] C. L. A. Clarke, G. V. Cormack, D. I. E. Kisman, and T. R. Lynam. Question answering by passage selection (multitext experiments for trec-9). In *Proceedings of the Ninth Text REtrieval Conference (TREC 2000)*, 2000. 1.3

[12] Michael Collins. Ranking algorithms for named-entity extraction: boosting and the voted perceptron. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 489–496, 2001. 4.4.2

[13] W. Bruce Croft. Learning about ranking and retrieval models. In *Proceedings of the SIGIR 2007 Workshop on Learning to Rank for Information Retrieval (LR4IR 2007*, 2007. (Keynote). 1.5, 8.3

[14] Hang Cui, Renxu Sun, Keya Li, Min-Yen Kan, and Tat-Seng Chua. Question answering passage retrieval using dependency relations. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2005. (document), 2.2, 6.8, 6.3

[15] I. Dagan, O. Glickman, and B. Magnini. The pascal recognising textual entailment challenge. *Machine Learning Challenges. Lecture Notes in Computer Science*, 3944:177–190, 2006. 2.5, 5

[16] Jonathan L. Elsas, Vitor R. Carvalho, and Jaime G. Carbonell. Fast learning of document ranking functions with the committee perceptron. In *Proceedings of the First ACM International Conference on Web Search and Data Mining (WSDM 2008)*, 2008. 4.4.2, 6.1, 6.5, 7.4, 8.4

[17] C. Fellbaum. *WordNet, an Electronic Lexical Database.* MIT Press, Cambridge, MA, 1998. 2.2, 2.3

[18] S. I. Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191, 1990. 4.4.2

[19] D. Graff. *The AQUAINT Corpus of English News Text.* Linguistic Data Consortium (LDC), 2002. Cat. No. LDC2002T31. 4.1.1

[20] Bert F. Green, Jr., Alice K. Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question answerer. In E. A. Figenbaum and J. Feldman, editors, *Computers and Thought*, pages 207–216. McGraw-Hill, New York, 1963. 2.1

[21] M. Greenwood. Using pertainyms to improve passage retrieval for questions requesting information about a location. In *Proceedings of the SIGIR Workshop on Information Retrieval for Question Answering*, 2004. 1.3, 2.2

[22] Lynette Hirschman and Robert Gaizauskas. Natural language question answering: The view from here. *Journal of Natural Language Engineering, Special Issue on Question Answering*, Fall–Winter 2001. 1.1, 2.1

[23] E. Hovy. Question answering in webclopedia. In *Proceedings of TREC'00*, 2000. 2.2

[24] Kevin Humphreys, Robert Gaizauskas, Mark Hepple, and Mark Sanderson. University of sheffield trec-8 q & a system. In *Proceedings of the Fifteenth Text REtrieval Conference (TREC)*, 1999. 1.2

[25] Thorsten Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, 2002. 4.4.2

[26] Boris Katz, Jimmy Lin, Daniel Loreto, Wesley Hildebrandt, Matthew Bilotti, Sue Felshin, Aaron Fernandes, Gregory Marton, and Federico Mora. Integrating web-based and corpus-based techniques for question answering. In *Proceedings of the 12th Text REtrieval Conference (TREC 2003)*, 2003. 1.3

[27] P. Kingsbury, M. Palmer, and M. Marcus. Adding semantic annotation to the penn treebank. In *Proceedings of the 2nd International Conference on Human Language Technology Research (HLT 2002)*, 2002. 4.2.3, 5.7

[28] J. Kupiec. Murax: a robust linguistic approach for question answering using an on-line encyclopedia. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 181–190, 1993. 2.1

[29] V. Lavrenko and W. B. Croft. Relevance-based language models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2001. 4.4.1

[30] Dekang Lin. Dependency-based evaluation of minipar. In *Proceedings of the Workshop on the Evaluation of Parsing Systems*, 1998. 2.2

[31] J. Lin. The role of information retrieval in answering complex questions. In *Proceedings of COLING-ACL'06*, 2006. (Poster.). 2.2

[32] J. Lin and B. Katz. Building a reusable test collection for question answering. *Journal of the American Society for Information Science and Technology*, 57(7):851–861, 2006. 4.1.1

[33] M. Marcus, M. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19(2):313–330, 1993. 4.2.3, 5.7

[34] D. Metzler and W. B. Croft. Combining the language model and inference network approaches to retrieval. *Information Processing & Management Special Issue on Bayesian Networks and Information Retrieval*, 40(5):735–750, 2004. 4.4.1

[35] Dan Moldovan, Marius Paşca, Sanda Harabagiu, and Mihai Surdeanu. Performance issues and error analysis in an open-domain question answering system. *ACM Transactions on Information Systems*, 21(2), 2003. 1.2

[36] C. Monz. *From Document Retrieval to Question Answering*. PhD thesis, Universiteit van Amsterdam, 2003. 1.3

[37] Christof Monz. Document retrieval in the context of question answering. In *Proceedings of ECIR'03*, 2003. 2.2

[38] Christof Monz. Minimal span weighting retrieval for question answering. In *Proceedings of the SIGIR Workshop on Information Retrieval for Question Answering*, 2004. 2.2

[39] Alessandro Moschitti. Making tree kernels practical for natural language learning. In *Proceedings of the Eleventh International Conference on European Association for Computational Linguistics*, 2006. 6.4

[40] S. Narayanan and S. Harabagiu. Question answering based on semantic structures. In *Proceedings of COLING'04*, 2004. 2.2

[41] Eric Nyberg, Robert Frederking, Teruko Mitamura, Matthew Bilotti, Kerry Hannan, Laurie Hiyakumoto, Jeongwoo Ko, Frank Lin, Lucian Lita, Vasco Pedro, and Andrew Schlaikjer. Javelin i and ii systems at trec 2005. In *Proceedings of the 14th Text REtrieval Conference (TREC 2005)*, 2005. 2.2, 2.3

[42] Eric Nyberg, Teruko Mitamura, Robert Frederking, Vasco Pedro, Matthew Bilotti, Andrew Schlaikjer, and Kerry Hannan. Extending the javelin qa system with domain semantics. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 2005)*, 2005. 2.2, 2.3

[43] P. Ogilvie and J. Callan. Hierarchical language models for retrieval of xml components. In *Proceedings of the Initiative for the Evaluation of XML Retrieval Workshop (INEX 2004)*, 2004. 4.4.1, 5.1

[44] Marius Paşca and Sanda Harabagiu. High performance question/answering. In *Proceedings of SIGIR'01*, 2001. 1.3, 2.2

[45] A. Phillips. A question-answering routine. Memo 16, Artificial Intelligence Project, 1960. 2.1

[46] Emanuele Pianta, Christian Girardi, and Roberto Zanoli. The textpro tool suite. In *Proceedings of LREC, 6th edition of the Language Resources and Evaluation Conference*, 2008. 4.3.1

[47] Luiz Augusto Pizzato and Diego Mollá. Indexing on semantic roles for question answering. In *Proceedings of the Second Information Retrieval for Question Answering (IR4QA) Workshop at COLING 2008*, 2008. 2.3

[48] J. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1998. 4.4.1

[49] S. Pradhan, W. Ward, K. Hacioglu, J. Martin, and D. Jurafsky. Shallow semantic parsing using support vector machines. In *Proc. of HLT/NAACL 2004*, 2004. 2.2

[50] S. Pradhan, W. Ward, K. Hacioglu, J. Martin, and D. Jurafsky. Shallow semantic parsing using support vector machines. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2004)*, 2004. 4.2.3, A.5

[51] J. Prager, E. Brown, A. Coden, and D. Radev. Question-answering by predictive annotation. In *Proceedings of SIGIR'00*, 2000. 2.2, 7.7

[52] J. Reynar and A. Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, 1997. 4.2.1

[53] Nico Schlaefer, Petra Gieselmann, and Guido Sautter. The ephyra qa system at trec 2006. In *Proceedings of the Fifteenth Text REtrieval Conference (TREC)*, 2006. 7.7

[54] R. Simmons, S. Klein, and K. McConlogue. Indexing and dependency logic for answering english questions. *American Documentation*, 15(3):196–204, 1963. 2.1

[55] Mark D. Smucker, James Allan, and Ben Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management (CIKM)*, pages 623–632, 2007. 5.5, 5.5, 5.6, 5.7, 6.6, 7.5, 7.7

[56] T. Strohman, D. Metzler, H. Turtle, and W. B. Croft. Indri: A language model-based search engine for complex queries. In *Proceedings of the International Conference on Intelligence Analysis*, 2005. 4.4.1

[57] Stefanie Tellex, Boris Katz, Jimmy Lin, Aaron Fernandes, and Gregory Marton. Quantitative evaluation of passage retrieval algorithms for question answering. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 41–47, New York, NY, USA, 2003. ACM Press. 1.3, 2.2

[58] J. Tiedemann. Integrating linguistic knowledge in passage retrieval for question answering. In *Proceedings of the 2005 Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP 2005)*, 2005. 2.2

[59] J. Tiedemann. Comparing document segmentation strategies for passage retrieval in question answering. In *Proceedings of the Conference on Recent Advances in Natural Language Processing (RANLP'07)*, 2007. 1.3

[60] H. Turtle and W. B. Croft. Evaluation of an inference network-based retrieval model. *ACM Transactions on Information Systems*, 9(3):187–222, 1991. 4.4.1

[61] Suzan Verberne, Hans van Halteren, Stephan Raaijmakers, Daphne Theijssen, and Lou Boves. Learning to rank qa data. In *Proceedings of the Learning to Rank Workshop at SIGIR 2009*, pages 41–48, 2009. 2.3

[62] E. Voorhees, N. Gupta, and B. Johnson-Laird. The collection fusion problem. In *Proceedings of the Third Text REtrieval Conference (TREC-3)*, 1994. 5.4.2

[63] Ellen M. Voorhees. The trec-8 question answering track report. In *Proceedings of the Eighth Text REtrieval Conference (TREC-8)*, 1999. 1

[64] E. Wendlandt and J. Driscoll. Incorporating a semantic analysis into a document retrieval strategy. In *Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 270–279, 1991. 2.1, 2.2

[65] William A. Woods, R. M. Kaplan, and B. L. Nash-Webber. The lunar sciences natural language information system. Final Report 2378, BBN, Cambridge, MA, 1972. 2.1

[66] Le Zhao and Jamie Callan. Dirichlet smoothing for correcting bias towards short fields. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM 2008)*, 2008. 2, 5.8

[67] Le Zhao and Jamie Callan. Effective and efficient structured retrieval. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM 2009)*, 2009. (To appear.). 1, 5.8