Exploiting Syntactic Relations for Question Answering

by

Daniel Loreto

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2006

Author .....
Department of Electrical Engineering and Computer Science
September 20, 2006

Certified by... .....
Boris Katz
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Exploiting Syntactic Relations for Question Answering

by

Daniel Loreto

Submitted to the Department of Electrical Engineering and Computer Science
on September 20, 2006, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Recently there has been a resurgent interest in syntax-based approaches to information access, as a means of overcoming the limitations of keyword-based approaches. So far attempts to use syntax have been *ad hoc*, choosing to use some syntactic information but still ignoring most of the tree structure. This thesis describes the design and implementation of SMARTQA, a proof-of-concept question answering system that compares syntactic trees in a principled manner. Specifically, SMARTQA uses a tree edit-distance algorithm to calculate the similarity between unordered, unrooted syntactic trees. The general case of this problem is NP-complete; in practice, SMARTQA demonstrates that an optimized implementation of the algorithm can be feasibly used for question answering applications.

Thesis Supervisor: Boris Katz
Title: Principal Research Scientist

Acknowledgments

Special thanks go to:

- Mom and Dad for always being there for me and constantly showing me their support.
- Boris Katz for being a superb advisor. Boris, I hope this brings QA a step closer to our dream.
- Roni Katzir for invaluable advice during all phases of this thesis, and for the late-night conversations about projects that are still to come. Stay tuned, this is just the beginning.
- Sue Felshin, Gary Borchardt, Gregory Marton and Yuan Shen, for their valuable suggestions during the writing and editing of this document.
- To the rest of the Infolab group: Federico Mora, Gabriel Zaccak, Jimmy Lin, Ben Lu and all others that made our corner of the Stata Center such an enjoyable place to work.
- To MIT for the resources and education it provided me with.

Contents

List of Textual Examples

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Dream

In the last couple of decades the amount of digital information has been increasing dramatically. The push towards digitization has been driven by the advantages it offers over conventional media: massive repositories of knowledge can be stored efficiently and information within them can be retrieved rapidly.

Extrapolating from this trend, one can't help but dream of a world where users can access precisely the information they need exactly when they need it. While current technology has certainly taken us closer to that dream, the ideal remains out of reach—it is often quite hard to find the information one needs amongst the overwhelming amount of data that is available.

An ideal information access system would allow users to express requests for information in natural terms and would have human-level understanding of the information in its knowledge base. Upon receiving a request from a user, it would return precisely the information that is relevant to that request. Neither more nor less.

I believe that ultimately such an ideal system will be in the form of a question answering system. To this end, this thesis contends that question answering systems can and should exploit syntactic information to better understand their knowledge base and ultimately provide users with more relevant answers.

1.2 Traditional Information Access

Most currently deployed information access systems attack the problem from an information retrieval (IR) perspective by considering indexed documents as bags of words. Under such a scheme, the semantic content of a document is equated with the set of keywords it contains regardless of their order or interrelationships.

Unfortunately, keywords alone are not enough to capture the full expressiveness of language. Word order and syntactic relationships are important when determining the meaning of a sentence. Consider for example the following pairs of sentences and phrases with essentially the same keyword content but different meanings:

(1.1) a) France defeated Brazil at the World Cup Final.

b) Brazil defeated France at the World Cup Final.

(1.2) a) Video of kidnapped journalists being released.

b) Kidnapped journalists released with video.

A keyword-based system would treat each of those pairs as desirable responses to the same query. A user would probably consider only one of the entries in each pair to be relevant. To illustrate: if the user is interested in knowing the answer to "Who defeated Brazil?" then the sentence "France defeated Brazil at the World Cup Final." would be relevant, but "Brazil defeated France at the World Cup Final." would not be.

To improve their performance, IR systems employ several techniques. Some of the most common ones include removing stopwords[1], stemming terms[2], and weighing keywords according to $tf \cdot idf$[3] [44]. The most significant improvements of IR technology in recent years have come from exploiting structured and semi-structured

[1]Stopwords are common words without useful semantic content, such as "the", "a", "of", and "in".

[2]Removing the endings of words to keep only their root or *stem*. For example, "eat", "eats", "eating", and "ate" are all stemmed to "eat" and considered equivalent.

[3]The frequency of a term in a given document (*term frequency* or tf) times its inverse document frequency (idf). Intuitively, if a word is rare in general, but frequent in a document, it is an important indicator of the meaning of the document.

information. Witness, for example, the highly successful Google search engine, which weighs documents according to the hyperlink structure of the web to obtain better results [4]. Unfortunately, information we are interested in is often found as free-form, natural language text without any explicit structure. Even in semi-structured documents, such as HTML and XML, most of the relevant content is in a natural language. Unlike traditional IR approaches that only exploit the few structural elements made explicit in a document, language-based approaches are in a position to exploit the significant amount of latent structure hidden within the grammar of English sentences.

1.3 Question Answering

The limitations of traditional IR should be enough motivation to pursue a more linguistically motivated approach to information access. While one could simply try to extend a traditional IR system with language-based heuristics, it would be more desirable to make natural language understanding a cornerstone of information access.

Question answering (QA) systems take such an approach by using natural language, both in their answer selection mechanisms and in the interface they use to communicate with their users. By allowing users to issue information requests in human terms, QA systems present an intuitive interface that is easy to deploy and has virtually no learning curve.

Despite the obvious advantages that an ideal QA system would bring, researchers are currently not in a position to develop an automated system capable of human-level understanding. Current technology is too primitive to cope with the complexity of natural languages—we still lack proper ways of dealing with intersentential references, common-sense implication, and word sense disambiguation to name a few open problems.

We are in a position, however, to reach a promising middle ground: we can develop systems that exploit the syntactic information contained within natural language sentences. Parsing technology has made great strides in recent years, bringing large-

scale syntactic analysis within the realm of possibility. This thesis demonstrates that it is possible to use a principled approach for comparing syntactic trees in order to improve the current state of the art in information access.

Chapter 2

Exploiting Syntax for Question Answering

The main contribution of this thesis is the design and implementation of SMARTQA, a proof-of-concept question answering system that relies heavily on syntax to determine answers.

Today's state-of-the-art question answering systems are rather complex, containing dozens of modules performing various different tasks (information retrieval, question-type analysis, anaphora resolution, named entity recognition, inferencing, etc.). Given the complexity of current systems, it is often difficult to understand how each different module affects performance. In order to fully understand the potential impact that syntax-driven approaches can have in question answering, SMARTQA departs from this complex architecture, instead selecting answers based solely on a dynamic programming algorithm for comparing syntactic trees. Nevertheless, the system was designed with extensibility in mind; in the future it can be easily extended into a full-fledged question answering system that incorporates many of these additional modules.

In its crudest form, a QA system consists only of two modules: an IR-type module that selects candidate information fragments from its knowledge base, and an answer selection module that evaluates each of those fragments and assigns them a score based on how likely they are to answer the posed question.

If we restrict our information fragments to sentences, our problem reduces to that of (i) finding an appropriate representation for questions and candidate sentences and (ii) defining a similarity measure between those representations such that sentences containing the right answer are deemed highly similar to the question, and sentences not containing the answer are deemed highly dissimilar.

2.1 Design Principles

- **Wide Coverage.**

  Often linguistically motivated question answering systems are built to perform very well within a limited domain, but their performance degrades significantly when they are extended to handle arbitrary questions. Much of the usefulness of an information access system derives from its ability to handle a wide variety of knowledge requests; SMARTQA was therefore designed to handle questions in unrestricted domains. The only assumption that SMARTQA makes about its input is that the user submits questions formulated in English. This requirement is, in fact, an advantage over traditional IR systems. Natural language questions contain the useful grammatical structure SMARTQA capitalizes on, and serve as an intuitive interface to information access.

- **Robustness to Variable Phrasing.**

  Natural language offers speakers many different ways to express the same ideas. SMARTQA is designed to be robust to variable phrasings. The system should still be able to find an answer even if the source of knowledge it is accessing does not contain the requested information with the exact same phrasing employed by the user when posing the question.

- **Principled.**

  Previous attempts at comparing syntactic trees have done so in an *ad hoc* manner, choosing, for example, to disregard most of the tree structure and computing instead the percentage of dependency triples that overlap between the

question and the candidate. SMARTQA uses a principled approach that takes into account the structure of the whole tree.

- **Formalism Independence.**

  The algorithm SMARTQA uses to compare syntactic trees is independent of any specific syntactic formalism. The only requirement placed on syntactic structures is that they be represented as trees. As linguists improve their theoretical understanding of grammar, SMARTQA can easily update its representation to reflect better theories.

- **Tunable Precision.**

  In most information access systems there is usually a tradeoff between recall and precision. The ideal balance between the two is often application dependent. In certain applications it is preferable to provide a user with no information than with misleading information. In others, it is preferable for users to sift through somewhat relevant information, and make their own judgments, than to obtain no information at all.

  SMARTQA provides a confidence score with each answer that, along with a tunable threshold, allows users to set the balance they prefer for their respective application. In addition SMARTQA returns a supporting source of knowledge with every answer so that users may inspect it if desired.

2.2 Knowledge Representation

In order to take advantage of syntax, SMARTQA moves away from the representation of sentences as a bag of words, and instead uses dependency trees—a tree-based structure with labeled edges that makes explicit the syntactic relationships between the different nodes. Section 2.2.2 describes dependency trees in more detail.

Questions are represented similarly, but before their corresponding trees are generated, they are converted into an assertive sentence with a variable. We perform the transformation to bring the syntactic structure of the question closer to that of

a candidate answer; essentially, the result of the transformation serves as a *template* against which we want to match candidate answers. The examples below show some questions with their corresponding assertive forms and expected answers.

(2.1) **Question:** Who invented the radio?

Template: ${who} invented the radio

Answer: Marconi invented the radio.[1]

(2.2) **Question:** When did the Iron Curtain fall?

Template: the Iron Curtain fell ${when}

Answer: The Iron Curtain fell in 1989.

(2.3) **Question:** What country invaded Italy in 1494?

Template: ${what country} invaded Italy in 1494

Answer: France invaded Italy in 1494.

Examples of questions transformed into assertive templates. Variables are denoted by a dollar sign and enclosing braces.

2.2.1 Linguistic Background

Linguists generally agree that the best way to represent the grammatical structure of a sentence is using a labeled ordered tree known as a *syntactic tree*[2]. The words of a sentence are represented as leaves in the tree. Mother[3] nodes represent *constituents*, groups of words and subconstituents that form a larger functional unit. Daughter nodes are referred to as *elements* of a constituent. Figure 2-1 shows an example of a syntactic tree.

[1]Sources disagree on whether the radio was actually invented by Marconi, Tesla or Popov.

[2]There is a wide variety of syntactic formalisms that use trees as their main representational structure, including Government Binding (GB), Head-Driven Phrase Structure Grammar (HPSG), Lexical Functional Grammar (LFG), and Tree Adjoining Grammar (TAG). Even some formalisms that don't naturally use trees in their formulation have been show to be equivalent to tree-based formalisms. Combinatory Categorical Grammar (CCG), for example, is weakly equivalent to TAG.

[3]It is traditional in the linguistic literature to refer to parent nodes and child nodes as mother nodes and daughter nodes respectively.

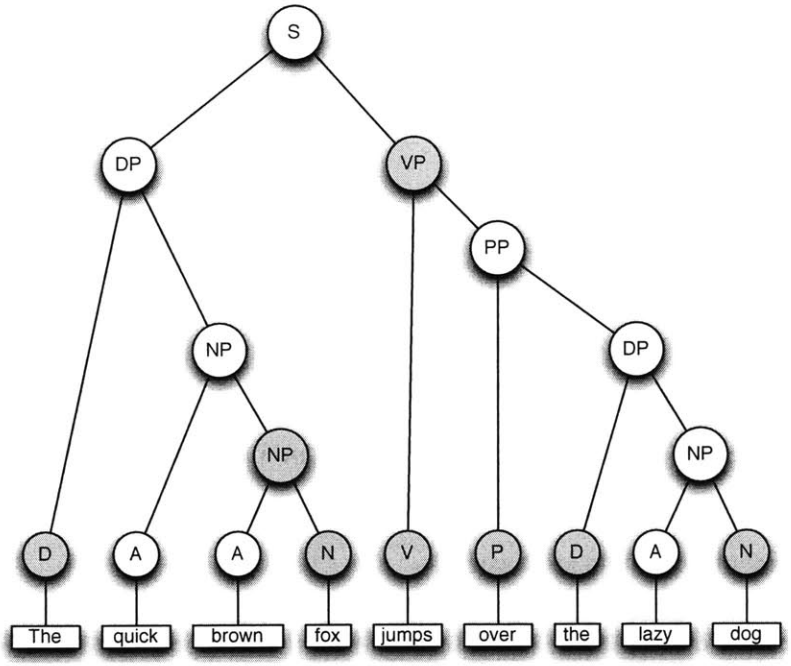


Figure 2-1: A syntactic tree for the sentence "The quick fox jumps over the lazy dog". Words are represented as rectangles and constituents are represented as circles labeled with their corresponding categories. Heads are depicted by the color gray.

Heads

Each constituent contains a *head*, a special element that determines the syntactic function of its mother and often determines the presence of other elements. Intuitively, the head of a constituent corresponds to the most important element, often the one that captures the most prominent semantic idea. In Figure 2-1 heads are denoted by the color gray; for example, the head of the noun phrase "lazy dog" is "dog".

Types of Constituents

Constituents are labeled with *categories* that indicate the syntactic function they perform. For example, phrases that have a noun as a head are known as noun phrases, and are labeled with the category NP. Similarly, phrases that have a verb as

a head are known as verb phrases and are labeled with the category VP. In Figure 2-1, "jumps over the lazy dog" is a verb phrase since it is headed by the verb "jumps".

Arguments and Modifiers

Besides heads, constituents have two other types of elements: arguments and modifiers.

Arguments are elements necessary to complete the meaning of the constituent; the sentence would be ungrammatical without them. In general, their semantic role depends not only on the contents of the argument but also on the head itself. The examples below show sentences with and without arguments. Notice that the versions without arguments are ungrammatical.

(2.4) a) Louise **abandoned** the project.

b) *Louise **abandoned**.

(2.5) a) Thelma **put** the text on the desk.

b) *Thelma **put** the text.

c) *Thelma **put** on the desk.

Examples of sentences with and without arguments (some examples taken from [12]). In each case the head of the relevant phrase is set in boldface and the arguments are underlined. Ungrammaticality is denoted by a * as is usual in linguistics.

Modifiers or *adjuncts* are optional elements of a constituent that modify its meaning. The sentence remains grammatical if they are removed, and in general their semantic role is independent of the head. The examples below show sentences with and without modifiers.

(2.6) a) Louise **abandoned** the project.

b) Louise **abandoned** the project on a Sunday.

(2.7) a) Thelma **put** the parcel on the desk.

b) Because it was heavy, Thelma **put** the parcel on the desk.

c) Thelma probably **put** the parcel on the desk.

d) Thelma **put** the parcel on the desk last night.

e) Because it was heavy, Thelma probably **put** the parcel on the desk last night.

(2.8) a) The **fox** jumps over the dog.

b) The quick **fox** jumps over the dog.

Examples of sentences with and without modifiers (some examples taken from [12]). In each case the head of the relevant phrase is set in boldface and the modifiers are underlined.

2.2.2 Dependency Trees

Syntactic trees, as described so far, make explicit the phrase structure of sentences and implicit the syntactic relations between a constituent's head and its other elements. For computational purposes, we are interested in making syntactic relations explicit even at the cost of making the phrase structure implicit. We therefore seek an alternate representation of grammatical structure known as a dependency tree.

A *dependency* [13] [32] is an asymmetric binary relationship between a word called the *head* and a word called the *dependent*. Dependencies usually have a label known as its *type*. A *dependency triple* is the 3-tuple formed by the head, the type and the dependent.

A *dependency tree* is a tree-based representation of a sentence where each node represents a word and each labeled, directed edge represents a dependency between two words. Figure 2-2 shows an example of a dependency tree. Note that in this representation there is an explicit link between a head and its adjuncts or arguments. For example, there's a labeled edge between "fox" and "quick" indicating that "quick" is an adjectival modification of "fox". Notice, also, that in this representation "the" plays a much less prominent role. In Figure 2-1 the word "the" is the head of both

DPs, in Figure 2-2 "the" is not the head, but rather the dependent, of the `det` dependency. This is because dependecy trees, in comparison to syntactic trees, emphasize semantics instead of phrase structure.

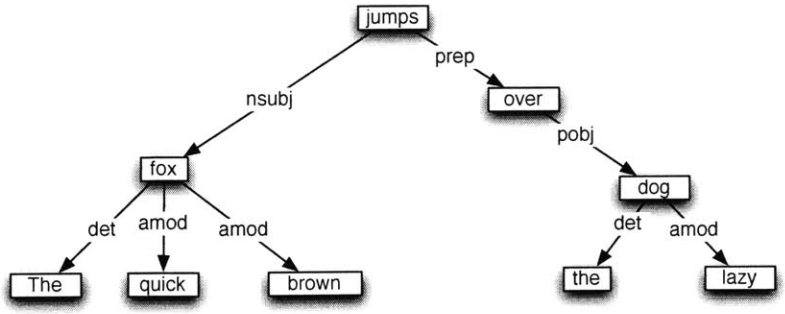


Figure 2-2: The dependency tree for the sentence "The quick fox jumps over the lazy dog". Words are represented as rectangles and dependencies are represented as labeled, directed edges.

2.3 Similarity Measure

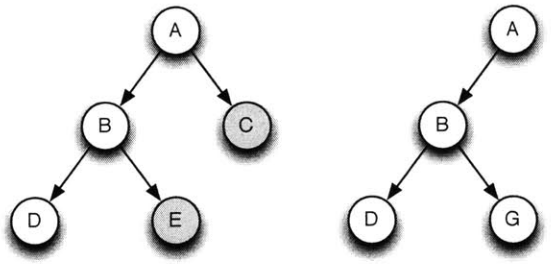
Given that both the question and a candidate sentence are represented as dependency trees, and that the question has been transformed into a template that a correct answer is expected to match, it is natural to choose a variant of tree edit-distance as the appropriate similarity measure.

2.3.1 Tree Edit-Distance and Variants

Tree edit-distance on ordered trees was originally studied by Tai [40] as a generalization of the well-known string edit-distance [46] or Levenshtein distance [24].

Given a small set of basic operations for transforming trees, where each operation has an associated cost, the edit-distance between two trees is defined as the minimum cost necessary to make the two trees identical. Trees can be transformed using only the given basic operations, and the total cost of a transformation is defined as the

sum of the cost of each of the operations applied. Figure 2-3 gives an example of the edit-distance between two trees.



(a) Source (b) Target

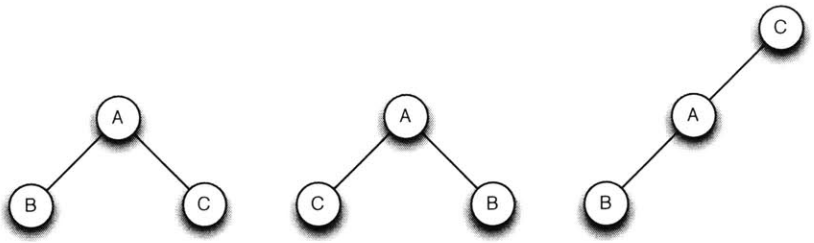
Figure 2-3: If we assume unit costs for each operation, the above trees have an edit-distance of 2: the source tree can be made identical to the target tree by performing a minimum of two modifications. Namely, node C should be deleted and node E should be relabeled as G.

In the original formulation of the problem, trees were rooted and ordered (see Figure 2-4 for the different types of trees); the set of basic operations consisted of node insertion, node deletion, and node relabeling (Figure 2-5 illustrates these operations). Since then several variants have been proposed (see [3] for a survey of the different variants). Proposed variants usually change the problem along two possible dimensions: (i) the types of the trees being compared, and (ii) the operations and costs permitted when modifying trees. The next section examines which variant is appropriate for question answering.

2.3.2 Requirements for Question Answering

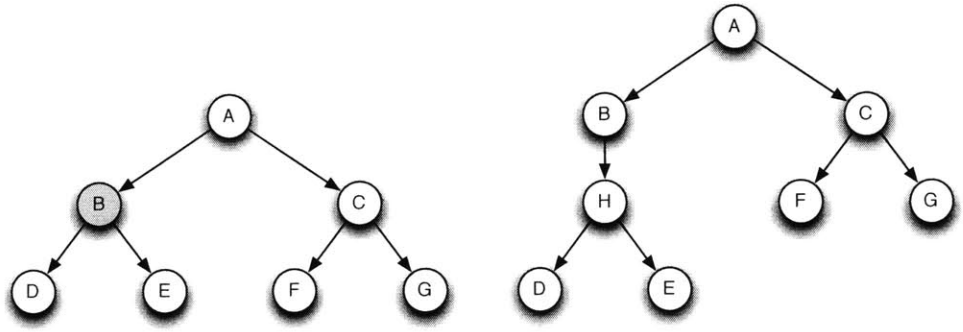
By looking at an example question and a corresponding answer, we can glean some of the properties necessary for a tree edit-distance variant to be useful for question answering.

Recall that we are converting a question into an assertion that serves as a template for possible answers. Intuitively, a candidate answer matches a template if they share the same concepts, and if those concepts are interrelated by semantically equivalent

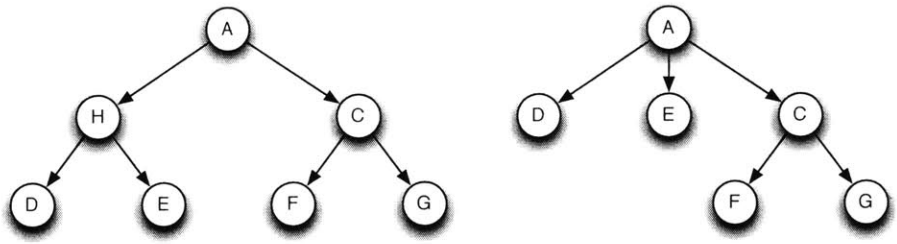


(a) (b) (c)

Figure 2-4: Trees can be rooted or unrooted, and ordered or unordered.
A tree is said to be *rooted* if it has a special node designated as the root or origin. If we consider the above trees to be rooted at their highest node, then (a) and (c) are different trees. If we consider the trees to be unrooted, however, (a) and (c) are identical trees.
A tree is said to be *ordered* if the order of a node's children is significant. If we consider the above trees to be ordered, then (a) and (b) are different trees. If we consider the trees to be unordered, however, (a) and (b) are identical trees.



(a) Original (b) Insertion



(c) Replacement (d) Deletion

Figure 2-5: Permitted operations in the original formulation of tree edit-distance. All operations are being performed on the grayed node.

syntactic relations.

We can model this intuition by *aligning* the template tree with the candidate tree. Two nodes are aligned if they represent the same concept. A *path* between two aligned nodes is the linear sequence of nodes and edges that connects the two. An alignment is considered desirable if the paths between aligned nodes in the template and aligned nodes in the candidate represent similar syntactic relationships.

Take for example the question "Who was the commander of the Russian submarine Kursk?", which has the assertive form "the commander of the Russian submarine Kursk was ${who}". A correct answer is found in the sentence "Gennady Lyachin, the Kursk commander, was awarded posthumously the title of Hero of Russia." by virtue of the alignment shown in Figure 2-6.

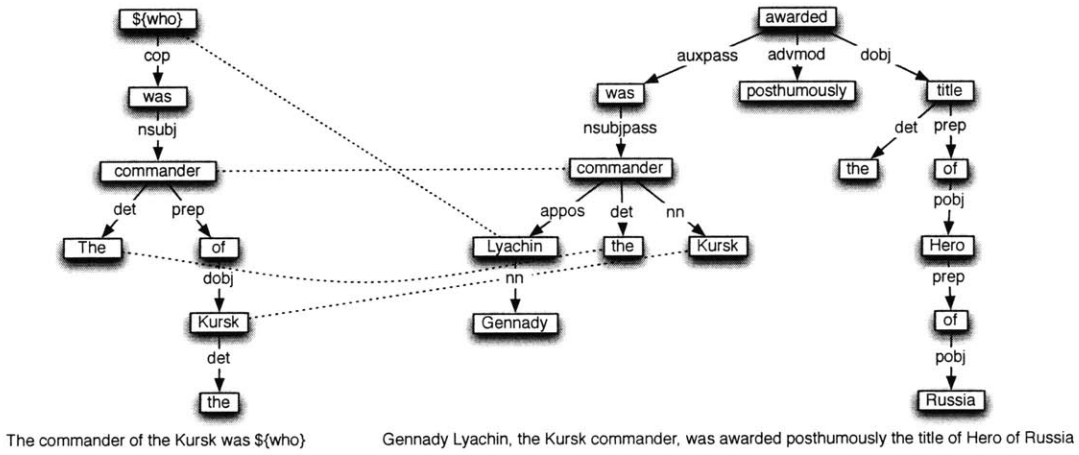


Figure 2-6: The ideal alignment between the template "the commander of the Russian submarine Kursk was ${who}" and the answer "Gennady Lyachin, the Kursk commander, was awarded posthumously the title of Hero of Russia.". Aligned nodes are connected using dotted lines. Observe that node alignments implicitly define alignments between paths. For example, `${who}` $\xrightarrow{\texttt{cop}}$ `was` $\xrightarrow{\texttt{nsubj}}$ `commander` is aligned with `Lyachin` $\xleftarrow{\texttt{appos}}$ `commander`.

From the discussion and example above, we infer the following properties about the tree-edit variant:

- **Permitted operations should reflect the alignment model.** Our version of tree edit-distance will have two permitted operations: node alignment, and

edge contraction. Node alignment pairs a node from the template with a node from the candidate. Pairings between semantically similar nodes are desirable. Edge contraction substitutes a node by one of its neighbors, destroying at the same time the edge that connects them. In general, contractions are undesirable; their application implies that the syntactic relationships between two aligned nodes do not match perfectly. Note however, that not all contractions should be penalized as heavily as others; some are necessary to find appropriate alignments. For example, in order to find the alignment shown in Figure 2-6, a tree edit-distance algorithm would have to first contract the `dobj` edge in the template path `commander` $\xrightarrow{\texttt{prep}}$ `of` $\xrightarrow{\texttt{dobj}}$ `Kursk` so that the path becomes `commander` $\xrightarrow{\texttt{prep}}$ `Kursk` (bringing it closer to the corresponding candidate path `commander` $\xrightarrow{\texttt{nn}}$ `Kursk`).

Notice that the type of the syntactic dependencies between nodes is important. Without them, the alignment in Figure 2-7 would be more desirable than the one in Figure 2-6 since the latter also finds an alignment for the "was" node. To simplify the tree edit-distance algorithm and obviate the need for another operation, the implementation described in Chapter 4 takes into account any relevant neighboring syntactic dependencies within the node alignment cost. An alternative would be to make all edges unlabeled, and instead model syntactic types as additional nodes. Either way, the theoretical exposition of the algorithm can ignore edge labels without any loss of generality.

- **Templates often match subtrees of candidate sentences.** Often the answer to a posed question is contained within a sentence fragment. For example, in Figure 2-6 the question is really being answered by the sentence fragment "Gennady Lyachin, the Kursk commander". There is no need to penalize the alignment because other fragments in the candidate, such as "the Hero of Russia", were not aligned with any nodes in the template.[4] On the other hand, not

[4]This is a simplification. Certain terms outside a fragment can change its meaning (e.g., conditional expressions, modal expressions, negations, etc.). In practice the issues discussed in Section 5.2 have a greater impact on performance. The framework presented here is still sound—a modification of Function 2.10 on page 33 could handle these cases by penalizing the appropriate constructions.

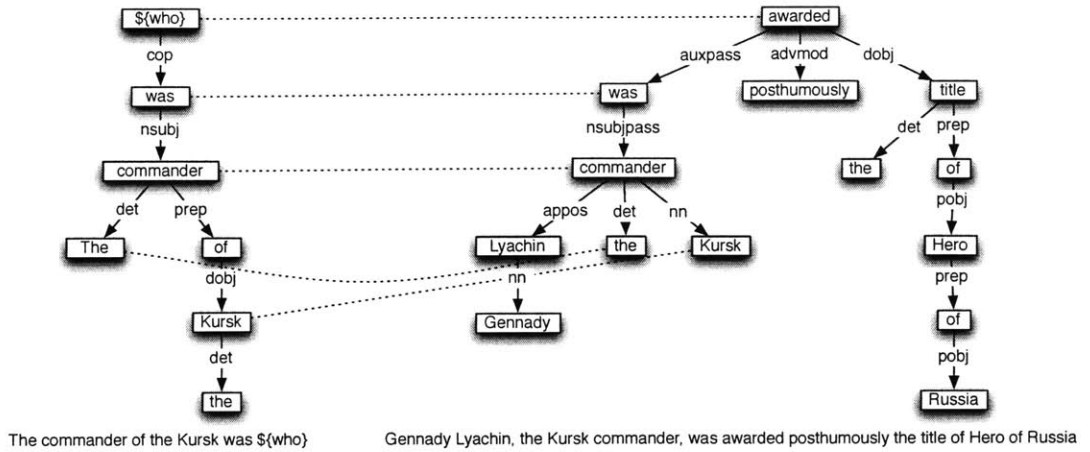


Figure 2-7: A suboptimal alignment that results from ignoring the type of the syntactic dependencies in the tree. If dependency types are ignored, an algorithm will consider this alignment preferable since "was" is now aligned.

aligning nodes from the template should heavily penalize alignments (consider alignments where "Kursk" or "commander" are not paired at all).

- **Trees are unrooted.** To be able to obtain the alignment in Figure 2-6, we have to view the dependency trees as unrooted trees. An alignment between rooted trees will necessarily maintain the same ancestor–descendant dominance in both trees. The node alignments `${who}` $\Longleftrightarrow$ `Lyachin` and `commander` $\Longleftrightarrow$ `commander` show that this dominance is violated in Figure 2-6.

- **Trees are unordered.** The alignment in Figure 2-6 also suggests that dependency trees should be viewed as unordered. This might be easier to see by considering that "Mother Teresa died on September 5, 1997, in India." and "Mother Teresa died in India on September 5, 1997." are two sentences with different orderings that have the same meaning.

  Unfortunately, while there have been a number of increasingly efficient algorithms for determining the edit-distance between ordered trees [22] [49] [9], the unordered case has proven especially difficult to compute. Zhang et al. showed

that the problem is NP-complete [47] [50]. Later Zhang and Jiang showed that the problem is MAX-SNP[5] hard [48], so unless P=NP there is not even a polynomial time approximation scheme (PTAS) [1].

Fortunately, we will see that it is possible to design an algorithm that, at least for question answering applications, runs fast enough in practice.

2.3.3 Aligning Unordered Trees

Most algorithms for computing the edit-distance between trees are based on a dynamic program that minimizes the transformation cost necessary to make the trees identical. It is fairly straightforward to write a similar dynamic program for unrooted, unordered, labeled trees. Below I present such a program but describe it as maximizing a similarity score between two trees instead of minimizing their transformation cost. This is done only for expository purposes; the two formulations are equivalent to each other.

The Dynamic Programming Algorithm

Let G be a graph, we denote by $\mathrm{V}(G)$ the set of vertices in the graph, and by $\mathrm{E}(G)$ the set of edges in the graph. Given a vertex $v \in \mathrm{V}(G)$, we denote by $\mathrm{E}(v)$ the set of outgoing edges of vertex v. We use the symbol $\emptyset$ to denote the empty graph. Recall that our dynamic program admits two operations: node alignment and edge contraction. The score obtained from aligning a node v with a node v' is denoted by $\textsc{AlignScore}(v, v')$. The cost of contracting an edge e is denoted by $\textsc{ContractCost}(e)$. The numeric score/cost of these two operations is purposefully not specified at the moment. The algorithm does not depend on their particular values and only assumes that they are greater than or equal to zero; several possibilities can in fact be tried depending on the desired results. Chapter 4 describes the values actually used in the implementation of SmartQA.

[5]MAX-SNP stands for MAXimization–Strict NP. It is the complexity class containing problems with constant-factor approximation algorithms but no polynomial time approximation scheme unless P=NP.

The base case for the dynamic programming algorithm is given by Functions 2.9, 2.10 and 2.11. As discussed on page 30, Function 2.9 penalizes for failing to align nodes in the template, but Function 2.10 does not penalize for failing to align nodes in the candidate.

$$\textsc{Score}(G, \emptyset) = -\sum_{e \in \mathrm{E}(G)} \textsc{ContractCost}(e) \tag{2.9}$$

$$\textsc{Score}(\emptyset, G') = 0 \tag{2.10}$$

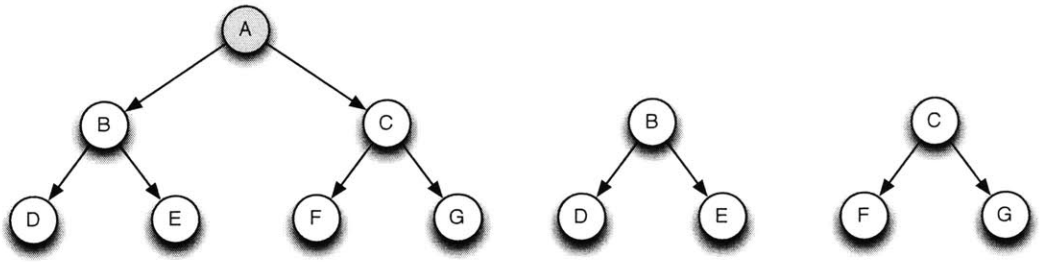
$$\textsc{Score}(\emptyset, \emptyset) = 0 \tag{2.11}$$

The rest of the dynamic programming algorithm is specified by defining the score functions for three different cases: unrooted trees, rooted trees, and unordered forests of rooted trees.

Let T be an unrooted tree. We denote by T_v the rooted tree obtained by selecting the vertex v in $\mathrm{V}(T)$ as the root of the tree. Given two unrooted, unordered trees T and T' we can calculate their similarity score according to Function 2.12.

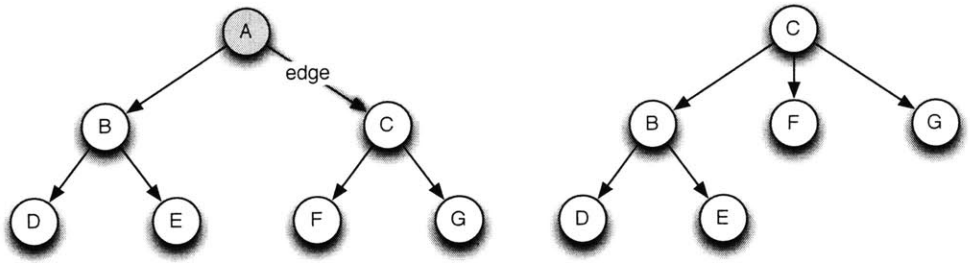
$$\textsc{Score}(T, T') = \max \begin{cases} \textsc{Score}(T_{v_1}, T'_{v'_1}) \\ \quad \vdots \qquad \text{for each } v' \text{ in } \mathrm{V}(T') \\ \textsc{Score}(T_{v_1}, T'_{v'_m}) \end{cases} \tag{2.12}$$

Let T_v be a rooted tree. We denote by $T_v - v$ the forest obtained by removing the root vertex v from the tree. Figure 2-8 illustrates the resulting forest. Similarly, we denote by $T_v - e$ the tree obtained by contracting the edge e in $\mathrm{E}(T_v)$. Figure 2-9 illustrates the resulting tree. Given two unordered, rooted trees T_v and $T'_{v'}$ we can calculate their similarity score according to Function 2.13. The case for scoring forests, $\textsc{Score}(F, F')$, is defined later in Function 2.14.



(a) Original (b) Result

Figure 2-8: After the roots of the template and the candidates are aligned, the algorithm removes the aligned root (denoted in gray) and then recurses on the remaining forest.



(a) Original (b) Result

Figure 2-9: Tree obtained after contracting an edge. The edge being contracted is the only one labeled in the original tree.

$$\textsc{Score}(T_v, T'_{v'}) = \max \begin{cases} \textsc{Score}(T_v - v, T'_{v'} - v') + \textsc{AlignScore}(v, v') \\ \textsc{Score}(T_v - e_1, T'_{v'}) - \textsc{ContractCost}(e_1) \\ \quad \vdots \quad \text{for each } e \text{ in } \mathrm{E}(v) \\ \textsc{Score}(T_v - e_n, T'_{v'}) - \textsc{ContractCost}(e_n) \\ \textsc{Score}(T_v, T'_{v'} - e'_1) - \textsc{ContractCost}(e'_1) \\ \quad \vdots \quad \text{for each } e' \text{ in } \mathrm{E}(v') \\ \textsc{Score}(T_v, T'_{v'} - e'_m) - \textsc{ContractCost}(e'_m) \end{cases} \tag{2.13}$$

Let F be a forest. We denote by $F - T_v$ the forest resulting from removing the rooted tree T_v from F. Given two unordered forests of rooted, unordered trees we can calculate their similarity score according to Function 2.14. Note that most of the running time complexity of the given algorithm comes from this definition. Because the forests are assumed to be unordered, we essentially have to try all possible tree permutations in order to determine the optimal one.

$$\textsc{Score}(F, F') = \max \begin{cases} \textsc{Score}(T_{v_1}, \emptyset) + \textsc{Score}(F - T_{v_1}, F') \\ \textsc{Score}(T_{v_1}, T'_{v'_1}) + \textsc{Score}(F - T_{v_1}, F' - T'_{v'_1}) \\ \quad \vdots \quad \text{for each } T'_{v'} \text{ in } F' \\ \textsc{Score}(T_{v_1}, T'_{v'_m}) + \textsc{Score}(F - T_{v_1}, F' - T'_{v'_m}) \end{cases} \tag{2.14}$$

Speeding up Computation

The algorithm, as described so far, runs in an unacceptable exponential time with respect to the number of nodes in the input trees. In practice, two key observations allow us to cut the running time significantly.

1. **Upper and lower bounds for each branch of the computation can be calculated efficiently.** Recall that node alignments increase the score of an

alignment, and that edge contractions decrease its score. It is easy to see, then, that given graphs G and G' the score of their optimal alignment can't be more than the bound given by Function 2.15. Note that the function can be computed efficiently; it has a polynomial running time of $O(|V(G)||V(G')|)$.

$$\textsc{UpperBound}(G, G') = \sum_{v \in V(G)} \max \begin{cases} \textsc{AlignScore}(v, v'_1) \\ \quad \vdots \qquad \text{for each } v' \text{ in } V(G') \\ \textsc{AlignScore}(v, v'_m) \end{cases} \tag{2.15}$$

Similarly, given graphs G and G' the score of their worst alignment can't be less than the bound given by Function 2.16. Note that the function can be computed efficiently; it has a polynomial running time of $O(|E(G)|)$.

$$\textsc{LowerBound}(G, G') = \textsc{Score}(G, \emptyset) \tag{2.16}$$

2. **Most possible alignments are far from the optimum and don't need to be computed.** A naïve implementation of the algorithm would try all possible operations and corresponding alignments and then return the optimal one. A better implementation would use upper and lower bounds for each computational branch to determine which possibilities should be tried at all. A significant amount of branches can be cut in practice since almost all possible alignments are far from the optimal one.

   For example, take the template and candidate from Figure 2-6. Each word in the template has at most 3 or 4 possible alignments that would contribute anything to the total score. For example, an algorithm should try all alignments that pair any of the occurrences of `the` in the template with any of the occurrences of `the` in the candidate, but it shouldn't waste much time trying to pair `the` with, say, `Russia`. In fact, even if some individual words can be aligned, there are certain tree alignments that we would like to avoid calculating all together. Take for

example any alignment between the template "the commander of the Russian submarine Kursk is ${who}" and the fragment "was awarded posthumously the title of Hero of Russia". It is clear that we won't find any useful alignments within that fragment.

The notion of *branch cutting* can be formalized as follows. Given two computational branches $\textsc{Score}(G, G')$ and $\textsc{Score}(H, H')$ there is no need to compute $\textsc{Score}(G, G')$ if $\textsc{UpperBound}(G, G') < \textsc{LowerBound}(H, H')$. Under these conditions $\textsc{Score}(H, H')$ will always yield a score closer to the optimum.

Chapter 3

Related Work

The idea of relying on syntax for question answering is not new. However, when compared to keyword-based approaches, the use of syntactic relations for question answering has been much more sparse, possibly because of the computational costs of generating parse trees and comparing them in a principled manner.

3.1 START

START is an information access system developed by Katz over the past two decades. START stands for Syntactic Analysis using Reversible Transformations and was the first question answering system to use a dependency-based representation [15] [19] [18]. START heavily influenced the work in this thesis, so it is not surprising that its approach to question answering is somewhat similar.

In its most fundamental mode of operation, START matches English questions to English assertions that have been previously submitted to the system. START parses all questions and assertions it receives and transforms them into sets of *ternary expressions*[1], which are then matched—questions to potential answers—using sentence-level linguistic processing techniques that account for such phenomena as synonymy,

[1] A version of dependency triples that permits embedding (other triples can serve as heads or dependents). Ternary expressions can capture exactly the same structural information that dependency trees can.

hyponymy, use of passive voice and verb alternations[2] [17]. To illustrate one of these techniques: START uses a rule-based engine, with knowledge about property-factoring alternations, to infer that "Miriam amused Jessica with her performance." and "Miriam's performance amused Jessica." are two sentences with essentially the same semantic content.

To provide access to free text, structured databases, semi-structured data, images and other types of multimedia, START matches questions to natural language *annotations*—content-describing English phrases and sentences composed by human annotators—that have been associated with various retrievable information segments [16]. Natural language annotations can be parameterized, so that in many cases, a few, simple annotations can describe entire information resources for purposes of information access. START currently provides access to information in approximately two hundred sources, and its use of natural language annotations and matching at the level of ternary expressions has been extensively proof-tested, with START having answered several million questions submitted by users of the World Wide Web since START's appearance on the Web in 1993[3].

This thesis bypasses the human annotation process that enables START to provide access to information. It demonstrates that useful answers can often be obtained by matching syntactically analyzed questions directly to syntactically processed statements within documents. In addition, the approach taken in this thesis allows for partial matches, which increases the robustness of the matching process for application to large corpora.

3.2 Surface Pattern Approaches

Many of the early systems that exploited syntactic relationships did so through the use of textual patterns.

In 1993, Kupiec developed MURAX [23], a question answering system that used

[2]For a detailed study of verb alternations, see Levin's work [25].

[3]START is available at `http://start.csail.mit.edu`

Grolier's online encyclopedia as its knowledge base. MURAX used a two-staged approach to question answering. Stage one consisted of finding relevant documents using a boolean, proximity-based IR engine, along with a set of heuristics for broadening or narrowing the search. Stage two used a series of regular expressions to analyze the articles and select answers. The regular expressions captured certain syntactic relations such as: apposition, conjunctions, and copular verbs.

In an informal evaluation done on 70 Trivial Pursuit questions[4], Kupiec reported an accuracy of 53%, with 74% of the answers found in the top 5 returned results.

Similar results were later obtained by pattern-based systems participating in TREC. Soubbotin [38] developed a system for TREC-10 that selected answers based on a predefined series of surface patterns. The patterns were constructed from simple units, such as letters and punctuation marks, but were used to capture similar syntactic relations to those in MURAX in addition to a few more domain-specific ones (for example, there were specific patterns for finding birth dates in encyclopedia style contexts: "John Lennon (October 9, 1940 – December 8, 1980)"). At the time, the system was the leading participant in TREC, obtaining a mean reciprocal rank (MRR) [45] of 0.676.[5]

Despite their early promise, surface patterns are not able to capture the full range of syntactic relations. They are only capable of capturing syntactic relations that occur between elements that appear fairly close to each other in the surface text, and are unable of recognizing relations between elements that are farther away, such as a series of prepositional phrases all modifying the same head (only one of the prepositional phrases would appear next to the modified head in the surface text).

3.3 Parsing-Based Approaches

Approaches that make use of syntactic relations obtained through parsing have also been tried.

[4]The TREC Question Answering track was not introduced until TREC-8 in 1999.

[5]Note that TREC has gotten progressively harder over the years, and its evaluation methodology has changed. It is therefore not straightforward to compare results obtained in different years.

For TREC-8, Litkowski developed a question answering system based on semantic triples [28]. The system worked by parsing each sentence of the knowledge base documents, and using the resulting parse trees to generate selected relational triples that were stored in a database. Upon receiving a question, the system would obtain a similar set of semantic triples from the question, this time with some unbound variable, and use them to query the database for possible answers. While semantic triples were supposed to capture semantic roles as generally understood in linguistics (roles such as agent, theme, location, etc.), in practice semantic analysis is still an open problem, so the system settled for a few syntactic relations including subject–object relations, adjectival modifications, and prepositional phrase attachments. The system received an MRR score of 0.281, which was slightly below the average 0.331 for that year.

While early results were less than ideal, recently there has been a resurging interest in parsing-based, syntactically-motivated approaches to question answering.

Attardi et al. developed the PIQASSO question answering system in 2001 [2], that relied heavily on the Minipar parser by Lin [26] to perform syntactic analysis. Before answer selection, a series of rules normalized Minipar's output to overcome some cases of inaccurate parsing and to select only the dependencies Attardi et al. identified as the most relevant for question answering. Candidate answers were then filtered according to the expected answer type (as determined by the type of question that was asked). In this case the type was one of person, organization, time, quantity and location. Finally, candidates were scored based on how many dependency triples overlapped with the question. Triples were assigned different weights depending on their deemed importance—the rest of the tree structure was ignored. PIQASSO received an MRR score of 0.271 and was ranked 15th out of 36 participants in that year's TREC.

For TREC-11, Litkowski developed a system that relied heavily on discourse analysis and tree-structure patterns to answer questions [30]. Sentences in the knowledge base were parsed using his PROXIMITY parser [29]. The resulting parse trees were then run through a discourse analysis module that annotates the tree with the results

of several linguistic submodules. Examples of submodules include anaphora resolution, named entity tagging, and segmentation of sentences into smaller discourse units. Finally, a series of predetermined XPath patterns[6], that differ based on the type of question, were used to select answers. For factoid questions[7], the system had an accuracy of 0.070.

Echihabi and Marcu proposed a noisy channel approach to question answering [10]. Candidate sentences were scored according to the probability that they generated the original question. The probabilities were calculated according to a generative story in which the candidate parse tree is transformed into the question tree through a series of sequential transformations, including: deleting complete subtrees, deleting nodes, duplicating nodes, replacing candidate nodes by nodes in the question, and permuting the order of nodes. To determine the probability of each operation, they trained their model using approximately 2000 question/answer pairs extracted from the web. They reported an MRR score of 0.354 when tested against the TREC 2002 questions.

Punyakanok et al. [36] take an approach similar to this thesis, in that the similarity measure they used was the tree edit-distance between questions and candidates. In their case however, the edit-distance used was for ordered trees as opposed to unordered trees. The system also performed candidate filtering using a named entity recognizer to verify expected answer types. They performed an evaluation on the TREC 2002 questions, but checked only if their system returned known supporting documents and not whether it also extracted the correct answer. Their reported accuracy was 36.6%.

Mollá and Gardiner developed the ANSWERFINDER system for TREC 2004 [33].

[6]For those unfamiliar with XML technologies, an XPath expression is a sequence of patterns, similar in spirit to a regular expression, that selects nodes from XML trees based on their labels and attributes. See the XPath Technical Specification [43] for more information.

[7]For the first time in TREC, TREC-11 predivided questions according to 3 categories: factoid questions that have a single, correct, factual answer ("Who killed John F. Kennedy?"), list questions that have several correct answers ("Name countries that produce coffee"), and definition questions that are answered with information nuggets that "define" the topic ("Tell me about John Lennon"). The evaluation methodology was also changed—reported scores for factoid questions were no longer based on MRR but rather on accuracy (precision at 1).

Sentences were parsed using the CONNEXOR dependency parser [41] and then converted into logical forms[8]. The system used answer type filtering along with a weighed average of three similarity measures: a lexical one that measured the number of overlapping words between the question and the candidate, a syntactic one that measured the number of overlapping dependency triples, and a semantic one that measured the number of overlapping terms in the logical forms. For factoid questions, ANSWERFINDER obtained an accuracy of 10%.

Sun et al. [39] developed a question answering system for TREC 2005 that used the ASSERT shallow semantic parser [35]. Their similarity measure focused heavily on verbs. The similarity score between two verb-rooted trees was a weighed average between (i) a WordNet[9]-based score that captured the semantic closeness between the two verbs, and (ii) a keyword overlap score between the arguments of the verb (stopwords were removed and repeated words were counted only once). The similarity between a question and a candidate was computed by choosing the optimal alignment between all verb-rooted subtrees. The system reported an accuracy of 0.666 for factoid questions and was the 2nd highest ranked system in 2005. This system also had the best IR module in TREC 2005—it is unclear how much of the performance is due to the system's excellent IR and how much is due to its matching algorithm.

In fact, for all of these systems it is hard to determine exactly how beneficial their different syntactic approaches have been. Evaluations are generally done only on the end-to-end systems with no detailed study of the contributions that each module makes. From a thorough reading of all the different system descriptions and a careful study of many of the answers that these systems reportedly miss, it seems that the biggest problems that have plagued previous work are: the lack of a mechanism for coping with phrasing variability, and the use of *ad hoc* methods that ignore most of the structure of syntactic trees. It is precisely these two problems that SMARTQA attempts to tackle with its alignment algorithm.

[8] A Prolog-like expression that attempts to capture the semantic content of a sentence.

[9] A lexical database for the English language developed by the Cognitive Science Laboratory at Princeton University [11].

Chapter 4

Implementation

4.1 Knowledge Representation

As discussed in Section 2.2, we would like our representation to make overt the predicate–argument structure present between between a head of a constituent and any of its adjuncts. This requirement naturally leads to a dependency-tree representation. The question still remains, however, of which dependency formalism should be used.

For the proof-of-concept implementation of SMARTQA described in this thesis, the choice was pragmatically motivated. I chose the formalism of the Stanford natural language parser [21], whose dependencies are based on those by Carroll et al. [5] and King et al. [20], because the parser had the following features:

- **Open Source**. The parser binaries and source code are made freely available under the GPL license. Access to the code was particularly useful; it allowed me to modify small portions of the parser code to better integrate it with the rest of the system.

- **Wide Coverage**. The parser is based on a probabilistic context-free grammar (PCFG) that was trained on the Penn Treebank corpus. Klein and Manning, the parser authors, report a performance of 86.36% (LP/LR F_1) [21].

- **Phrase and Dependency Representation Support**. There are a number of

parsers based on dependency grammars, such as Minipar by Lin [26] or the Link Parser by Sleator and Temperley [37], but they are less robust and accurate than phrase parsers trained on large corpora. There are also a number of treebank-trained statistical parsers, such as Charniak's [6] or Collins' [7], but they don't provide a dependency-based representation. Stanford's parser is the only parser I'm aware of that provides the best of both worlds: it is a treebank-trained phrase parser that in addition uses a series of patterns to provide syntactic dependencies [8].

4.2 Architecture

SMARTQA follows the general question answering architecture outlined in the question answering survey by Hirschman and Gaizauskas [14]. In particular, it is composed of four main modules: question analysis, document selection, document analysis and answer extraction. Figure 4-1 illustrates how the modules interconnect.

4.2.1 Question Analysis

The question analysis module is responsible for turning the question into an assertive template and generating the corresponding dependency tree. The Stanford parser performs poorly on questions—statistical parsers are generally not trained to parse questions—so this transformation is performed by the START system.

Once the question has been turned into a template, the Stanford parser is used to create the corresponding dependency tree. Because templates contain variables, the Stanford parser was slightly modified to parse all variables as nouns.

4.2.2 Document Selection

The document selection module is responsible for returning a set of relevant documents in which to look for answers. SMARTQA uses the open-source Lucene engine [42] as its IR engine. Queries are generated by removing all variables from the

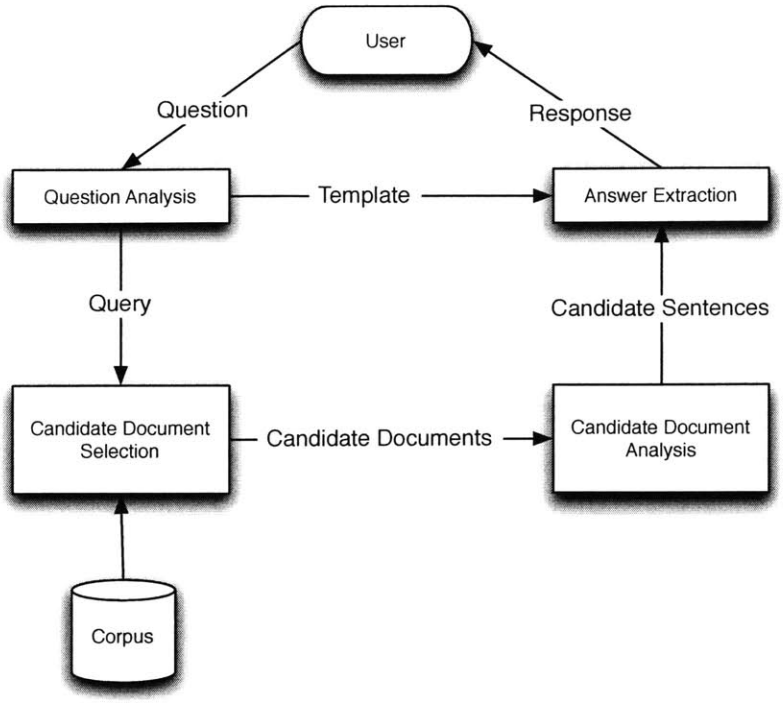


Figure 4-1: Architecture of SMARTQA. Modules are represented as rectangular boxes. Communication between modules is denoted by arrows with labels indicating the type of information that is transmitted.

analyzed question and performing a boolean OR query of all the remaining keywords. Note that Lucene automatically removes stopwords and weights query terms according to a variant of $tf \cdot idf$.

Ideally we would like to look at every document returned by Lucene, but because that is too computationally expensive, SMARTQA restricts its analysis to the top 50 documents. The implementation allows this parameter to be changed if one is willing to expend more computational time looking at additional documents.

4.2.3 Document Analysis

Once a set of relevant documents has been selected, the document analysis module is responsible for selecting candidate sentences and converting them into an appropriate

representation.

In the simple approach taken by SMARTQA this module simply breaks documents into sentences using a rule-based chunker, and then parses the sentences using the Stanford parser. In future versions of SMARTQA this would be the ideal module to perform other linguistic analyses such as named entity recognition and anaphora resolution.

4.2.4 Answer Extraction

Answers are selected using the tree alignment algorithm specified in Section 2.3.3. Because the algorithm uses the branch-cutting method defined on page 36 to speed computation, a custom interpreter was written to evaluate it. The implementation maintains a queue containing the computational branches that still need to be evaluated. During each iteration the interpreter: (i) grabs the first expression from the queue, (ii) partially evaluates it and adds any resulting subexpressions to the queue, (iii) computes new upper and lower bounds for the partially evaluated expression and any of the new subexpressions, and (iv) performs branch cutting. The iteration continues until the optimal answer is found.

Recall that Section 2.3.3 left open the definitions of both CONTRACTCOST(e) and ALIGNSCORE(v, v'). For this first version of SMARTQA, CONTRACTCOST(e) is implemented according to Definition 4.1. Note that it always returns a score of 0. This doesn't mean, however, that there is no penalty for contracting edges. There is a penalty, but because the penalty depends on the next aligned node, it was easier to compute inside the next call to ALIGNSCORE(v, v'). The implementation of CONTRACTCOST(e) therefore stores the contracted edge in a list L so that it can be referenced later.

$$\text{CONTRACTCOST}(e) = 0 \qquad \text{and } e \text{ is stored in } L \tag{4.1}$$

Denote by path(v) the path from node v to the last previously aligned node (this path is computed by selecting the appropriate edges from L). Let $I_{\text{stem}}(v, v')$ be an

indicator variable that is 1 if both v and v' have the same stem (according to an implementation of the Porter stemmer [34]), and 0 otherwise. Then $\textsc{AlignScore}(v, v')$ is implemented according to Definition 4.2, where α is a damping factor that was set to 0.8.

$$\textsc{AlignScore}(v, v') = I_{\text{stem}}(v, v') \cdot idf(v) \cdot \alpha^{\sum_{u' \in \text{path}(v')} idf(u')} \tag{4.2}$$

Intuitively the definition assigns a score based on the *idf* of the aligned nodes, and then penalizes for any edge contractions performed since the last alignment.

Chapter 5

Experimental Results

5.1 Results

To evaluate the impact that syntactic information has on question answering, the following systems were run against the 362 factoid questions from TREC 2005:

- SENTENCE. A dummy system based on the same architecture as SMARTQA that simply returns every sentence from every document selected by the IR engine. Sentences are returned in the same order as they appear in the document.

- KEYWORD OVERLAP. A base-line system based on the same architecture as SMARTQA that returns sentences based on how many keywords overlap with the questions. In particular, each candidate sentence S receives a score according to the following formula:

$$\sum_{\text{word } w \in S} idf(w) \cdot I_Q(w)$$

  where $I_Q(w)$ is an indicator variable that is 1 if the word is also present in the question, and 0 otherwise.

  Sentences are sorted according to the assigned score and returned in that order.

- ARANEA. A keyword-based question answering system that uses a set of filters and heuristics to ensure the validity of its answers [27]. ARANEA was used as

the basis of MIT's entry in the 2005 TREC QA Track and obtained an accuracy of 0.273.

Traditionally ARANEA first finds answers on the web and then tries to project those answers into the relevant corpus. For purposes of this evaluation, however, ARANEA was modified to rely only on the TREC corpus. While it is undeniable that the web is an excellent source of knowledge, this evaluation is interested in measuring the usefulness of syntactic approaches to question answering. Using the web as a resource would hinder this comparison. The sheer size of the web often voids the need to handle phrasing variability—on the web, there's almost always a document with the same phrasing as the posed question.

- SEMANTIC OVERLAP. The passage retrieval algorithm introduced by Marton in [31] that was the top-scoring, fully-automated system for the Relationship Task in the TREC 2005 Question Answering Track.

  The algorithm scores sentences using variants of two standard IR measures: precision and recall. Given a similarity measure $s(q, w)$ between two words, the implemented version of the SEMANTIC OVERLAP algorithm computes the similarity between a candidate sentence S and a given question Q, according to following equations:

$$Q^* = \{q \in Q \mid s(q, w) > 0 \text{ for some } w \in S\} \tag{5.1}$$

$$S^* = \{w \in S \mid s(q, w) > 0 \text{ for some } q \in Q\} \tag{5.2}$$

$$\text{recall}(Q, S) = \sum_{q \in Q^*} idf(q) \tag{5.3}$$

$$\text{precision}(Q, S) = \frac{\sum_{q \in Q^*}(1 - \prod_{w \in S^*} 1 - s(q, w))}{|Q^*|} \tag{5.4}$$

$$\text{similarity}(Q, S) = \frac{3 \cdot \text{recall}(Q, S) \cdot \text{precision}(Q, S)}{2 \cdot \text{recall}(Q, S) + \text{precision}(Q, S)} \tag{5.5}$$

As suggested by Marton in a private conversation, $s(q, w)$ was set to 1 for words that have the same stem, and for words that are synonyms according to WordNet. For all other cases it was set to 0.

- SMARTQA. The system described in this thesis.

Each system returned a set of ranked sentences as its answers[1], and these were evaluated using the official answer patterns from TREC 2005. The measured accuracy for each system is shown in Table 5.1.

System	Accuracy
SENTENCE	0.035
KEYWORD OVERLAP	0.150
ARANEA	0.162
SEMANTIC OVERLAP	0.192
SMARTQA	0.253

Table 5.1: Measured accuracies for each of the systems evaluated against the TREC 2005 questions.

If we allow each system to return more than one answer per question, it is possible to measure the percentage of questions that have a correct answer within the top k sentences. Figure 5-1 plots this percentage for values of k between 1 and 10.

5.2 Discussion

The obtained results are promising. SMARTQA's accuracy outperformed all of the keyword-based systems it was compared against.

Examination of the results show that there were 84 questions for which the IR system never returned a sentence containing a valid answer, so SMARTQA never had a chance of answering those. If we evaluate the accuracy of the system based only on the remaining 278 questions, SMARTQA obtains an accuracy of 0.329. This is

[1]This is not exactly accurate. ARANEA traditionally returns answer nuggets and has no concept of sentences. For purposes of this evaluation, ARANEA was modified to return a window of 90 characters around a selected nugget.

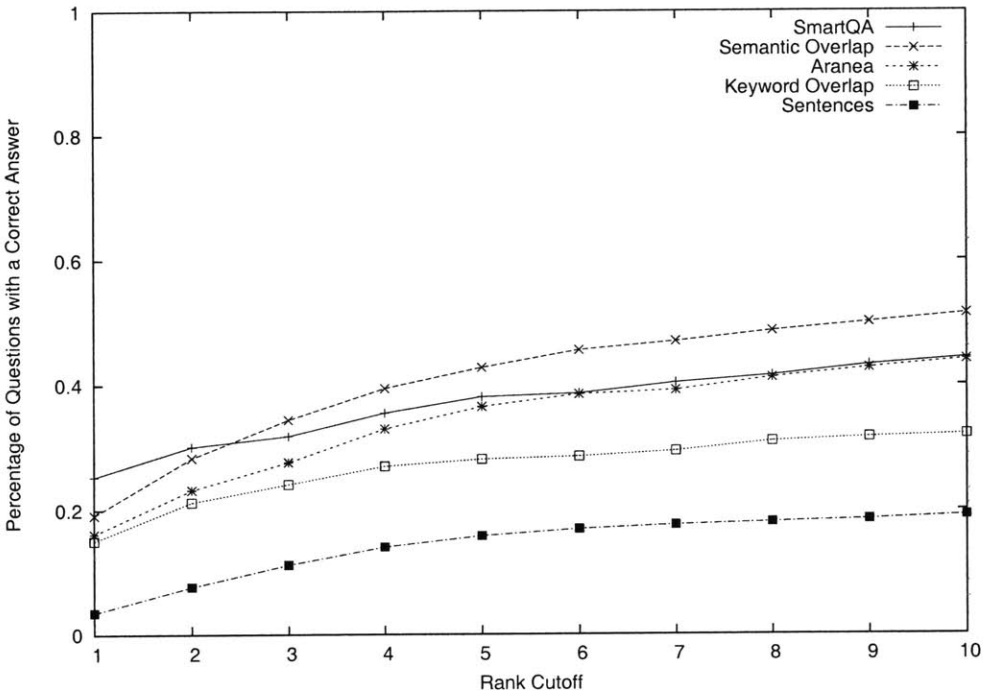


Figure 5-1: Performance of the evaluated systems as a function of cutoff rank.

as expected; in a pipeline architecture the performance of the IR module can have a significant impact on the end-to-end performance of the system.

For comparison purposes Table 5.2 shows the accuracies of the top 10 ranked systems for TREC 2005. Similarly, Table 5.3 shows the performance of the top 13 IR modules, with the position that SMARTQA's would occupy.

System	Accuracy
Language Computer Corp	0.713
National University of Singapore	0.666
IBM TJ Watson Research	0.326
University of Albany	0.309
Harbin Institute of Technology	0.293
MIT	0.273
Fudan University	0.260
National Security Agency	0.257
Saarland University	0.235
University of Edinburgh	0.215

Table 5.2: Reported accuracies for the top 10 systems in TREC 2005.

System	R-Precision
National University of Singapore	0.4570
Hummingbird	0.4127
IBM T.J. Watson Research	0.3978
National Security Agency (NSA)	0.3414
Johns Hopkins University—Applied Physics Lab	0.3201
Arizona State University	0.2958
University of North Texas	0.3205
Sabir Research	0.3366
Language Computer Corporation	0.2921
Macquarie University	0.3038
Peking University	0.2732
MIT	0.2699
SMARTQA	**0.2667**
University of Albany	0.2445

Table 5.3: Reported R-Precision for the top 13 IR modules in TREC 2005.

It is quite impressive that SMARTQA is able to achieve such an accuracy relying purely on syntax and looking at only 50 documents. Detailed inspection of the results

shows that, if coupled with many of the additional modules full-fledged question answering systems normally have, SMARTQA has the potential of outperforming most of the top 10 systems from TREC 2005. Specifically, the incorrect answers returned by the system suggest that SMARTQA would particularly benefit from the adding the following features:

- **Improved Node Alignment**

  A fundamental assumption in SMARTQA's alignment algorithm is that it is able to align the appropriate terms between a template and a candidate answer. In the current implementation, alignments are permitted between words that have the same stem. Clearly, this not enough to handle all cases. Consider the question "Which countries expressed regret about the loss of the Russian submarine Kursk?" and the sentence

  (5.6) Czech President Vaclav Havel has expressed grief and sorrow over the victims of the disaster aboard the Russian nuclear submarine Kursk.

  Ideally, it should be possible to align "regret" from the question with "grief" or "sorrow" in 5.6. A possibility might be to allow matches between related terms according to WordNet or some other lexical resource.

- **Question Typing**

  In some cases, syntax alone is not enough to discriminate between a correct and an incorrect answer. Consider for example the question "When did the Russian submarine Kursk sink?". The following two sentences would obtain a perfect score when only their syntax is compared with the template:

  (5.7) The Russian submarine Kursk sank in the Barents Sea.

  (5.8) The Russian submarine Kursk sank in August, 2000.

  This problem can be solved by filtering candidate sentences according to the expected answer type. In this case the posed question was a "when" question, and the expected answer type is therefore a date. A named-entity recognizer

should be able to identify that in Example 5.7 the returned answer contains a location, while in Example 5.8 the returned answer contains a date. The system would therefore discard 5.7 and choose 5.8.

- **Coreference Resolution**

  Sometimes SMARTQA is unable to identify a sentence as a correct answer because key entities are not mentioned explicitly but rather referred to through pronominal and definite NP anaphora.

  Consider for example the sentence

  (5.9) If the submarine were left at the bottom of the Barents Sea, where it sank on August 12, 2000, the public would remain alarmed.

  It correctly answers the question "When did the Kursk sink?", but SMARTQA assigns it a poor score because it is unable to tell that "it" refers to the Kursk.

  A coreference resolution module would allow SMARTQA to handle such cases.

- **Integrating Knowledge from Multiple Sources**

  At the moment, SMARTQA uses sentences as the basic unit of knowledge. In some cases however, information necessary to answer a question is spread across two or more sentences.

  Consider the question "Who is the President of the largest country in Asia?" and the two sentences

  (5.10) China is the largest country in Asia with a total area of 9,596,960 square meters.

  (5.11) The President of China, Hu Jintao, was elected to office the 15th of March, 2003.

  Taken individually, SMARTQA finds both sentences unsatisfactory. Taken together, they correctly answer the posed question.

Notice that a solution to this problem is closely related to coreference resolution. In order to combine both sources of knowledge, a system must first determine that "China" refers to the same entity in both sentences.

- **Off-line Preprocessing**

  Despite the branch-cutting technique that SMARTQA uses to make unordered tree matching feasible, running the system over a large corpus is still too computationally expensive. To lessen the time before an answer is returned, SMARTQA currently processes at most 50 documents per question, and sometimes misses important information that is contained outside those 50 documents.

  To further speed up the response time, it should be possible to perform certain computation beforehand. In particular, one could preprocess the corpus by tokenizing and parsing every sentence ahead of time. Increased speed would enable SMARTQA to consider more documents in the same amount of time, allowing it to find relevant answers outside the top 50 documents.

Chapter 6

Conclusion

The main contribution of this thesis was the design and implementation of SMARTQA, a proof-of-concept question answering system that compares syntactic trees in a principled way. In particular, this thesis provides:

- A clear formulation of the dynamic programming algorithm necessary for comparing unrooted, unordered syntactic trees.
- A formalization of a branch-cutting technique useful for speeding up computation of a dynamic program while still guaranteeing optimality.
- An implementation of the above-mentioned algorithm, fully integrated into an end-to-end question answering system.
- Experimental results that demonstrate the viability of syntax-based approaches to question answering.

Bibliography

[1] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[2] Giuseppe Attardi, Antonio Cisternino, Francesco Formica, Maria Simi, and Alessandro Tommasi. PiQASso: Pisa question answering system. In *Text REtrieval Conference*, 2001.

[3] Philip Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217–239, 2005.

[4] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[5] John Carroll, Guido Minnen, and Ted Briscoe. Corpus annotation for parser evaluation. In *EACL '99 Workshop on Linguistically Interpreted Corpora (LINC-99)*, Bergen, Norway, July 1999.

[6] Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the First Conference on North American Chapter of the Association for Computational Linguistics*, pages 132–139, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[7] Michael John Collins. *Head-driven statistical models for natural language parsing.* PhD thesis, University of Pennsylvania, 1999.

[8] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure parses. In *LREC*, 2006.

[9] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An $O(n^3)$-time algorithm for tree edit distance. `http://arxiv.org/abs/cs/0604037`, 2006.

[10] Abdessamad Echihabi and Daniel Marcu. A noisy-channel approach to question answering. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 16–23, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[11] Christiane Fellbaum. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press, May 1998.

[12] Liliane Haegeman and Jacqueline Guéron. *English Grammar: A Generative Perspective*. Blackwell Textbooks in Linguistics. Blackwell, Malden, Massachusetts, first edition, 1999.

[13] David G. Hays. Dependency theory: A formalism and some observations. *Language*, 40(4):511–525, October 1964.

[14] Lynette Hirschman and Robert Gaizauskas. Natural language question answering: the view from here. *Natural Language Engineering*, 7(4):275–300, 2001.

[15] Boris Katz. Using English for indexing and retrieving. Technical Report AIM-1096, MIT Artificial Intelligence Laboratory, Cambridge, MA, USA, 1988.

[16] Boris Katz. Annotating the world wide web using natural language. In *Proceedings of RIAO '97*, Montreal, Canada, 1997.

[17] Boris Katz and Beth Levin. Exploiting lexical regularities in designing natural language systems. In *Proceedings of the 12th Conference on Computational Linguistics (COLING '88)*, pages 316–323, Budapest, Hungary, 1988. Association for Computational Linguistics.

[18] Boris Katz and Patrick H. Winston. Parsing and generating English using commutative transformations. Technical Report 677, MIT Artificial Intelligence Laboratory, Cambridge, MA, USA, 1982.

[19] Boris Katz and Patrick H. Winston. A two-way natural language interface. In *Proceedings of the European Conference on Integrated Interactive Computing Systems (ECICS '82)*, 1982.

[20] Tracy King, Richard Crouch, Stephen Riezler, Mary Dalrymple, and Ronald Kaplan. The PARC700 dependency bank. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)*, 2003.

[21] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 423–430, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[22] Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *ESA '98: Proceedings of the 6th Annual European Symposium on Algorithms*, pages 91–102, London, UK, 1998. Springer-Verlag.

[23] Julian Kupiec. MURAX: A robust linguistic approach for question answering using an on-line encyclopedia. In Robert Korfhage, Edie M. Rasmussen, and Peter Willett, editors, *Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Pittsburgh, PA, USA, June 27 - July 1, 1993*, pages 181–190. ACM, 1993.

[24] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[25] Beth Levin. *English Verb Classes and Alternations: A preliminary investigation.* University of Chicago Press, Chicago and London, 1993.

[26] Dekang Lin. PRINCIPAR: An efficient, broad-coverage, principle-based parser. In *Proceedings of the 15th Conference on Computational Linguistics*, pages 482–488, Morristown, NJ, USA, 1994. Association for Computational Linguistics.

[27] Jimmy Lin and Boris Katz. Question answering from the web using knowledge annotation and knowledge mining techniques. In *CIKM '03: Proceedings of the Twelfth International Conference on Information and Knowledge Management*, pages 116–123, New York, NY, USA, 2003. ACM Press.

[28] Kenneth C. Litkowski. Question-answering using semantic relation triples. In *TREC*, 1999.

[29] Kenneth C. Litkowski. Syntactic clues and lexical resources in question-answering. In *TREC*, 2000.

[30] Kenneth C. Litkowski. Use of metadata for question answering and novelty tasks. In *TREC*, pages 161–176, 2003.

[31] Gregory Marton and Boris Katz. Using semantic overlap scoring in answering TREC relationship questions. In *Proceedings of the LREC 2006*, Genoa, Italy, May 2006.

[32] Igor Mel'čuk. *Dependency Syntax: Theory and Practice.* State University of New York Press, 1988.

[33] Diego Mollá and Mary Gardiner. AnswerFinder—question answering by combining lexical, syntactic and semantic information. In *Proceedings of the Australasian Language Technology Workshop (ALTW04)*, 2004.

[34] Martin F. Porter. *An algorithm for suffix stripping*, pages 313–316. Readings in Information Retrieval. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[35] Sameer Pradhan, Wayne Ward, Kadri Hacioglu, James H. Martin, and Daniel Jurafsky. Shallow semantic parsing using support vector machines. In *HLT-NAACL*, pages 233–240, 2004.

[36] Vasin Punyakanok, Dan Roth, and Wen tau Yih. Mapping dependencies trees: An application to question answering. In *the 8th Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, January 2004.

[37] Daniel D. Sleator and Davy Temperley. Parsing English with a link grammar. In *Third International Workshop on Parsing Technologies*, 1993.

[38] Martin M. Soubbotin. Patterns of potential answer expressions as clues to the right answers. In *TREC*, 2001.

[39] Renxu Sun, Jing Jiang, Yee Fan Tan, Hang Cui, Tat-Seng Chua, and Min-Yen Kan. Using syntactic and semantic relation analysis in question answering. In *TREC*, 2005.

[40] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, 1979.

[41] Pasi Tapanainen and Timo Järvinen. A non-projective dependency parser. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 64–71, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[42] The Apache Software Foundation. Lucene. `http://lucene.apache.org/java/docs/index.html`.

[43] The World Wide Web Consortium (W3C). XML Path Language (XPath) 2.0. `http://www.w3.org/TR/xpath20/`, 2006.

[44] C. J. Van Rijsbergen. *Information Retrieval.* Department of Computer Science, University of Glasgow, second edition, 1979.

[45] Ellen M. Voorhees and Dawn M. Tice. The TREC-8 question answering track evaluation. In *TREC*, 1999.

[46] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[47] Kaizhong Zhang. *The editing distance between trees: Algorithms and applications.* PhD thesis, Courant Institute, Department of Computer Science, 1989.

[48] Kaizhong Zhang and Tao Jiang. Some MAX SNP-hard results concerning unordered labeled trees. *Information Processing Letters*, 49(5):249–254, 1994.

[49] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.

[50] Kaizhong Zhang, Rick Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42(3):133–139, 1992.