Learning Semantic Patterns for Question Generation and Question Answering

Hugo Patinho Rodrigues

Ph.D. Thesis Proposal

Thesis Advisory Committee

Advisors: Prof. Doctor Maria Luísa Coheur
          Prof. Doctor Eric Nyberg

Jury:    Prof. Doctor Teruko Mitamura
         Prof. Doctor Robert Frederking

May 2017
Abstract

In recent years we have been watching an increase of educational research. A huge effort was made to push online courses, or Massive Open Online Courses (MOOCs), and Intelligent Tutoring Systems (ITSs). These paradigms have different pros and cons, but they share the same limitation: the content has to be provided manually by experts. This creates a huge burden on professors and instructors, who need to spend more time creating simple questions to automatically assess the students.

In this thesis, we propose a example-based system to automatically perform the task of creating Question/Answer pairs given an information source, through Question Answering (QA) and Question Generation (QG).

QA systems try to answer questions posed in natural language, in contrast to search engines that just return a set of related documents given some keywords, while QG systems, on the other hand, try to do the inverse, that is, to generate questions from raw text. Although each task has its own characteristics, some steps can be shared or reversed.

Many works in different areas use example-based strategies. Regarding QA and QG, pattern-based approaches have been widely applied; however, most of these methods only go to the lexical and syntactic level, and do not use past interactions to learn new knowledge that can be adapted and reused. In this work, we intend to add a semantic layer to this kind of approach, creating a system that automatically creates Question/Answer pairs from an information source, while being able to learn from past examples how to perform better.

Our proposed QG system aims at providing a tool for many scenarios. First, it can be used as an authoring tool to help professors create content for their courses. The relevance and usefulness of the generated questions will be evaluated, as well as the time saved by the teacher. Secondly, it can prove to be an useful tool to create a large dataset of Question/Answer pairs that can be used by real external QA and QG systems. This will be evaluated by qualitative
measures, like grammatical correctness and semantic soundness, through the crowd (Amazon
Mechanical Turk), and extrinsically, as a supporting data source for real systems in their
tasks (for instance, to train neural network systems).
## Contents

1 Introduction .................................................. 1
   1.1 Motivation and Vision ................................. 1
   1.2 Proposed Work Overview .............................. 2
   1.3 Document Overview .................................... 4

2 Problem Statement ............................................ 5
   2.1 Question Generation System ......................... 5
   2.2 Large Question/Answer pairs Dataset ................. 6
   2.3 Quiz Creation .................................... 7

3 Related Work .................................................. 9
   3.1 Question Answering ..................................... 9
      3.1.1 Lexical and Syntactic Patterns for Question Answering ...... 10
      3.1.2 Improving Question Answering Patterns with Semantic Features 12
   3.2 Question Generation ..................................... 14
      3.2.1 Rule-based Question Generation .................... 16
      3.2.2 Semantic Features in Rule-based Question Generation .... 18
      3.2.3 Merging Syntactic and Semantic Features for other Applications 18
   3.3 Patterns in Other Domains ............................. 19
   3.4 Case-based Paradigm ................................... 21
   3.5 Resources for Question Answering and Question Generation .... 23
      3.5.1 Annotated Corpora and Lexical Resources ............. 23
      3.5.2 Parsers and Tools ................................ 24
   3.6 Discussion .................................. 25

4 Question Generation System ................................ 27
   4.1 Towards Semantic Patterns ............................ 28
      4.1.1 Formalization ...................................... 30
         4.1.1.1 Sentence Representation ....................... 30
         4.1.1.2 Pattern ................................... 31
      4.1.2 Implementation ................................... 32
         4.1.2.1 Sentence Representation ....................... 32
         4.1.2.2 Pattern ................................... 32
   4.2 Pattern Acquisition .................................. 33
      4.2.1 Formalization ...................................... 33
         4.2.1.1 Alignment of Seeds’ Components ................ 33
      4.2.2 Implementation ................................... 34
         4.2.2.1 Bag of tokens for a sentence ................. 34
         4.2.2.2 equiv function ................................ 35
1.1 Motivation and Vision

In recent years we have been watching an increase of educational research. Both online courses (Massive Open Online Courses (MOOCs)) and Intelligent Tutoring Systems (ITSs) have been the focus of researchers towards a better educational setting. The former are accessed by millions of students online, and focus on the quality of the content provided, while the latter are small, contained systems that interact with students but which, to do so, focus only in a small portion of a course’s content.

However, despite all the pros and cons both strategies might have, one piece is always missing: the actual content. This is done by professors, and it requires them to not only provide learning artifacts, but also questions, hints, and any other kind of study materials that can be useful to students. This is, of course, highly time-consuming, and studies show that authoring tools may yield a 75% decrease in the costs of building an ITS [Koedinger et al., 2004].

This lack of automation leads us to the necessity of providing teachers with means to create content in a faster way. Question Generation (QG) and Question Answering (QA) are two Natural Language Processing tasks that tackle exactly the problems of generating questions and retrieving answers, and are, thus, good candidates to be useful tools in the process of automation of such tasks for professors. Both have been studied for some time now, and significant improvements have been made in the past few years. QA systems try to answer questions posed in natural language, in contrast to search engines that just return a set of related documents given some keywords. This is challenging because natural language is flexible and questions may be formulated in different forms. Also, returning a single and concise answer is a much harder problem than returning a set of more or less related documents.
QG systems, on the other hand, try to do the inverse, that is, to generate questions from raw text. In the context of educational research, these can be seen as tools to automatically create Question/Answer pairs that can be utilized for educational purposes.

Meanwhile, with the advancement of neural networks, systems require large datasets to work with and both QA and QG have been lacking such data. The trend is changing, now that SQuAD [Rajpurkar et al., 2016] is available. SQuAD is a large dataset of Question/Answer pairs, created by humans through Amazon Mechanical Turk (AMT) (over 100,000 questions across over 500 Wikipedia articles). However, there is a clear limitation on how many questions are created, as it is virtually impossible to have workers come up with all possible questions, specially without spending a lot of time and money. Our system can, thus, play an important role as a tool to generate a significant number of Question/Answer pairs that can prove useful to extend any current dataset.

1.2 Proposed Work Overview

Although each task (QG and QA) has its own characteristics, it makes sense to look at them together, as many steps can be shared or reversed. More concretely, many of such systems use patterns in order to either extract answers from text (QA) or to transform sentences into questions (QG). For example, given the question *What is the capital of Portugal?* and the pattern $x_{\text{answer}}$ is the capital of $Y$, one could extract the answer *Lisbon* from a sentence like *Lisbon is the capital of Portugal*. On the other hand, it would also be possible to create the original question from that same sentence, with a rule that swaps the $x_{\text{answer}}$ in the pattern above for the appropriate Wh-word. Figure 1.1 depicts this idea. Many works follow a pattern-based approach [Heilman, 2011, Kalady et al., 2010], and some actually include a learning process to automatically generate such patterns [Curto et al., 2011]. This approach of learning by analogy, or example-based systems [Aamodt and Plaza, 1994], has also been applied in other domains, such as in creation of dialog systems [Nio et al., 2014] or translation of unknown words [Langlais and Patry, 2007].

Despite recent progresses, the used pattern methods usually only go to the lexical and syntactic level, sometimes adding Named Entities and Semantic Role Labelers as semantic
1.2. PROPOSED WORK OVERVIEW

Figure 1.1: Generic overview of pattern applicability in the tasks of QA and QG.

features. Also, although some works try to automatically learn new patterns, to the best of our knowledge, none of them actually improves previously learnt patterns, that is, systems do not try to learn from past interactions. Finally, these patterns are only used in one task alone, either QA or QG. In this work we intend to tackle these problems by adding a semantic layer, and a learning and revision process, based in a loop through the tasks of QA and QG. The ultimate goal is to create more flexible, generic, and applicable patterns, with the least human-interaction possible, in order to provide the user with new Question/Answer pairs from new, unseen text. Ultimately, this could save experts several hours in the procedure of creating content for educational purposes, and can help us creating a large dataset for QA and QG development.

The impact of our system can be measured on each domain in different ways. For content creation, the utility and relevance will play an important role, so the professor can actually save time when creating quizzes for his course. This can be evaluated in a formative way, by closely interacting with professors. For the generation of a massive number of Question/Answer pairs to create a dataset, AMT will play an important role as a way to quickly grade the quality of the generated questions. However, the impact of such dataset can only be measured if we take a concrete task. Therefore, real external systems should be used with the new data,
which allows us to evaluate their performance through comparable metrics, like accuracy and recall, against their own baseline.

1.3 Document Overview

This document is organized as follows:

**Chapter 2** details more precisely the goals of this thesis, in the form of research statements, together with a brief description supporting each one of them. We also formulate the testable hypothesis for our thesis.

**Chapter 3** overviews the relevant related work, namely on Question Answering and Question Generation. The chapter is divided by topic, concluding with a section that covers the major points of interest.

**Chapter 4** presents our proposed solution for the QG system. The chapter goes in depth on each part of the system to be, mapping them into the research statements presented. Each section offers a separate formal view and an implementation description.

**Chapter 5** shows the initial results obtained in different tasks with the current system developed according to the specifications of our proposed QG system solution.

**Chapter 6** details the proposed future work, making a clear connection with our goals and testable hypothesis.

**Chapter 7** sets a tentative timeline for the remaining work, estimated for one year of work.

**Chapter 8** finally concludes the document.
Problem Statement

We introduced a couple of points we want to address in this thesis in the introduction. In this chapter, we detail those more concretely in the form of research statements and the testable hypothesis.

2.1 Question Generation System

Research Statement Adding semantic features to a pattern-like approach will improve the quality of the generated questions and extracted answers in the tasks of Question Generation (QG) and Question Answering (QA), respectively.

We believe that it is possible to improve the tasks of QG and QA by adding stronger semantic features to an already in use pattern-based approach. Most used patterns are lexico-syntactic, meaning that they are limited by the strong lexical constraints they contain, while being too generic in some cases by accepting any entities matching the Part of Speech (POS) tags they have.

Research Statement It is possible to increase the precision and recall of a pattern-like approach for QG and QA by restraining and/or generalizing the previously acquired patterns.

There are some successful studies that report both QG and QA systems that start from a few seeds and automatically learn new patterns [Curto et al., 2012, Ravichandran and Hovy, 2002]. However, this is a one time initialization process. As far as we are aware, only Mendes [2013], Shima [2015] and Velardi et al. [2013] employ a strategy of using past interactions to improve the system performance in future interactions. We believe that this approach
is under-exploited and that it is possible to refine previous patterns, contributing to better patterns in the long run.

**Research Statement** Question Answering and Question Generation can be two sides of the same coin and be used together: using patterns to retrieve answers for good questions (that is, performing QA) can evaluate those patterns as good sources for quality QG.

To the best of our knowledge, QG and QA were never used in the same work as means to improve each other. Closely related to the previous point, we believe that a loop between the two tasks can create the chance to cover overlooked aspects of each of these tasks when tackled individually. This would also mean a faster and more efficient system, that could outperform a human in the same task of generating questions from a document.

This set of research statements respect to the capability of generating questions only, that is, to our QG system. The evaluation of such system and generated data, done through Amazon Mechanical Turk (AMT), will show how it performs:

**Hypothesis 1 (H1):** The proposed QG system outperforms current state of the art systems, measured by qualitative metrics as grammaticality, semantical correctness, relevance, among others, evaluated independently by the crowd through AMT.

### 2.2 Large Question/Answer pairs Dataset

**Research Statement** Our QG system can be used to create a large dataset of Question/Answer pairs that can be used by external systems in their tasks of QA or QG.

One problem in the field of QG has been the lack of data to evaluate systems' performances, while in the QA field there was never massive amounts of data to train new neural network systems. This has recently changed with the creation of SQuAD, but all questions were created by humans, which puts a limitation of the completeness of the dataset. Our
2.3. QUIZ CREATION

system can be an important tool to generate a large dataset that complements SQuAD.

**Hypothesis 2 (H2):** The proposed QG system can be used to create a large dataset of Question/Answer pairs that can be used by external systems as support data. The impact can be evaluated by measuring the difference in performance those systems attain, as measured by quantitative metrics like accuracy and recall.

2.3 Quiz Creation

**Research Statement** Our QG system can be applied as an authoring tool, automatically creating Question/Answer pairs of relevance for a professor to use in content creation.

Content creation is a bottleneck when building advanced teaching platforms, like Intelligent Tutoring Systems (ITSs) or Massive Open Online Courses (MOOCs), but authoring tools can decrease significantly the effort experts need to put into them. Our system can prove useful on the task by reducing the time spent by professors on tasks of this sort.

**Hypothesis 3 (H3):** The proposed QG system will help professors create content for their courses, like quizzes, by automatically generate questions that can be used or easily modified for usage. The impact of such claim can be tested through a formative evaluation, by measuring the utility and relevance of the generated questions, and the time spent by professors on the task with and without the system.

Finally, the last two hypothesis lead us to a forth one that we are presenting here, but is not part of our plans as of now:

**Hypothesis 4 (H4):** A correlation between AMT evaluations and the teacher’s feedback can be found, that is, generated questions annotated as quality questions correlate to the questions found useful and relevant by a professor when during content creation.
Related Work

In this chapter we overview the work done in Question Answering (QA) – Section 3.1 – and Question Generation (QG) – Section 3.2 –, giving special attention to approaches that use patterns. We also look at works that use patterns in other domains (Section 3.3), and briefly discuss the case-based paradigm (Section 3.4). Finally, we investigate different resources and tools that might help us build our system (Section 3.5).

3.1 Question Answering

QA systems try to find the answer to questions in natural language. Usually these are factoid questions, meaning that they inquire about a fact and, thus, require a concise answer, instead of a description or an explanation. Most QA systems follow a 3-step architecture: (1) Question Interpretation, (2) Passage Retrieval, and (3) Answer Extraction.

First, the question is processed and interpreted, from what one or more queries result. These are passed down to the Passage Retrieval step, where the system tries to find information related to the question. For this purpose, it can use pre-defined corpora, local databases, or the web. Finally, according to the fetched snippets or documents, the system extracts possible answers and ranks them, selecting one or more as correct answers (Answer Extraction step).

To exemplify the process, consider the following question: What is the capital of Portugal? The first step could generate queries like capital Portugal, Portugal’s capital is and/or capital(Portugal, x?), depending on the target information source. Then the system would retrieve data from corpora, databases or the web, gathering a set of snippets or documents related to the queries above (Passage Retrieval step). Finally, in the Answer Extraction step, the system uses those snippets or documents to extract candidate answers,
from which it would try to select one or more as correct answers. A possible set of candidate answers is \{Lisboa, Guimarães, Lisbon, Lisboa, Porto\}.

In this section answer extraction will be the focus, namely when it is done by the means of patterns.

3.1.1 Lexical and Syntactic Patterns for Question Answering

The use of patterns to extract text is a technique used for some time now. They go back to 1992, when Hearst [1992] used them in relation acquisition. In QA, many works followed this strategy of using handwritten lexico-syntactic patterns (like regular expressions) to extract the candidate answers [Fleischman et al., 2003, Haddad and Desai, 2008, Joho and Sanderson, 2000, Kupiec, 1993, Sarmento et al., 2008, Soubbotin, 2001].

However, designing such patterns is labor intensive and does not provide good coverage, as it is impossible to include all intended scenarios. To overcome these problems, other approaches have been being pursued. One line of research is the automatic acquisition of such patterns.

Ravichandran and Hovy [2002] designed a system that learns the patterns from one or more seeds. A seed is a sample containing one or more question terms (for example ‘Mozart’) and the correct answer (‘1791’), for the given question category (in this case ‘date of death’). The pair is submitted to a search engine and the top \( N \) results containing both terms are used to extract a pattern able to match the question terms and answer. Those patterns are then generalized, by swapping the question terms and answer by the <NAME> and <ANSWER> tags, respectively. In this example, patterns like ‘<NAME> died in <ANSWER>’, and ‘<NAME> (1756 - <ANSWER>)’ could be extracted. The process is repeated with other seed pairs, learning thus more patterns. After this step, new queries are created from the seeds with only the question terms (that is, without the answer this time) and the previously obtained patterns are used to extract the answers. The answers retrieved with the patterns are then compared with the expected answer. The ratio of correct answers retrieved is the precision of the pattern that originated such answers. The values found represent a probability of each pattern to find the correct answer and are used as a ranking system.

Bouma et al. [2011] report a relation extraction technique for both open and closed domain
3.1. QUESTION ANSWERING

QA, in order to improve these systems’ performance. This work follows their experiments with Joost, a QA system for Dutch, where the authors found that using Database (DB) relations produced better results than using Information Retrieval techniques. In order to populate such DBs, and similarly to Ravichandran and Hovy [2002]’s work, the system starts with a set of seeds, corresponding to some kind of relation between entities (for example, country-capital), and learns patterns from sentences containing both relation terms. The difference is that this system uses a dependency parser, resulting in dependency patterns instead of a typical surface pattern: \texttt{Arg1+subj$\leftarrow$leid$\rightarrow$pc+tot+Arg2$^1$}. The validation process follows the same process as the original work, in which patterns are weigh according to their precision. This allows the system to build an offline knowledge base that answers these type of questions with greater precision.

Fader et al. [2013] study open QA as well. Realizing that DBs contain a lot of structured information that could be exploited by a QA system and that those are not being used to their full potential, the authors propose a system that maps new questions into queries for those DBs. In order to map new unseen questions into a query, the system is based on paraphrases, that is, the new questions are \textit{translated} into a known question and then mapped into a known query. The relevance for us is that it all starts with a set of question patterns/query pairs, like \texttt{Who r e$?$/r(?, e)} or \texttt{When did e r$/r$-in(e, ?)}, where \texttt{r} represents a relation, and \texttt{e} an event. These patterns are used to learn a lexicon that will be employed in mapping paraphrases with equivalent meaning. For example, knowing that the question \texttt{What is the population of New York?} should be answered through the query \texttt{population(?, new-york)}, the question \texttt{How big is nyc?} needs to be aligned to the first question. The lexicon learned will have the mapping between \texttt{population} and \texttt{big}, \texttt{New York} and \texttt{nyc}, and \texttt{What is} and \texttt{how}, allowing the system to create the correct query.

Just.Ask [Mendes, 2013] is an end-to-end QA system with no specific domain targeted. It uses different approaches to retrieve the correct answer to multiple questions, depending on their type. For example, for \texttt{NUMERIC} and \texttt{ABBREVIATION} type questions, Just.Ask utilizes a set of regular expressions (hand-written) to extract candidate answers, while for

\footnote{Examples extracted from Bouma et al. [2011].}
questions of type Human:Individual, requiring a more flexible approach, Just.Ask uses
a machine learning-based named entity recognizer. Just.Ask also has a pattern learning
strategy, similar to the works presented before Ravichandran and Hovy [2002]. Patterns
created this way are lexico-syntactic, based on parsing trees. However, Just.Ask goes a step
further on this matter and, in addition to this process, it also employs a learning process,
where new data contributes to learning new patterns. To do so, the system receives feedback
from the user, who provides the correct answer to the current question, and then the system
uses the new information (that is, the new Question/Answer pair) to learn new patterns (in
the experiments conducted, the system would consult the goldstandard in order to reciprocate
the user’s feedback).

3.1.2 Improving Question Answering Patterns with Semantic Features

Automatic extraction of patterns saves the developer from all the heavy work of designing
suitable patterns for a given domain. However, the automatic acquisition of patterns like the
presented before usually results in too specific lexical patterns (especially when those lexical
cues are between the expected answer and the question terms), thus decreasing the system’s
recall and overall performance – notice how some date of death patterns from previous section\(^2\)
have a specific date of birth, thus not being possible to apply to other people. With this in
mind, many systems (including a few presented before) started incorporating a semantic layer
to their patterns, in order to augment their applicability. Greenwood and Gaizauskas [2002],
Mendes [2013] and Schlaefer et al. [2006] all use Named Entities (NEs) to abstract certain
aspects of the patterns, in order to match more sentences. Taking the example before, the
pattern would be extended to ‘\(<\text{NAME}> \ (<\text{DATE}> - <\text{ANSWER}>)\)’, allowing the system to find
the answer for more entities. However, there are some examples of systems that use more
semantic features than just NEs.

Shen et al. [2005] present an interesting take on sentence matching for QA, along with
the use of syntactic patterns. The authors use a modified Word Edit Distance (Levenshtein
distance [Levenshtein, 1966]), where instead of the typical binary 0 or 1 score, word differences
\(^{2}\)For instance, ‘\(<\text{NAME}> \ (1756 - <\text{ANSWER}>)\)’.
are scored in increments of 0.2. This is done by matching words semantically instead of only lexically, using WordNet [Miller, 1995] to find these alternations. More concretely, identical words cost 0, words with the same morphological root cost 0.2, hypernyms or hyponyms cost 0.4, words in the same synset cost 0.6, words with subsequent relations cost 0.8, and, finally, different words cost 1. Another work also uses WordNet in a similar fashion in order to improve word-matching [Yih et al., 2013], using it in a machine learning model that utilizes feature vectors representing the word-matching between the question and candidate sentences’ terms according to the WordNet semantic relations.

However, other systems take it a step further. Kosseim and Yousefi [2008] designed a QA system that uses automatically acquired patterns with semantic restrictions. The patterns are automatically discovered in a similar fashion to Ravichandran and Hovy [2002]’s work, by using Question/Answer seed pairs. The semantic restrictions are created through the sense of the verb in the sentence originating the pattern, and are then used in matching time to compare the question’s and candidate sentence’s verb senses (also extended to the noun phrases in the candidate sentence as well), through synonym and hypernym relations from WordNet. The patterns are created from the sentences by substituting the question terms for tags, the answer’s noun phrase for its NE, and the remaining noun phrases and verbs for their syntactic tags, removing all other terms except for prepositions: for example, the pattern <Organization> <verb> <QArg1> in <QArg2> | senseOf(provide) is created for the Question/Answer pair Who provides telephone service in Orange County, California?/Pacific Bell from the sentence Pacific Bell still provides nearly all of the phone service in Orange county, California. Finally, patterns are associated with a weight, based on features like frequency, length, and the similarity between the sense captured and the original question’s sense.

Bouma et al. [2011] (presented in the previous section) also uses semantic features when adapting their system to a closed domain QA – medical domain. In open domain QA, an approach like the one used (using seed pairs to find common patterns, like country-capital) is suitable, but the same is not true for closed domains, where one does not have access to such large corpora and each instance to be captured is less likely to appear often. Thus, the
authors resort to a medical annotated corpus, where casual relations are noted, together with each argument’s semantic annotation:

\[
<rel\_cause>
Een tekort aan
<con\_body\_part>insuline</con\_body\_part>
leidt tot
<con\_disease>sikerziekte</con\_disease>
</rel\_cause>

This corpus is used to train a causal sentence extractor, from which patterns are created and augmented with the semantic labels from the original corpus for the arguments of the relation (Arg1+subj←leid→pc+tot+Arg2 into NeoplasticProcess+subj←leid→pc+tot+DiseaseOrSyndrome$^3$).

### 3.2 Question Generation

The goal of QG is to automatically create questions from text. However, different types of questions may be asked. Usually, systems focus on Yes/No and Wh-questions$^4$, as they tend to be easier to generate and can be extracted more frequently. But it can be useful to create other types of questions, such as Why questions, or even more complex questions, like Give an example of (…), Enumerate (…) or Describe (…) type of questions$^5$.

Despite the similarities QA and QG might have, as one can consider them symmetric, QG has a few limitations not present in QA. First, it contains a subtask of natural language generation: the questions created must be grammatical and semantically correct, besides, of course, of being useful just as answers retrieved may be right or wrong in QA. Secondly, it is hard, if not impossible, to have a gold standard of what are the right questions to create from a text, just as it is not possible, in QA, to automatically evaluate the answers to descriptive questions.

---

$^3$Examples extracted from Bouma et al. [2011].
$^4$Who, what, where, when are the main examples.
$^5$Forăscu and Drăghici [2009] discuss a taxonomy for QG and other tasks.
3.2. QUESTION GENERATION

Take into consideration the following sentence:

\[
\textit{Ana, the first child of Carla, gave an apple to Bob after class and he thanked her.}\tag{3.1}
\]

This simple sentence can be decomposed into different chunks. It is a compound sentence, \textit{i.e.}, it can be separated in two different sentences: \textit{Ana, the first child of Carla, gave an apple to Bob after class} and \textit{He thanked her}, and can also be stripped of additional information: \textit{Ana is the first child of Carla}. Many systems perform these kind of tasks in order to facilitate the generation process.

Multiple questions can be created for the sentence presented:

\begin{align*}
\text{Who gave Bob an apple?} \\
\text{To whom did Bob thank?} \\
\text{What did Ana give to Bob?} \tag{3.2} \\
\text{To whom did Ana give an apple?} \\
\text{When did Ana give an apple to Bob?}
\end{align*}

Of course, questions such as \textit{Who gave Bob an apple after class?} are also valid; they only differ by having more information, but that does not change the meaning of the question. Also, if anaphora resolution is not performed, from the second part of the sentence one can create the following question: \textit{Who thanked her?}, which lacks enough information to be answered without context.

The limitlessness of questions goes beyond when we take into consideration other verbs or semantic formulations. For instance, \textit{From whom did Bob receive an apple?} is a perfectly admissible question, but it requires another source of information: the opposite of \textit{give} is \textit{receive}, and this action involves two subjects.

These and other problems are discussed in greater detail by Heilman [2011].
3.2.1 Rule-based Question Generation

QG has been studied for some time now and, from 2008 to 2011, there was a huge contribute to its research mostly due to a QG workshop [Rus and Lester, 2009, Rus et al., 2010, 2011, 2012][6] taking place, with the last two containing a Shared Task Evaluation Campaign [Rus et al., 2011, 2012]. Many papers were submitted during these 4 years, but only a few contain detailed descriptions of the systems and a good evaluation process.

In this section we will present a general approach that is followed, at least partially, by most systems. Kalady et al. [2010], Ali et al. [2010], Yao and Zhang [2010], Varga and Ha [2010], and Heilman [2011] are examples of such systems.

There are three main tasks shared among most QG systems: sentence decomposition, syntactic parsing, and Named Entity Recognition (NER). The first tries to deal with sentences like Sentence 3.1, as discussed earlier, simplifying them in order to ease the process of question generation [Kalady et al., 2010, Yao and Zhang, 2010].

Syntactic parsers are also widely used. They allow the mapping of sentences into trees, by grouping words and the associated tokens into nodes representing their syntactic tags. These tags, such as noun phrases (NP) and prepositional phrases (PP), identify different targets for the question generation process, with their textual value being the answer to those questions. Figure 3.1 shows an example of a syntactic tree parsed from the sentence In the 14th century Dante has written The Divine Comedy, considered the preeminent work of Italian literature (from Mendes [2013]).

Finally, NER is used to help choosing the correct Wh-word to use for the question. For example, the sentences Bob hit Ana and The car hit Ana have the same syntactic tree, but the chosen Wh-keyword must be different, depending on the subject of the sentence, if one wants to generate a question of the type <Wh-word> hit Ana?

Given the data acquired with the previous tasks, systems resort to a sort of rules or patterns to create the desired questions. For example, some systems design rules as NP₁ VB NP₂ → Wh-word VB NP₂?, which covers the example before. Ali et al. [2010], Pal et al. [2010], Varga and Ha [2010] are examples of systems that use such strategies. Others, like Heilman

and Smith [2009, 2010], Kalady et al. [2010] and Wyse and Piwek [2009], use tree operations directly on the parse trees to transform the sentences into the questions. Most of these rules are created manually, but, just like in QA, some systems also tried to automatically create the patterns necessary to generate questions. TheMentor [Curto et al., 2011, 2012] is an example that uses a similar approach to Ravichandran and Hovy [2002]: Question/Answer pairs of a given type are submitted to a search engine and sentences that contain both the question and answer terms are turned into patterns, by replacing those terms by syntactic tags, from which it is possible to generate questions of the original type. Both TheMentor and Guo et al. [2016] generate distractors, although the latter generates fill the blank type of questions through syntactic rules.

Many other minor tasks are performed as well, such as inversion of the subject and auxiliary verb, and decomposition of the main verb, (i.e., the transformation ate to did eat). Other systems include coreference resolution [Kalady et al., 2010], as pointed to be needed by Heilman [2011], and Mazidi and Nielsen [2015] actually combines multiple techniques, like dependency parsers, syntactic parsers, and discourse cues (which the authors call of using multiple views of the text) to improve the quality of the questions generated.

Mannem et al. [2010] also ranks the generated questions in the end, according to the depth of the verb in the parsed graph (it uses a dependency parser instead of a syntactic parser). Heilman [2011] applies an overgeneration method, which produces more questions.
(and potentially more errors) to apply, then, a statistically ranking strategy, by using a linear regression model, which pushes quality questions to the top, improving this way the top-N precision of the system. Lindberg et al. [2013] also build a classifier based on the judgement provided by the human rater, to classify the questions on the learning value (one of the parameters rated).

### 3.2.2 Semantic Features in Rule-based Question Generation

Some systems include other semantic resources, besides NER and coreference resolution. The main example is the use of Semantic Role Labelers (SRLs) to help design the generation rules, as knowing the arguments of the verbs allow a finer detail on the transformational actions to be taken [Chen, 2009, Lindberg et al., 2013, Mannem et al., 2010, Mazidi and Nielsen, 2014, Pal et al., 2010]. Another example is the use of WordNet to elaborate on what kind of entity the *Which* questions are about (for instance, *Which country*), based on the hypernym of the target NP [Varga and Ha, 2010].

More recently, Araki et al. [2016] developed a QG system based on strong semantic features, extending the scope of the target from a sentence to a paragraph. It analyses events described in paragraphs and what triggers those events, and, through a set of manually created rules, it generates a question regarding that event. However, this system was possible due to the existence of a labelled corpus with such semantic relations (events, triggers, coreference, etc.), making it highly dependent of such data. The proof of concept is, nonetheless, an important step for this research topic.

### 3.2.3 Merging Syntactic and Semantic Features for other Applications

Despite the conjunction of both techniques not being widely used for QG, there are other domains which can be solved with a more machine learning-focused approach, which can take advantage of this multitude of features. For instance, for Semantic Textual Similarity, Vu et al. [2015] uses as features, among others, the sentences’ alignment number of matches, the number of shared attributes from subject-verb-object structures, identical NEs, semantic similarity through word embeddings, and synset similarity through WordNet.
Both Pazienza and Pennacchiotti [2005] and Tymoshenko and Moschitti [2015], in two different domains (textual entailment and answer passage reranking, respectively), use a similar approach to the one presented by Mazidi and Nielsen [2015] of combining syntactic trees with dependency trees, in a hybrid representation of the sentence, adding semantic resources as features for matching the tokens, like Yago [Suchanek et al., 2007] and WordNet [Miller, 1995].

3.3 Patterns in Other Domains

DIPRE [Brin, 1999] and Snowball [Agichtein and Gravano, 2000], which was based on the first, are two relation extraction systems that also use patterns. Both were used in a single domain (author/books and organization/location, respectively), and learn patterns from a small set of seeds (pairs of the relation to be acquired). The patterns consist of two tags (the entities to extract) and three flexible fields: on the left, right and middle of the tags (<left, tag1, middle, tag2, right>). The original system has tags defined as regular expressions and the other three fields as the exact lexical sequences. Snowball evolved this concept by adding NER, using NEs in place of regular expressions for the tags, and a vector of terms with associating weights for the remaining fields (for example, <\{the, 0.2\}, LOC, \{\text{-}, 0.5\}, \{\text{based}, 0.5\}, ORG, {}>, that captures sentences like the Irving-based Exxon Corporation) [Agichtein and Gravano, 2000]. These patterns are learnt from the seed pairs, by identifying the elements of the relation on text, and capturing the surroundings. When multiple instances are found, if they are similar enough (degree of matching, based on the inner product of the term vectors), a clustering algorithm is used to merge those patterns (that is, the vectors for the three fields are combined and reweighed). Finally, to find new pairs of relations, the generated patterns are used to match (using the degree of matching) sentences containing both locations and organizations.

OntoLearn Reloaded [Velardi et al., 2013] is a system designed to create a taxonomy from scratch in any domain. To do so, the system identifies relevant terms from the domain, expands that lexicon by automatically extracting definition sentences referring those terms and builds a graph-like taxonomy connecting those. Although the details in graph construction
will not be addressed, it makes sense to get into details on how hypernyms are extracted.

To start, the system automatically extracts relevant terms in the domain, from the corpora used. These compose the initial terminology. Then, learning from a dataset of manually annotated definitional sentences\(^7\), the system identifies relations following the pattern \(<\text{Def}, \text{V}, \text{Hyp}, \text{R}>\) [Navigli and Velardi, 2010], corresponding, respectively, to the term to be defined (Def – a chiaroscuro), the verb phrase describing it (V – is), the phrase that represents the hypernym (Hyp – a monochrome picture) and the rest of the sentence that carries some differentiation meaning to the relation (R – in arts). Each one of the sentences found is transformed into two different patterns: a tagged pattern (containing Part of Speech tags) and a star pattern (containing wildcards – ∗). Table 3.1 shows an example for the definitional sentence presented. Sentences are clustered based on their star pattern, and, for each group, the algorithm starts by creating a graph (not related with the taxonomy graph referred above) based on the terms of the tagged pattern of one of the sentences in the cluster. Then, for each other sentence in the group, the system expands the graph with the appropriate edges corresponding to the minimal mismatches. From this process results a Word-Class Lattice for each cluster. These are finally applied to the corpora for each term in the initial terminology, extracting different definitions (that is, hypernym relations) for them. These are used to extend the ontology and the algorithm continues recursively, using the new extracted hypernyms as targets in each new iteration.

<table>
<thead>
<tr>
<th>Original sentence</th>
<th>In arts, a chiaroscuro is a monochrome picture.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tagged pattern</td>
<td>In NNS, a &lt;TARGET&gt; is a JJ NN.</td>
</tr>
<tr>
<td>Star pattern</td>
<td>In ∗, a &lt;TARGET&gt; is a ∗.</td>
</tr>
</tbody>
</table>

Table 3.1: The tagged and star patterns used by Velardi et al. [2013].

Shima [2015] developed a pattern acquisition system for paraphrases, accounting lexical diversity. The idea is that most systems are limited by different formulations of a given pattern, like \(X\) died of \(Y\) and \(X\) has died of \(Y\), and are not able to expand to other semantically equivalent formulations, such as \(X\) fell victim to \(Y\) or \(X\) succumbed to \(Y\). The system starts from a few seeds, like Ravichandran and Hovy [2002] does, but it adds a iter-

\(^7\)Available at http://icl.uniroma1.it/wcl.
3.4. CASE-BASED PARADIGM

Case-based reasoning (or analogical learning) is a paradigm that can be more easily understood as an example-based approach. The general idea is that a system employing such technique does not use general knowledge only, but also specific knowledge acquired with previous situations (cases/problems experienced). New problems are, thus, solved by reusing solutions used on similar past occasions.

The parallel can be drawn for our domain: previously seen question/sentence pairs are supposed to be examples for future questions or sentences. The solution to be reused is the set of operations applied to transform the sentence into the question, or vice-versa.

Although it is not our goal to do a in-depth analysis of the state of the art in this area,
we find important to mention a few works and ideas that can be useful in our work.

According to Aamodt and Plaza [1994], case-based reasoning follows four steps: 1) Retrieve, where one is supposed to retrieve the most similar examples, 2) Reuse, in which the knowledge of the retrieved case is reused (by just copying or adapting it), 3) Revise the found solution, and 4) Retain the pieces of the new experience that will be useful in the future.

The first two steps are already used by most pattern-based systems: finding a pattern to apply and use it matches this part of the cycle. It is, however, on the two other steps we want to make a significant contribution, in comparison to the state of the art systems: being able to Revise patterns used and Retain knowledge on how the patterns perform.

One example that merges this kind of approach and pattern-based paradigms are example-based dialogue systems. Murao et al. [2003] developed a system aimed at retrieving shopping information, where the corpus was collected through a Wizard-of-Oz and the requests and replies are searched, found and modified through pattern matching and slot filling. A similar system and approach was also used by Jung et al. [2006] and Lee et al. [2009], while Nio et al. [2014] collects turn-based interactions from movie scripts and creates a chatbot type of system, where examples are searched through semantic and syntactic similarity, measured as an weighted function of the intersection of WordNet synsets and ration of Part of Speech (POS) tags.

Another representation of analogical reasoning are the class of problems that fit into the template \([A : B = C : D]\), which states a relation between the four entities. These relations can be in different combinations (for example, \(A\) is to \(B\) as \(C\) is to \(D\) is a as valid reasoning as \(A\) is to \(C\) as \(B\) is to \(D\)'), and range many meanings, depending on the elements. For instance, morphological relations of the form \(\text{wife}\) is to \(\text{wives}\) as \(\text{wolf}\) is to \(\text{wolves}\) state how to pluralize words, while semantic relations of the type \(\text{wheel}\) is to \(\text{car}\) as \(\text{window}\) is to \(\text{house}\) represent meronymy. Langlais and Patry [2007] use this type of approach to represent words to be translated. The final goal is to be able to translated unknown words through the existing examples. Refer to Miclet et al. [2008] for a more in depth analysis on analogical similarity.
3.5 Resources for Question Answering and Question Generation

In a growing wide web world, more and more tools and information sources are available to researchers. Many of the systems described before use some of these to some extent, and so will our system. In this section we briefly mention a couple of examples.

3.5.1 Annotated Corpora and Lexical Resources

WordNet [Miller, 1995], a lexical database for English, is one of the most used resources and is used to search for synonyms, hyponyms, and other relations between words. In QA, this can be useful not only to query expansion, creating different queries meaning the same, but also when it comes to answer extraction and selection, as these relations may be important to find potential answers (for instance, to know that whale is a mammal, when answering the question *What is the largest mammal?*) [Ferrucci et al., 2010, Mendes, 2013, Prager et al., 2000]. In QG it can also be used to better select the Wh-word to use when generating new questions [Varga and Ha, 2010].

There are also many annotated corpora available, such as PennTreebank [Marcus et al., 1993], QuestionBank [Judge et al., 2006], PropBank [Palmer et al., 2005], and NomBank [Meyers et al., 2004]. The first two are a collection of corpora where text is annotated with POS, creating a *bank of trees* representing the sentences’ (or questions’) parsing structures. PennTreebank consists of text from different sources such as the Wall Street Journal, while QuestionBank has questions from collections such as the QA tracks from Text REtrieval Conference (TREC)\(^8\). The remaining are an extension to the PennTreebank annotations focused in shallow semantics: PropBank focus in the verbs’ roles, while NomBank focus on the nominalization of verbs and their semantic roles (*her claim* vs. *she claims*).

There are two other important lexical resources to note as well: FrameNet [Baker et al., 2003] and VerbNet [Kipper et al., 2000]. In these, verbs’ annotations are more directed to the roles themselves, while in the treebanks above these are generically represented with agent and object.theme in the form of *Arg0* and *Arg1* (instead of *agent* and *instrument*, for instance, [http://trec.nist.gov/data/qamain.html](http://trec.nist.gov/data/qamain.html))
for the verb *hit*). Also, PropBank and NomBank only cover the instances present in the original corpora, in opposition to these resources, which try to capture the whole semantics of existing verbs, only including a few examples as reference. This means that, in the banks, each verb belongs to its own class. Table 3.2 exemplifies how these resources are organized.

VerbNet is somehow similar to FrameNet, but more focused in grouping verbs according to their syntactic behavior, alike Levin’s classes [Levin, 1993]\(^9\). As an example, the **ApplyHeat** frame and **Cooking** class\(^10\) share most of their verbs (like *broil, cook, fry, sear*...), but VerbNet includes *pickle* as a cooking verb, while FrameNet puts it under the **Preserving** frame, together with verbs like *cure, dry, or salt*.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Frame/Class</th>
<th>Other Verbs</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrameNet</td>
<td>Hit_target</td>
<td>pick off, shoot</td>
<td>Agent</td>
</tr>
<tr>
<td>VerbNet</td>
<td>18.1</td>
<td>bang, bash, click, dash, squash, ...</td>
<td>Agent</td>
</tr>
<tr>
<td>PropBank</td>
<td>hit.01</td>
<td>-</td>
<td>Arg0</td>
</tr>
</tbody>
</table>

Table 3.2: Example of FrameNet, VerbNet and PropBank organization for verb *hit* in the sense of hitting a target.

There is also a project that tries to map all these resources, called SemLink\(^11\), that can be accessed at the Unified Verb Index\(^12\). The downloadable content contains text files with the mapping between PropBank, FrameNet and VerbNet as shown in Table 3.2.

### 3.5.2 Parsers and Tools

The resources described above are used to train lexical parsers, like Stanford parser
[Klein and Manning, 2003] (Heilman [2011] uses it trained on PennTreebank), Berkeley parser
[Petrov and Klein, 2007] (Just.Ask [Mendes, 2013] and TheMentor [Curto et al., 2011] use it trained on QuestionBank), OpenNLP\(^13,14\), or Charniak [2000] parser (trained on Pen-

\( ^9 \)Baker and Ruppenhofer [2002] discusses the difference between FrameNet and Levin’s classes in more detail.

\( ^10 \)FrameNet categories are called frames, while VerbNet’s are called classes.

\( ^11 \)http://verbs.colorado.edu/semlink

\( ^12 \)http://verbs.colorado.edu/verb-index/index.php

\( ^13 \)http://opennlp.apache.org/index.html

\( ^14 \)We found no evidence of what corpora is used to train the system.
3.6 Discussion

In this chapter we overviewed the work done in Question Answering and Question Generation, with special focus on the usage of patterns. Regarding QA, we have seen that multiple systems use lexico-syntactic patterns, with a few improving those by adding NEs. We also saw that QG is mostly rule-based, using rules that boil down to lexico-syntactic patterns, and with just a few systems resorting to semantic features as well, namely SRLs and NEs.

We also mentioned case-based reasoning cycle and how the parallel can be drawn to a pattern-based approach like we want to implement. Concretely, the two last steps, Revise and Retain, are not used in the current state of the art systems, and we believe it is an important component to explore.

There are, however, a few works that should be mentioned, as they are closely related with the previous point. First, the Just.Ask system [Mendes, 2013] that uses past interactions to learn new patterns which can be used in future interactions of the system, and Shima [2015]'s work that also uses past interactions to learn new patterns, and accounts for lexical diversity in order to expand the coverage of the generated patterns. Secondly, Velardi et al. [2013]'s work that tries to merge similar patterns into a single one, and Pang et al. [2003]'s work that

\footnote{http://www.surdeanu.info/mihai/swirl/index.php}

\footnote{We did not find any reference to the year of the corpus.}

nTreebank), and also SRLs (or shallow semantic parsers), like Illinois Semantic Role Labeler [Punyakanok et al., 2008] (trained on PropBank), Senna [Collobert et al., 2011], SwiRL\textsuperscript{15} (trained on CoNLL – Computational Natural Language Learning) corpora\textsuperscript{16}, used by Pal et al. [2010], SEMAFOR [Das et al., 2010] (trained on FrameNet), and ASSERT [Pradhan et al., 2004], (trained on PropBank) – used by Mannem et al. [2010] and Chen [2009].

Other useful tools are Tregex and Tsurgem [Levy and Andrew, 2006], that allow one to query and modify a parsing tree through regular expressions. Heilman [2011], Kalady et al. [2010], Wyse and Piwek [2009] use these in their work to transform the sentences' parsing trees into the desired questions.

\textsuperscript{15}http://www.surdeanu.info/mihai/swirl/index.php

\textsuperscript{16}We did not find any reference to the year of the corpus.
merges paraphrases representations in automata. Finally, the idea that Mazidi and Nielsen [2015] and Pazienza and Pennacchiotti [2005] propose of using different views to represent a sentence.

We also briefly described the available resources we have at our disposal, namely lexical resources, parsers and Semantic Role Labelers.
So far we discussed previous work done in Question Answering (QA) and Question Generation (QG) with pattern-based approaches. In this chapter we will introduce our solution to the QG system, that corresponds to the work towards Hypothesis 1.

From Chapter 2, the three main goals are: (a) to add stronger semantic features to patterns; (b) to not only learn new patterns, but also to be able to improve previously learnt patterns; and (c) to automatically evaluate the quality of the patterns created. How to learn these patterns is, thus, the key to the problem. The next sections will detail the proposed solutions for each of the aforementioned points, which can be seen aggregated in Figure 4.1.

Figure 4.1: Proposed solution for the pattern acquisition and validation process.
4.1 Towards Semantic Patterns

Our first goal is to introduce semantic features to typical patterns. This section will go through on how we intend to do so. Figure 4.2 exemplifies the different types of patterns that could be generated from a source sentence, and how they differentiate.

Figure 4.2: The different types of patterns by level: lexical, syntactic and semantic.

At the lexical level, the pattern is mostly composed by lexical units, with possible wildcards to allow some degree of flexibility. In the figure, the verb *bake* contains a wildcard to allow different formulations (*baked* vs. *bakes*). At the syntactic level we can see a generalization of the previous pattern. However, this generalization is too loose, allowing almost any match. Finally, at the semantic level, we have a more constrained pattern, but still with some degree of flexibility. *Apply Heat* is the frame extracted from FrameNet, *Agent* a semantic role from that frame, and *Food* and *Kitchen Appliance* two hypernyms, extracted from WordNet, from *cookies* and *oven*, respectively.

Notice, however, that although this is the usual order for the different levels in Natural Language Processing (NLP) (lexical, syntactic, semantic), this does not necessarily implies an order of generality for patterns. Actually, the levels are rearranged (lexical < semantic < syntactic), as syntactic patterns tend to be more generic than semantic ones.

Many systems in the past follow this linear view of a sentence, where tokens are replaced by their constituents or lexical lemmas, and sometimes Named Entitiess (NEs) tags, such as *Date* [Ali et al., 2010, Curto et al., 2011, Varga and Ha, 2010]. We want our patterns to be more dynamic, in the sense that all the information is there and the generation algorithm decides what to do with it later.
A pattern also needs to know how to create a question from a sentence, and to do so, it must have mapped the tokens from the Question/Answer seed pair. Figure 4.3 depicts this idea: the given pattern maps the tokens in the original question to the ones in the snippet, and contains two generic representations of the snippet. This follows the ideas of multiple sentence representation [Mazidi and Nielsen, 2014, Pazienza and Pennacchiotti, 2005, Tymoshenko and Moschitti, 2015].

Figure 4.3: A pattern representation for the Question/Answer seed *Where did John cook the lamb?* / *Red stove* and sentence *John cooked the lamb in the red stove.* The picture depicts the mapping between tokens and two generic representations of the sentence.

Independently of if we are creating questions or answering them, the system will need to match the input sentences with the available/necessary patterns. To exemplify how the pattern application can be performed, consider the following sentences:

1. *John cooked the lamb in the red stove.*
2. *John cooked the lamb in 5 minutes.*
3. *Marie baked cookies in the blue oven.*

and the question:

4. *Where did John cook the lamb?* / *Red stove*
Assuming the pattern in Figure 4.3 was created with the question/snippet pair Sentence 4/Sentence 1, given the two other sentences, this is what is intended to happen when trying to generate questions from them:

- Sentence 2 will not generate any questions, as, despite the semantic structure being similar, *in 5 minutes* does not represent where the lamb was cooked, but rather how quickly.

- Sentence 3 will generate the question *Where did Marie bake the cookies?/Blue oven*, as there is a structural match with the pattern seed sentence (there is an equivalent *who*, *what*, and *where*), despite lexically different.

### 4.1.1 Formalization

#### 4.1.1.1 Sentence Representation

Let $S$ be a sentence composed by a bag of tokens $T = t^1, \ldots, t^n$, and $|S| = n$ be the size of the sentence $S$. Each token $t$ is annotated with different lexical, morphological, and semantic information $F = \{f_1, \ldots, f_m\}$, where $f_i$ is one of those features.

$S$ as a whole can be represented in trees, such as constituent trees. Let, thus, $n$ be a node with $c$ children nodes, $n.c$, from $n^1$ to $n^c$, $n_t$ a special node representing the token $t \in T$, and $\hat{T}$ be the tree starting at the root node $n_r$. A constituent parser transforms a sentence $S$ into a tree $\hat{T}_c$ with $n_r = \text{ROOT}$, with $n_r$ and its children representing the syntactic structure of $S$, and the leaves of $\hat{T}_c$ being $n_{t^1}, \ldots, n_{t^n}$. A dependency parser transforms a sentence $S$ into a tree $\hat{T}_d$ with root node $n_r = n_{t^i} : t_i \in T$ (the head of the sentence), and with $n_{r.c}$ children being other nodes $n_t$, through labeled dependencies $d$. Therefore, children in trees $\hat{T}_d$ are indexed by $d$.

The function $\text{child}(n, index)$ gives the child node of $n$ indexed by $index$. In dependency trees $\hat{T}_d$, it is trivial that $index$ corresponds to the dependency labels $d$. For trees $\hat{T}_c$ representing the syntactic structure of a sentence from a constituent parser, $index$ can be, without loss of generality, the indices of the children of the node $n$, starting at 0, that is, $n^{index}$.

Finally, a subtree $ST$ is a tree which $n_r$ is any node in a tree $\hat{T}$. A tree has, thus, as many
4.1. TOWARDS SEMANTIC PATTERNS

Figure 4.4: Parse trees for the sentence *John cooked the lamb in the red stove* with Stanford syntactic parser (a), Stanford dependency parser (b), and MatePlus SRL (c).

subtrees as the number of nodes in it.

Figure 4.4 depicts the trees for sentence *John cooked the lamb in the red stove*. Constituent trees $\hat{T}_c$ are useful to grasp sequences of tokens $t$ that make sense together, like noun phrases (*the lamb* in the Figure 5.3a), whereas dependency trees $\hat{T}_d$ are useful to grasp word modifiers, as *red* in *red oven*, as seen the Figure 5.3b. However, the whole picture gets blended in the tree structure. SRLs provide yet another representation of a sentence, where chunks that represent a role are seen together.

Let $t_i \in T$ be a verb token in sentence $S$ and $A$ the set of arguments identified by the SRL for $t_i$. A predicate $PA$ is the tuple $\langle t_i, A, st(a) \rangle$ such that $t_i$ is the root of the predicate and each argument $a \in A$ is represented by the $ST_a = st(a)$.

4.1.1.2 Pattern

A pattern $P$ is the tuple $\langle S_s, PA, \text{align}(S_s, S_t) \rangle$, where $S_s$ and $S_t$ are the source and target sentences respectively, and $\text{align}(S_s, S_t)$ is the alignment of their tokens (see Section 4.2). This tuple contains all needed information to create a new sentence $S'$ of the same type as $S_s$; the alignments should tell how to transform the new unseen sentence $S'$ into $S_s$-like sentence, and $PA$ represents how similar $S'$ and $S_t$ are.
4.1.2 Implementation

4.1.2.1 Sentence Representation

We use Stanford constituent and dependency parsers [de Marneffe et al., 2006, Klein and Manning, 2003] to create trees $T_c$ and $T_d$, and Senna SRL [Collobert et al., 2011] to create predicates $PA$ (as seen previously in Figure 4.4). The subtrees $ST$ can be from either tree. If both trees contain subtrees that can represent the predicate’s argument $a$, two subtrees will be associated with the argument $a$ in the predicate $PA$. For example, from the same figure, the argument $AM LOC$ would be represented by two subtrees, one from each: PP (IN (in) (NP ( ... stove))) and stove $\rightarrow$ red, $\rightarrow$ the.

Each token $t$ is annotated with the following semantic features: $F = \{Ne, Wn, Vb, Pos, Emb\}$, with each feature $f$ being defined as follows:

**Named Entities** The feature $Ne$ represents NEs detected by Stanford Named Entity Recognition (NER) or regular expressions designed to extract dates. If a word, or multi-word expression, is detected as a NE, it will be collapsed as a single token. This feature represents the type of NE the token is.

**WordNet** The feature $Wn$ contains the synsets the token belongs to, given by WordNet [Miller, 1995].

**Verb Sense** The feature $Vb$ contains the frame and/or class the token is in, according to FrameNet [Baker et al., 2003] and VerbNet [Kipper et al., 2000], respectively, if it is a verb.

**Part of Speech** $Pos$ feature represents the Part of Speech (POS) tag attached to the token as it was parsed by the Stanford parser.

**Word Embedding** The feature $Emb$ is the word embedding vector that represents the token. The embeddings used are from Word2Vec [Mikolov et al., 2013].

4.1.2.2 Pattern

The implementation of the pattern does not require further explanation, as it is mainly a container of the tuple $<S_s, PA, align(S_s, S_t)>$. See Sections 4.2 and 4.3 for details on how the patterns are created and used, respectively.
4.2 Pattern Acquisition

To learn patterns such as the ones presented before, a set of Question/Answer seed pairs is used. Then, past the first interaction, these patterns can be improved. Figure 4.5 depicts this pattern acquisition step.

![Figure 4.5: Creation and improvement of patterns from Question/Answer seed pairs.](image)

The acquisition step will follow the ideas presented before, like Curto et al. [2011], Ravichandran and Hovy [2002], where seeds are used to generate new patterns. The difference, however, is that the patterns are semantic, as described in the previous section.

4.2.1 Formalization

4.2.1.1 Alignment of Seeds’ Components

Let $S$ be a sentence composed by a bag of tokens $T = t^1, \ldots, t^n$, $|S| = n$ be the size of the sentence $S$, and $equiv(t^i, t^j)$ be a function of equivalence over two tokens returning a value between 0 and 1 saying how much related the two tokens are.

The equivalence function is a composition $\Lambda$ of multiple different functions:

$$equiv(t^i, t^j) = \Lambda(e_1, \ldots, e_e).$$

Now let $S_1$ and $S_2$ be two sentences with some kind of relationship $r$. For instance, if $r$ is Paraphrase, both $S_1$ and $S_2$ would be two sentences conveying the same meaning. Another
example would be \( r = \text{Question To Sentence} \), where \( S_1 \) would be a question and \( S_2 \) would be a sentence that would answer the question \( S_1 \). To find the relation between two sentences, they must be aligned. An alignment of two sentences is a set of aligned tokens, and an alignment between two tokens is defined by the triple:

\[
\alpha(t^i, t^j) = <t^i, t^j, \text{equiv}(t^i, t^j)>
\]

To perform sentence alignment, we require that the content of one sentence must be included in the other. Without loss of generality, consider \(|S_1| \leq |S_2|\). The set of alignments \( \text{align}(S_1, S_2) = \{\alpha_1, \ldots, \alpha_n\} \) is a possible alignment for the two sentences, if \(|\text{align}(S_1, S_2)| = |S_1| \wedge \forall i \exists z : \alpha_z = <t^i_1, *, *>\). In other words, all tokens in \( S_1 \) are aligned with a token in \( S_2 \), and each belongs to one and one only alignment.

Finally, the chosen alignment \( \hat{\text{align}} \) over two sentences is given by:

\[
\hat{\text{align}}(S_1, S_2) = \arg \max_{\text{align}(S_1, S_2)} \sum_z \alpha_z(3) \in \text{align}(S_1, S_2),
\]

where \( \alpha_z(3) \) is the third entry of the tuple. In other words, we choose the alignment \( \text{align}(S_1, S_2) \) that maximizes the set of token alignments \( \alpha \) for the pair of sentences, according to the equivalence function.

### 4.2.2 Implementation

There are multiple decision points when it comes to implement the algorithm in Section 4.2.1.1: 1) how to create the bag of tokens \( T \) for a given sentence \( S \); 2) the definition of function \( \text{equiv}(t^i, t^j) \); 3) the composition function \( \Lambda \); and 4) the selection of the alignment \( \hat{\text{align}} \) over two sentences \( S_1 \) and \( S_2 \), given the possible \( \text{align}(S_1, S_2) \). Each one of these is discussed in the following subsections.

#### 4.2.2.1 Bag of tokens for a sentence

There are two major decisions to take into consideration at this step. First is to determine what tokens should be considered for alignment. For instance, some, if not all, stopwords could
4.2. PATTERN ACQUISITION

be removed from the bag $T$, like the or on. The second decision concerns sequences of words that can (or should) be treated as a single token. A good example is a person name as Barack Obama.

In our implementation, we are removing all stopwords from the bag $T$, and condensing into a single token both dates, extracted by a set of regular expressions, and NEs, recognized by the Stanford NER.

4.2.2.2 equiv function

The $equiv(t^i, t^j)$ function aims at comparing two tokens and returning a value that says how related they are, from 0 (unrelated), to 1 (totally related/identical). It is a combination of multiple functions (see Section 4.2.2.3).

In our implementation we developed five functions, each using different characteristics a token might have. As of now, all values were manually chosen.

**Lexical**  The first one, $equiv_L$, is a lexical comparison of the two tokens:

$$equiv_L(t^i, t^j) = \Lambda \left( f, \frac{t^i \equiv t^j, \text{lemma}(t^i) \equiv \text{lemma}(t^j)}{g} \right)$$

$$f = \begin{cases} 1 & \text{if } t^i = t^j \\ 0 & \text{otherwise} \end{cases} \quad g = \begin{cases} 0.75 & \text{if } \text{lemma}(t^i) = \text{lemma}(t^j) \\ 0 & \text{otherwise} \end{cases}$$

**Named Entities**  $equiv_{NE}$ compares their content, if both tokens are NEs categorized as dates, persons, or organizations. Dates are extracted by a set of regular expressions, and other NEs are recognized by the Stanford NER.

$$equiv_{NE}(t^i, t^j) = \begin{cases} 1 & \text{if } t^i = t^j \\ 0.9 & \text{includes}(t^i, t^j) \\ 0 & \text{otherwise} \end{cases}$$
includes\((t^i, t^j)\) is a function that uses a set of rules to determine if two tokens are referring to the same entity, or if two tokens represent the same date. For instance, both Obama and \textsc{D01 M01 Y2014} are included in Barack Obama and \textsc{M01 Y2014}, respectively.

**Verb sense**  
\(\text{equiv}_{VB}\) compares the sense of two tokens, if they are both verbs and are related according to SemLink\(^1\). This resource is a mapping between PropBank Palmer et al. [2005], VerbNet and FrameNet. If the two tokens belong to the same set in any of the resources, they are considered to match.

\[
edquiv_{VB}(t^i, t^j) = \begin{cases} 
0.75 & \text{if } \text{sense}(t^i) = \text{sense}(t^j) \\
0 & \text{otherwise}
\end{cases}
\]

For example, make and build both belong to the frame \textbf{Building} of FrameNet, which would make the function return 0.75 for those tokens.

**WordNet**  
\(\text{equiv}_{WN}\) matches two tokens if their path distance is below a manually defined threshold. We compute the path distance by traversing the synsets upwards until finding the least common subsumer [Resnik, 1995]. For each node up, a decrement of 0.1 is awarded, starting at 1.0.

\[
edquiv_{WN}(t^i, t^j) = \begin{cases} 
1 & \text{if } \text{syn}(t^i) = \text{syn}(t^j) \\
x & \text{if } \text{syn} (\text{hyp}(t^i)) \supset \text{hyp}(t^j) \\
x & \text{if } \text{hyp}(t^i) \subset \text{syn} (\text{hyp}(t^j)) \\
0 & \text{otherwise},
\end{cases}
\]

where \(\text{syn}(t)\) gives the synset of the token \(t\), \(\text{hyp}(t)\) gives the hypernyms of \(t\), and \(x = 1 - \max(n \times 0.1, m \times 0.1)\), with \(n\) and \(m\) being the number of nodes traversed in the synsets of \(t^i\) and \(t^j\) respectively. If no concrete common subsumer is found, then 0 is the result returned.

For example, feline and cat have the common synset \textbf{feline}, one node above cat, thus returning \(1 - 0.1 = 0.9\). Dog and cat result in \(1 - 0.2\), as one needs to go up two nodes in both tokens to find the common synset \textbf{carnivore}. We do not go up to generic synsets, like

\(^{1}\)http://verbs.colorado.edu/semlink
4.2. PATTERN ACQUISITION

**artifact** or **item**.

**Word2Vec** $equiv_{W2V}$ computes the cosine similarity between the vectors representing the two tokens $t^i$ and $t^j$:

$$equiv_{W2V}(t^i, t^j) = \cos(W2V(t^i), W2V(t^j)),$$

where $W2V(t)$ is the vector representing the word embedding for the token $t$. We use the Google News word2vec models available [Mikolov et al., 2013]². If the token is composed by more than one word (in the case of a NE for example), their vectors are added before computing the cosine similarity.

For example, car and vehicle obtain a cosine similarity of 0.78, while car and New York result in a score of 0.07.

### 4.2.2.3 Λ function

The Λ function, shown first in Equation 4.1, is a composition function that takes multiple functions with identical range as input and outputs a single value in that same range as final result. Many definitions may be used, from the simple average $\Lambda(f, g, \ldots, k) = \text{avg}(f, g, \ldots, k)$, to more complex combinatorial functions.

Our implementation of Λ uses the different functions available by the order listed in Section 4.2.2.2, returning the first non-zero value. More concretely, we would only use $equiv_{V_B}$ if $equiv_{L}$ returned 0, and so on.

### 4.2.2.4 Selection of final alignment $\hat{\text{align}}$

Choosing the best set of alignments is similar to the assignment problem, in which one is aiming at optimizing an utility function over a set of assignments – usually one tries to minimize the cost of doing $X$ jobs by using $X$ workers, each with a known hourly rate for each job. In our case we intend to pick the best pairwise token alignments $\alpha$ that maximize the overall quality of the alignment $\text{align}(S_1, S_2)$, given that we only use each token of a given sentence once.

²https://code.google.com/archive/p/word2vec/
The Hungarian algorithm [Kuhn, 1955] is a combinatorial optimization algorithm designed to solve the assignment problem. The adaptation to our problem is as follows: Let $M$ be a matrix of dimension $|S_1| \times |S_2|$, where $M_{ij}$ is the value of $equiv(t^i, t^j)$. The assignment problem usually takes an equal number of jobs and workers, but an adaptation is possible by adding the necessary dummy lines/columns if the matrix is not square. The original problem is also trying to minimize the utility function, while we are trying to maximize the value of the overall alignment. To make for this adaptation, we convert the values to have a minimization problem instead, replacing each cell $M_{ij}$ by $max - M_{ij}$, where $max$ is the maximum value present in the whole matrix.

### 4.2.2.5 Pattern Creation

For each $PA \in S$, the system checks the tokens used in the chosen alignment $\text{align}(S_s, S_t)$. If a subtree $ST \in PA$ contains tokens not present in the alignment, $\forall t \in ST \nexists \ z \mid t \in \alpha_z$, the whole argument that contains that token is removed. If, by doing so, tokens in the chosen alignment are also removed, the predicate $PA$ is discarded.

Predicates not discarded capture, thus, perfectly the context in which the aligned tokens are in the sentence. A pattern $P$ is created containing $S_s$, $PA$ and $\text{align}(S_s, S_t)$.

### 4.3 Pattern Application

This step takes patterns previously learned and applies them to new unseen sentences. The current task in hands dictates how the pattern is used, but the concept for its application is the same regardless of the task.

#### 4.3.1 Formalization

Let $P$ be a pattern as previously defined and $S'$ the new unseen sentence. A match between $S'$ and $S_t$ in the pattern $P$ is made through their predicates $PAs$. A match is issued if and only if:

1. $equiv(\text{root}_P, \text{root}_{S'}) \geq 0$, where $\text{root}_P$ and $\text{root}_{S'}$ are the roots of the predicates, $PAs_P$ and $PAs_{S'}$, respectively;
4.3. PATTERN APPLICATION

2. $\forall a \in A \subset PA_S' : \text{match}(ST_{S'}^a, ST_p^a) = true,$

where $\text{match}(\hat{T}_1, \hat{T}_2)$ is a function over two trees, returning a boolean regarding the equivalence of those trees. One tree is equivalent to another if they share the same structure and their leaves are equivalent according to $\text{equiv}$ function.

If any condition (1 or 2) is not met, there is no match between $S'$ and $S_t$ and, thus, $S'$ cannot be transformed into a sentence of the $S_s$ form.

If a match is issued, then $S'$ is transformed into a new $S'_s$ by replacing the tokens in $S_s$ with the new tokens from $S'$.

4.3.2 Implementation

The implementation of the pattern application has few decision points. The first is how to define $\text{match}(\hat{T}_1, \hat{T}_2)$. The second is how to transform $S'$ into a new $S'_s$ if a match is issued. The following sections describe each of the two points.

4.3.2.1 $\text{match}(\hat{T}_1, \hat{T}_2)$ function

This function compares two trees and returns a true/false value regarding their equivalence. We are talking about a structural comparison mostly, but the tokens in the leaves are important as well. Algorithm 1 details the process of matching, where $n.c$ represents the children of node $n$. It starts by comparing the roots of the two trees (Line 5) and, if equivalent – using the same $\text{equiv}$ function presented in Section 4.2 – the algorithm recursively matches their children (Line 13). An additional condition was added to make sure the trees are identical, in Line 10, which requires the number of children at every node to be the same. If the two trees are successfully matched recursively through their entire structure, the alignments between the two trees are returned, collected during its execution (Lines 8 and 17).

4.3.2.2 Extending $\text{equiv}$ function

We extend the function presented in Section 4.2 to include the following matching options:

**Named Entities** This one is not an addition, but a modification to $\text{equiv}_{NE}$. The previous version required that the entities were equivalent, which makes sense for an alignment task.
Algorithm 1 Algorithm for tree matching.

1: $match(T_1, T_2)$
2: $align \leftarrow []$
3: $n_1 = T_1.root$
4: $n_2 = T_2.root$
5: if $equiv(n_1, n_2) \leq 0$ then
6:   return false
7: else
8:   $align \leftarrow \alpha(n_1, n_2)$
9: end if
10: if $|n_1.c| \neq |n_2.c|$ then
11:   return false
12: end if
13: for $i : 0 .. |n_1.c|$ do
14:   if $match(n^i_1, n^i_2) = false$ then
15:     return false
16:   else
17:     $align \leftarrow match(n^i_1, n^i_2)$
18:   end if
19: end for
20: return $align$

However, when trying to generate new items from unseen text, we cannot put such a constraint on the process. Actually, any NE should be able to fill that slot on the pattern, as long as it shares the same type. The new version reflects this approach:

$$equiv_{NE}(t^i, t^j) = \begin{cases} 
1 & \text{if } t^i.type = t^j.type \\
0 & \text{otherwise}
\end{cases}$$

**Syntactic Noun** This function tries to relax the matching process, making nouns to be matched independently of their semantic meaning. This can introduce much noise to the process, but it can also widen the generation task by putting less restrictions into the matching process.
4.4. VALIDATION

\[
equiv_{\text{Sym}N}(t^i, t^j) =\begin{cases} 
1 & \text{if } t^i.\text{Pos} = t^j.\text{Pos} = \text{Noun} \\
0 & \text{otherwise}
\end{cases}
\]

**Syntactic Verb**  Identical to the previous one, but tailored for verbs only.

\[
equiv_{\text{Sym}V}(t^i, t^j) =\begin{cases} 
1 & \text{if } t^i.\text{Pos} = t^j.\text{Pos} = \text{Verb} \\
0 & \text{otherwise}
\end{cases}
\]

4.3.2.3 Generation

If the two predicates \( PA_P \) and \( PA_{S'} \) are matched, then the new sentence \( S' \) is structurally and semantically identical to the sentence \( S_t \) in the pattern. The transformation of it into a sentence of the type \( S_s \) is done with help of the alignments \( \text{align} \) returned by the various \( \text{match} \) function calls for Condition 2 of Section 4.3.1. Each token aligned from \( S' \) to \( S_t \) which can be mapped to \( S_s \) in the pattern \( P \) will replace that token in \( S_s \):

\[
\exists \alpha(t^i, t^j) \in \text{align}(S_s, S_t) \land \exists \alpha(t^j, t^k) \in \text{align} \implies \text{replace}(t^i, t^k)
\]

In other words, each token \( t^k \in S' \) that was aligned with a token \( t^j \in S_t \) that is mapped to a token \( t^i \) in the original sentence \( S_s \) will take its place in the final new sentence. This means, thus, that non mapped tokens in \( S_s \) will remain. For example, wh-words in questions will not be mapped to tokens in \( S_t \), which will be kept in the final new question, providing the same type of question.

4.4 Validation

This section is divided into two parts: the first corresponds to the validation of the generated Question/Answer pairs to be used by the system in a new pattern acquisition step. The second discusses the evaluation of the patterns used by the system. This part of
the work borrows ideas from works like Mendes [2013], Pantel and Pennacchiotti [2006] and Shima [2015].

4.4.1 Question/Answer Pair Validation

A bootstrap step to learn the patterns from a set of Question/Answer seed pairs, like TheMentor [Curto et al., 2011] does, is often employed. However, past interactions are usually disregarded as a source of improvement. All new generated questions with previous patterns can be, thus, used to augment the pool of available Question/Answer pairs. Figure 4.6 depicts this part only.

![Figure 4.6: Creation of new Question/Answer pairs to be used as seeds in future iterations.](image)

Although this can be done with no quality control at all, it might be useful to guarantee the correctness of the generated Question/Answer pairs before adding them to the new set of seeds to be used. These are either presented to the user, who assesses them, or compared against a goldstandard\(^3\). Although the costs of having a human rating the system’s output are high, this might be necessary, as there is no right question to ask about a given text or sentence. However, between different runs, we can store all user assessments to create a goldstandard that the system can use in future runs, minimizing this way the need of human interaction. Given this feedback, the system gets to know what questions were correctly generated and, thus, are a good source for a new pattern acquisition step.

\(^3\)The picture only denotes a user as source of feedback.
4.4. VALIDATION

4.4.2 Pattern Evaluation

Despite the previous feedback may be used, indirectly, to assess which patterns are performing better, it is important to evaluate the patterns in a more direct way. The idea is that, with a proper evaluation, bad patterns will be not used to generate bad questions in the future, reducing the likelihood of adding erroneous Question/Answer pairs to the seed pool. Figure 4.7 focus on this evaluation process.

Figure 4.7: Evaluation of the created patterns through Question Answering.

The system will be able to evaluate the patterns by performing QA. As the patterns can be used to either generate questions or extract answers, the system may validate itself by automatically extracting answers to a few questions and compare them with the expected answers. A correct match reinforces the quality of the pattern, which can be awarded a positive score, while incorrect answers extracted by a pattern will damage its value. Notice that these questions can either be the ones generated before or from another dataset.

4.4.2.1 Formalization

Let $x$ be the score of a pattern $P$, that is, the tuple is extended to include its score $x: <S_s, PA, \text{align}(S_s, S_t), x>$. This score can determined when the pattern is created, in function of the alignments, or adjusted afterwards, in function of the generation task that used the pattern:

$$x^{t+1} = \text{update}(x^t, S'),$$
where \textit{update} is a function that takes a score $x^t$ and a new generated result $S'$ to compute a new score $x^{t+1}$.

### 4.4.2.2 Implementation

The initial score $x^0$ can be computed in function of the alignments found:

$$x^0 = \Lambda'(\forall \alpha(3) \in \text{align}(S_s, S_t)),$$

where the combinatorial function $\Lambda'$ computes the final score based on the scores of each individual alignment $\alpha$ ($\alpha(3)$ is the third entry of the tuple, that is, the score obtained from $\text{equiv}(t^i, t^j)$).

This module is not fully implemented yet, but the patterns already support a score $x$ and the $\text{equiv}$ function already gives a score for its matches. Options for $\Lambda'$ (note that this function is different from the one presented in Section 4.2, but follows the same principles) range from simple techniques like the average or sum of all scores, to more sophisticated ones.

The $\text{update}$ function requires two things: a quantitative evaluation of the new sentence $S'$ and a way to incorporate that into $x^t$.

### 4.5 Pattern Improvement and Merging

The evaluation and validation of the patterns is important, but we would like to incorporate the remaining steps of example-based systems described in Section 3.4, namely \textbf{Revise} and \textbf{Retain}. The goal is that we could be able to generalize from seeing many similar examples (patterns) and draw conclusions from those, like what types of verbs could be used, or how far in a subtree we should look for a match. This idea was based on works like Pang et al. [2003] and Velardi et al. [2013]. This module is not implemented yet.
In this chapter we go through the different tasks previously enumerated and how the system can be evaluated in each scenario. These results correspond to the last chapter only, that is, they only cover Hypothesis 1.

5.1 Pattern Acquisition

The evaluation of the Pattern Acquisition step is closely related with the alignment task we perform, as described in the last chapter. This section describes the evaluation of this step as follows: first we discuss the dataset to be used, then we describe the metrics used in our evaluation, and finally we present the results.

5.1.1 Datasets

There are many datasets closely related with the alignment and/or paraphrase detection task, but just a few are monolingual (namely in english, which is the language we are working on).

First there is Microsoft Research Paraphrase Corpus (MSRP)\(^1\) which is specific for paraphrases [Dolan et al., 2004]. It contains a training set of 4076 sentence pairs (67.5% of which are positive examples) and a test set of 1725 sentence pairs (66.5% positive). However, the corpus does not contain any indication of how the sentences align and, thus, not suitable for our experiments.

Secondly, SemEval 2016 [Agirre et al., 2016] provided a dataset for the Interpretable Semantic Textual Similarity (iSTS) task, containing two corpora\(^2\). One is a set of 756 sentence pairs of news headlines, and the other is a set of 750 sentence pairs of image captions. However,

---

\(^2\)http://alt.qcri.org/semeval2016/task2/
it does not include paraphrases only, but also sentence pairs that capture contradictions and other linguistic phenomena, which we are not interested in, and since they are not annotated as such, it becomes impracticable to use.

Another available corpus is the manually aligned RTE 2006 corpus, provided by Microsoft Research [Brockett, 2007]. It contains 2400 sentence pairs, both positive and negative, where positive examples imply an entailment relationship between the pair. However, the negative examples were not annotated as such, making this dataset hard to use as well.

Finally, there is the Edinburgh dataset [Cohn et al., 2008], which is fully a set of paraphrases pairs, manually aligned. It contains a training set of 714 sentence pairs and a test set of 306 pairs, with all pairs being positive examples.

Of the four, the Edinburgh dataset is the one that fulfills our needs, which we use in our experiments.

5.1.1.1 Edinburgh Corpus

We detected some lack of consistency across the dataset, namely on how multiword Named Entities (NEs) were aligned. For instance, there are examples of East Timor being aligned as a unique token and as two separate alignments. In addition, the corpus captures long-distance alignments of token sequences that we are not aiming at capturing in our specific task. For example, in the paraphrase pair Fujian’s gross national product... / the gross national product of..., there is an alignment between ’s and the .. of. While technically correct (’s is indeed referring to a property of something), this is the kind of alignment we are not interested in and are not performing. To cope with these, we modified and extended (automatically at first, and manually afterwards) the corpus to contain multiple possible alignments for NEs and removed uninformative alignments such as the one showed. We performed these alterations on the training set only, which we use in our experiments. An example for the second entry of the corpus is in Figure 5.1; note how it is acceptable to align Doyle in the first sentence with either the full name or just the strict identical name. The modified corpus is available.

\[4\]http://www.ling.ohio-state.edu/~mwhite/data/coling12/edinburgh-json-20130322.tgz
5.1. PATTERN ACQUISITION

online\(^5\).

\[
\text{Figure 5.1: Part of the xml specification for instance “news-common:2143” from the training set – Hackett and Rossignol did not know each other and Hackett had no connection to Colby, Doyle said. / State police Lt. Timothy Doyle said Hackett and Rossignol did not know each other, and that Hackett had no connection to the college.}
\]

In Figure 5.2 are shown the number of alignments in each instance of the corpus, and the number of instances that contain a certain number of alignments. The graphs on the left report numbers after removing stopwords. As one can see, the corpus resembles a normal distribution, excepting a few outliers, which are understandable as natural language is not linear. This phenomena is more evident when keeping stopwords, which can vary significantly in number from one instance to another. The graphs also have a longer tail to the right side, showing this variability as well. It is noticeable how condensed the figures are on the left

\(^{http://l2f.inesc-id.pt/~hpr/edinburghModified/gold.train.alignments.fixed.xml}\)
side, without considering stopwords. The instance with greatest number of alignments drops from 46 alignments to 24, whereas the number of instances with the most common number of alignments almost duplicates. This clearly shows that, without considering stopwords, the content of the sentences is much more similar regarding the number of alignments required. This is also observable on the bottom half of the figure, where the ladder effect changes its shape: it’s less taller and each step is longer. Finally, it is important to note that, after removing stopwords, a few instances lack alignments, as they are short and lack content words; for instance “well, why should there be any more?” is a sentence including only tokens present in the stopword list, resulting in an empty sentence, content-wise.

Figure 5.2: Graphical analysis of the Edinburgh corpus. Above, the figures show the number of instances in the corpus with $x$ number of alignments (5.2a and 5.2b). Below it is depicted the number of alignments per each instance in the corpus, sorted increasingly (5.2c and 5.2d). The left side does not account for stopwords (5.2a and 5.2c), while the right side keeps them.
5.1. PATTERN ACQUISITION

5.1.2 Evaluation Metrics

To evaluate the alignments we use precision, recall and F1 measure, defined as follows:

\[
\text{precision} = \frac{tp}{tp + fp},
\]

\[
\text{recall} = \frac{tp}{tp + fn},
\]

\[
F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}},
\]

where \(tp, fp\) and \(fn\) are true positives, false positives and false negatives, respectively.

Given a evaluation measure \(B(tp, fp, fn)\) calculated in function of true positives, false positives and false negatives, there are two ways of calculating \(B\) across multiple datasets \(d \in D\): \(B_{\text{micro}}\) and \(B_{\text{macro}}\):

\[
B_{\text{micro}} = B(\sum_d tp_d, \sum_d fp_d, \sum_d fn_d)
\]

\[
B_{\text{macro}} = \frac{1}{|D|} \sum_d B(tp_d, fp_d, fn_d),
\]

where \(tp_d\) is the number of true positives for dataset \(d\) (likewise for \(fp_d\) and \(fn_d\)).

\(B_{\text{macro}}\) averages each individual \(B_d\), meaning each dataset has an equal weight on the final score, whereas \(B_{\text{micro}}\) averages the overall counts, which makes larger datasets \(d\) have more weight, dominating smaller datasets.

In our concrete case, we have a single dataset, but each instance in the dataset contains multiple alignments, which means \(B\) can be calculated for each instance. In other words, each instance on our dataset can be treated as a dataset \(d\) in the equation above.

Each instance has a number of alignments that is function of the sentences’ length. However, the difficulty of aligning two sentences, although growing with their size, is not dictated by that factor alone. Thus, it would not make sense to give more weight to \(B_s\) calculated for those larger instances. For this reason, we chose to macro-average the results in this
experiment [Van Asch, 2013]:

\[
\text{precision}_{\text{macro}} = \frac{1}{|D|} \sum_i \text{precision}(i),
\]

\[
\text{recall}_{\text{macro}} = \frac{1}{|D|} \sum_i \text{recall}(i),
\]

where \( |D| \) is the size of our dataset and \( i \) is an instance on \( D \). \( tp \) is the number of alignments found which are present in the goldstandard, \( fp \) is the number of alignments found which do not belong to the goldstandard, and \( fn \) is the number of alignments not found which are present in the goldstandard. Precision and recall can be thus rewritten as:

\[
\text{precision} = \frac{|\text{gold} \cap \text{alignments}|}{|\text{alignments}|},
\]

\[
\text{recall} = \frac{|\text{gold} \cap \text{alignments}|}{|\text{gold}|},
\]

where \( \text{gold} \) is the set of alignments from the goldstandard and \( \text{alignments} \) is the set of alignments found by the system being evaluated.

5.1.3 Experimental Setup

We use the modified Edinburgh training set in our experiments. Our alignment system, built as specified in Section 4.2, is used and compared against GIZA++ [Och and Ney, 2003]?6, which implements the IBM-4 alignment model, and Meteor aligner module [Denkowski and Lavie, 2014]?7. We calculate precision, recall and F1 as previously mentioned for both systems.

5.1.4 Experimental Results

We created 6 different configurations: one for each of the \( \text{equiv} \) functions in Section 4.2.2.2, and a last one using them all combined as described in Section 4.2.2.3. We also divided these in two runs: one where we remove stopwords, and another were no stopwords were removed at all. As Figure 5.1 shows, the goldstandard includes stopwords, which were not accounted
when calculating the system’s performance for the first runs, and punctuation, which were discarded overall.

The statistics of the alignments previous to the final selection performed by the Hungarian algorithm are not very informative as a whole, but they still provide some insight. For instance, if an alignment in the goldstandard has $|S_1|$ alignments, two of which are verbs, the function $equiv_{V_B}$ will at most find 2 alignments, which results in a really low recall. On the other hand, $equiv_{W2V}$ is able to generate an alignment for almost every pair of words, which means the precision will be really low and recall really high. Therefore, the pre-Hungarian runs are only useful to assess the recall of $equiv_{W2V}$ and the precision of generic functions, as $equiv_L$ and $equiv_{WN}$. Results over all 714 pairs are presented in Table 5.1 – instances with no alignments found have precision 1 and recall 0.

<table>
<thead>
<tr>
<th>Stopwords Removed</th>
<th>With Stopwords</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
</tr>
<tr>
<td>Lexical</td>
<td>0.90 ± 0.17</td>
</tr>
<tr>
<td>WN</td>
<td>0.58 ± 0.28</td>
</tr>
<tr>
<td>W2V</td>
<td>0.17 ± 0.25</td>
</tr>
<tr>
<td>All</td>
<td>0.17 ± 0.25</td>
</tr>
</tbody>
</table>

Table 5.1: Average precision, recall and F-measure for all runs, considering all matches found.

As expected, $equiv_L$ does not obtain perfect precision because some alignments are required between non-identical tokens. On the other hand, recall is not perfect as well for $equiv_{W2V}$ because sometimes multiple-token to one token alignments are required, and our system is not able to capture them (for instance, said in an interview should be aligned to told – one could argue that said could be an alignment hypothesis for told, but we have not extended the corpus for this cases).

However, the real important results are the ones obtained after the final selection, that is, pos-Hungarian algorithm application. These are shown in Table 5.2. For $equiv$ functions with small coverage, like $equiv_{V_B}$, no final solution is found, as it would be expected. For other functions with more applicability, the set of alignments will be much smaller than before, impacting this way the precision score (which should go up), and recall (which should go
down). For instance, for a given $S_1$ sentence, $equiv_{W2V}$ will have at most $|S_1|$ alignments, instead of $|S_1||S_2|$ as pre-application of the Hungarian algorithm; therefore, it might not cover all gold alignments, but precision will improve dramatically.

<table>
<thead>
<tr>
<th>Stopwords Removed</th>
<th>With Stopwords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>Lexical</td>
<td>0.99 ± 0.08</td>
</tr>
<tr>
<td>WN</td>
<td>0.97 ± 0.14</td>
</tr>
<tr>
<td>W2V</td>
<td>0.85 ± 0.24</td>
</tr>
<tr>
<td>All</td>
<td>0.82 ± 0.25</td>
</tr>
</tbody>
</table>

Table 5.2: Average macro-precision, recall and F-measure post Hungarian Algorithm application, along with the number of pairs with a solution found.

<table>
<thead>
<tr>
<th>Stopwords Removed</th>
<th>With Stopwords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>Lexical</td>
<td>0.93 ± 0.21</td>
</tr>
<tr>
<td>WN</td>
<td>0.83 ± 0.31</td>
</tr>
<tr>
<td>W2V</td>
<td>0.74 ± 0.26</td>
</tr>
<tr>
<td>All</td>
<td>0.72 ± 0.26</td>
</tr>
</tbody>
</table>

Table 5.3: Average macro-precision, recall and F-measure for the instances with solutions found with Hungarian Algorithm

As one can see, $equiv_L$, which had really good scores (namely recall and F1), dropped significantly because it is not able to find a solution by itself for the alignments, obtaining only 90 and 61 solutions for both runs (although with perfect precision for 73 and 32 of them, respectively). Results are also overall worse for runs that do not remove stopwords, because they require tokens to be matched that are not necessarily present on the paired sentence more frequently.

Table 5.3 shows the results for only the instances to which a solution was found. F1 values go up significantly, meaning the obtained alignments are of good quality. Regarding the $equiv$ function as a whole, we can see that adding semantic functions provides a good tool to be able to align sentences that would not be aligned just through lexical alignment strategies. When looking only to the pairs to which a solution was found, one can see the differences are substantial: removing stopwords increases the number of solutions found (the
5.1. *PATTERN ACQUISITION*

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIZA++</td>
<td>0.55 ± 0.19</td>
<td>0.65 ± 0.21</td>
<td>0.60</td>
</tr>
<tr>
<td>Meteor</td>
<td>0.69 ± 0.17</td>
<td>0.78 ± 0.17</td>
<td>0.73</td>
</tr>
<tr>
<td>All \Stopwords</td>
<td>0.73 ± 0.26</td>
<td>0.67 ± 0.30</td>
<td>0.70</td>
</tr>
<tr>
<td>All w/ Stopwords</td>
<td>0.68 ± 0.21</td>
<td>0.72 ± 0.26</td>
<td>0.70</td>
</tr>
</tbody>
</table>

Table 5.4: Average precision, recall and F-measure for GIZA++ and Meteor on the modified Edinburgh training corpus.

precision and recall cost seems small). Given the conclusions from the previous paragraph, we imagine, however, that more errors are being introduced in the second run, but being masked by the large number of correct but unimportant alignments of stopwords. Regarding the drop for *equiv* \text{W2V}, we believe the reason is that this function is able to pair any two tokens, even if with a low score, meaning a match for missing tokens is still produced, despite being incorrect, which lowers its performance.

Table 5.4 report the results obtained with GIZA++ and Meteor aligner. As one can see, the results are significantly better than ours if we consider the whole picture, but are on par with the results for pairs we find a solution to. However, that is a unfair comparison – whichever we use. The table shows too, for that reason, the results obtained by our system if no restriction was imposed by to the Hungarian Algorithm, that is, if we do not required all terms on the shortest sentence to have a match. As one can see, the results also improve significantly when compared with the ones from Table 5.2, while precision and recall suffer opposite trends, which is expected.

Upon further analysis, we found that many pairs, extracted from novels, contained a couple of dialogues. This introduces noise for the segmenter, which ends up treating instances on some pairs separately. What this means, in the end, is that our system ends up with less content for some pairs, trying its best to align the corrupted sentences, damaging this way both precision and recall. However, this is something we will find with real text, which means these errors will always exist. Because it is not our goal to compete in alignment tasks but, rather, to prove that our alignment module is suitable for our task, we decided to not tamper with the evaluation done. We believe, though, that our results could be even better than
CHAPTER 5. INITIAL RESULTS

presented, if those instances were dealt with.

5.2 Pattern Application

5.2.1 Question Generation

5.2.1.1 Datasets

As discussed before in this document, Question Generation (QG) is a tough task to evaluate, mainly because it comprehends a generation task which evaluation can be subjective and because it is virtually impossible to automate the evaluation process by listing all possible questions to be asked about a given topic or even passage. Here follow a few datasets that could be used in our QG task.

The first QG competition, Question Generation Shared Task & Evaluation Challenge (QGSTEC), provided both a small corpus of development and test. The development set contains 81 sentences associated with a few questions (typically from one to four questions), while the test set provides 90 sentences from which systems were supposed to generate questions of the indicated types. This corpus can be an interesting source of seed pairs for the Pattern Acquisition step, as it contains both support sentence and question pairs; however, it does not include answers to those questions.

Engarte is a corpus from the Answer Validation Exercise (AVE). This corpus is composed of question-answer-snippet triples. Systems competing in AVE must label those triples as “true” or “false”, depending if the answer can or cannot be derived from the given snippet. This corpus is closely related with the task of Question Answering (QA), as the idea behind it is to equip systems with the means to understand if a passage contains the correct answer to the given question. However, the answer slot is typically filled with an entire passage, instead of the actual answer.

QA corpora is also interesting in the sense that they can be a source of many Q/A pairs. However, typically they do not provide the support text, making them hard to use (either to create patterns or to automatically evaluate new generated questions).

8https://github.com/bjwyse/QGSTEC2010
9http://nlp.uned.es/clef-qa/repository/ave.php
Many systems also use Wikipedia articles in their evaluation as a source of text to generate questions from. It is openly available to the community, and documents of any topic can be used. This means, however, that the evaluation is processed manually (which is the case for most systems anyway).

Smith et al. [2008] published the results the students obtained in a QG course project – Question Answer Dataset (QAD). It includes three datasets, one for each year, partitioned in four sets of topics, each containing ten documents. Those documents are cleaned wikipedia articles, and are indexed by a single document containing all questions generated by student’s systems; each question has also answers and a difficulty metric, measured by the annotators. Many questions do not have answer attached, and many more are of the type yes/no, thus lacking variability. However, the documents can be a good benchmark for systems to use, as the wikipedia articles are cleaned already and ready to use, increasing this way reproducibility of experiments.

More recently Stanford published a large dataset, SQuAD [Rajpurkar et al., 2016][10], also based on Wikipedia articles. It contains over 100,000 questions across over 500 Wikipedia articles, divided in isolated paragraphs and Q/A pairs associated with those.

**Document Used** In our experiments we are looking for intuitions that could lead us in the future, rather than generating a big amount of questions. Also, because it is so costly to evaluate a large amount of items, we opted for using a single document. However, it is our expectation that the final experiments conducted by the end of this thesis contain a much larger number of documents and topics.

We used a single document from the QAD [Smith et al., 2008], from 2008 class: s4a1. This is an article about Isaac Newton, containing 346 sentences, with an average of 15 tokens, the longest having 86.

**5.2.1.2 Evaluation Metrics**

Because it is a natural language generation problem, it is virtually impossible to create a goldstandard containing all the correct formulations of all acceptable questions. This means

---

the evaluation must be done manually, and it is dependent of the evaluators subjectivity. This requires multiple raters, that manually go through the generated questions and evaluate them respecting a few guidelines. The agreement between the raters is calculated and can be used as an indicator of the quality of the generated questions. One method used in such cases is the Cohen’s Kappa agreement [Cohen, 1960], given by the following formula:

\[ \kappa = \frac{p_o - p_e}{1 - p_e}, \]

where \( p_o \) is the observed agreement between the raters, and \( p_e \) is the probability of agreement by chance, calculated with the observed data.

From state of the art, there are a few guidelines we could use:

- Curto et al. [2012] classify questions as either perfectly plausible (Pp), plausible with anaphora (Pa), plausible needing context (Pc), implausible due to incompleteness (Im), and implausible (I);
- Heilman [2011] classifies questions as acceptable or not acceptable. Un acceptable questions are further broke down into ungrammatical, vague, and semantically incorrect;
- Lindberg et al. [2013] uses a binary judgement on the following parameters: grammar, semantic validity, vagueness, and answerability.

We chose to draw metrics from both Heilman [2011] and Lindberg et al. [2013], although many can be mapped to Curto et al. [2012]’s guidelines. The following section will discuss them.

It is important to note that we will also plan save all acceptable questions that are generated throughout development, in order to build a goldstandard that can be used for quick evaluations and, thus, provide a good indicator of the system’s performance during development.

Finally, Shima [2015] introduces a new evaluation metric, DIMPLE (DIversity-aware Metric for Pattern Learning Experiments), that aims at assessing the quality of paraphrase pairs while taking into account the lexical diversity. It is given by:
5.2. PATTERN APPLICATION

\[
DIMPLE_k = \frac{\sum_{i=1}^{k} 2^{Q_i D_i} - 1}{Z},
\]

where \( Q \) and \( D \) are functions that quantify the quality of the paraphrase and its lexical diversity, respectively, and \( Z \) a normalizing factor based on the maximum values \( Q \) and \( D \) can output: \( Z = \sum_{i}^{k} 2^{(Q_i D_i)} - 1 \). Since we also aim at obtaining more generic patterns and, thus, lexical diversity, DIMPLE may be of our interest. While we are not working with paraphrases exactly, DIMPLE relies on the functions \( Q \) and \( D \), and these can be designed as we see fit. We do not have a concrete idea how this metric can be used in our work, but, given its innovation and similar purpose, we found it important enough to mention.

There are also other factors that might be important to measure, such as the quality and/or applicability of the questions regarding a specific task, or the time someone can save by using the system, when compared to the time spent creating the desired questions from scratch.

**Evaluation Guidelines** As previously mentioned, we used on this work an evaluation setting similar to the ones in the state of the art. Annotators were given the source sentence, the generated question and its answer, and were asked to choose among the following options, in each of the three categories:

**The question is:**

- **Plausible** – The question is well formed and makes sense for the shown sentence.

- **Plausible, but...** – The question is not plausible because it is missing some context, uses the wrong auxiliar verb, or is missing a small word to make it grammatically correct.

- **Implausible** – The question is ill formed, cannot be answered, or is inconsistent.

**The answer is** **Correct** or **Incorrect** – The answer is correct for the shown sentence/question pair or not.
If the question is Implausible or Plausible, but..., then it is flawed because of:

- **Grammar** – The question has grammatical errors, like missing particles, or using a noun as a verb.

- **Semantics** – The question is grammatically correct, but is semantically flawed. Examples include using the wrong Wh-word, or asking something that is not stated by the source sentence.

- **Incompleteness** – The question is missing crucial information to either not be vague or make sense as a whole.

### 5.2.1.3 Experiments

We designed this experiment as a proof of concept more than anything else. As of now, we want to show that our approach works and is feasible and, at the same time, to understand what are its flaws.

As a baseline, we took Heilman’s system [Heilman and Smith, 2009] and run it on the selected document. We used the default parameters of questions no longer than 30 tokens and ignoring yes/no questions (that we would not be creating as well in this experiment). The system generated 1004 questions, but we selected for manual evaluation those 89 questions which scored above 3.0 according to the system’s ranking module.

For our system, we automatically learned patterns from a manually designed set of 6 seeds, from which we learned 8 patterns. The manual selected seeds can be found in Table 5.5. The created patterns were applied using the following parameterizations: \texttt{equiv} function from Section 4.2.2.2 with a set threshold for \texttt{equiv}_{W2V} function of 0.1, 0.3 and 0.5, and the same three scenarios by adding either \texttt{equiv}_{SynV} or \texttt{equiv}_{SynN} (see Section 4.3.2.2), thus resulting in a total of nine runs.

We asked two annotators to rate the generated questions according to the guidelines presented in the previous section. Results are in Table 5.6. The total number of questions was 399, but many were shared across runs, which made the total number of questions to be
5.2. PATTERN APPLICATION

Table 5.5: Seeds used in the Pattern Acquisition phase.

<table>
<thead>
<tr>
<th>Question</th>
<th>Support Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>When was Leonardo da Vinci born?</td>
<td>Leonardo da Vinci was born on April 15, 1452.</td>
</tr>
<tr>
<td>Who killed Lee Harvey Oswald?</td>
<td>Lee Harvey Oswald was assassinated by Jack Ruby.</td>
</tr>
<tr>
<td>What did Kafka write?</td>
<td>In 1912, Kafka wrote “The Metamorphosis”.</td>
</tr>
<tr>
<td>Where is Paris located?</td>
<td>Paris is located in France.</td>
</tr>
<tr>
<td>What is the largest country in the world?</td>
<td>Russia is the largest country in the world by surface area.</td>
</tr>
<tr>
<td>How many ounces are in 1 pound?</td>
<td>There are exactly 16 ounces in 1 pound.</td>
</tr>
</tbody>
</table>

Table 5.6: Number of questions generated by Heilman’s system and our system, and the kappa Cohen value obtained by the annotators on each category.

<table>
<thead>
<tr>
<th>Questions Generated</th>
<th>Kappa Score</th>
<th>Total</th>
<th>Plausibility</th>
<th>Answer</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heilman 3.0</td>
<td>215</td>
<td>0.47</td>
<td>0.72</td>
<td>0.34</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>89</td>
<td>0.53</td>
<td>0.62</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>6</td>
<td>0.25</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>30</td>
<td>0.49</td>
<td>1.00</td>
<td>0.38</td>
<td></td>
</tr>
<tr>
<td>SynV 0.5</td>
<td>10</td>
<td>0.53</td>
<td>1.00</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td>SynV 0.3</td>
<td>12</td>
<td>0.62</td>
<td>1.00</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>SynV 0.1</td>
<td>41</td>
<td>0.52</td>
<td>0.90</td>
<td>0.34</td>
<td></td>
</tr>
<tr>
<td>SynN 0.5</td>
<td>43</td>
<td>0.41</td>
<td>0.79</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>SynN 0.3</td>
<td>49</td>
<td>0.35</td>
<td>0.81</td>
<td>0.51</td>
<td></td>
</tr>
<tr>
<td>SynN 0.1</td>
<td>113</td>
<td>0.39</td>
<td>0.75</td>
<td>0.37</td>
<td></td>
</tr>
</tbody>
</table>

The agreement score is considered to be moderate [Landis and Koch, 1977] for the total, Heilman’s questions, and most of our system’s questions, substantially good to almost perfect regarding the correctness of the answer, and poor for the detecting of error types.

We can see that our system does not generate many questions, which is expected given the number of patterns, and that relaxing the equiv function, both on equiv_{W2V} threshold or by introducing matching bypasses, increases the number of generated questions. The numbers presented, however, just indicate how much in agreement the annotators were, and not the quality of the questions. Table 5.7 shows a more quantitative analysis of the results.
### Table 5.7: Percentage of questions considered to be Plausible or Plausible, but... (Plausibility+b) and to have the correct answer (Answerability), for each annotator (a and b) and in common among the annotators.

<table>
<thead>
<tr>
<th>Questions Generated</th>
<th>Plausibility+a (%)</th>
<th>Answerability+a (%)</th>
<th>Plausibility+b (%)</th>
<th>Answerability+b (%)</th>
<th>Plausibility+common (%)</th>
<th>Answerability+common (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heilman 3.0</td>
<td>89</td>
<td>0.70</td>
<td>0.72</td>
<td>0.51</td>
<td>0.67</td>
<td>0.74</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>6</td>
<td>0.83</td>
<td>0.83</td>
<td>0.50</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>6</td>
<td>0.83</td>
<td>0.83</td>
<td>0.50</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>30</td>
<td>0.77</td>
<td>0.80</td>
<td>0.53</td>
<td>0.53</td>
</tr>
<tr>
<td>SynV 0.5</td>
<td>10</td>
<td>0.80</td>
<td>0.80</td>
<td>0.60</td>
<td>0.70</td>
<td>0.70</td>
</tr>
<tr>
<td>SynV 0.3</td>
<td>12</td>
<td>0.67</td>
<td>0.67</td>
<td>0.50</td>
<td>0.58</td>
<td>0.58</td>
</tr>
<tr>
<td>SynV 0.1</td>
<td>41</td>
<td>0.66</td>
<td>0.73</td>
<td>0.46</td>
<td>0.46</td>
<td>0.41</td>
</tr>
<tr>
<td>SynN 0.5</td>
<td>43</td>
<td>0.44</td>
<td>0.67</td>
<td>0.33</td>
<td>0.30</td>
<td>0.35</td>
</tr>
<tr>
<td>SynN 0.3</td>
<td>49</td>
<td>0.45</td>
<td>0.71</td>
<td>0.31</td>
<td>0.29</td>
<td>0.33</td>
</tr>
<tr>
<td>SynN 0.1</td>
<td>113</td>
<td>0.55</td>
<td>0.71</td>
<td>0.35</td>
<td>0.38</td>
<td>0.45</td>
</tr>
</tbody>
</table>

We merged the evaluations for both Plausible or Plausible, but... together in the calculations. The idea is that both annotations are for questions which are almost usable, but annotators might diverge on how perfect the questions are. Thus, it would not be useful for us to focus only on Plausible questions without understanding the bigger picture. Nonetheless, for sake of completion, we must add that Heilman’s system would drop to 43% on the shared annotations (less than a 0.1 drop), while our runs drop 0.2 points systematically. This means Heilman’s questions are, in general, well formed most of the times, while ours have some type of error.

However, if we find these errors to be fixable, we might have a system able to compete with state of the art systems with minimal supervision. Remember that these questions were generated by learning only a few patterns from six seeds.

#### 5.2.1.3.1 Error Analysis

In this section we are presenting the error analysis performed on the results previously presented. Table 5.8 shows the errors identified by one of the annotators, broke down by questions Plausible, but... or Implausible. We are only showing one of the annotators for easiness of reading. Both annotations follow the same pattern and we believe the same conclusions can be drawn from the other, or by using the common annotations. We also chose to show three runs only – one of each for which the
threshold was set to 0.1, which is more prone to errors.

<table>
<thead>
<tr>
<th></th>
<th>Grammar</th>
<th>Semantics</th>
<th>Incompleteness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heilman 3.0</td>
<td>6</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>SynV 0.1</td>
<td>8</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>SynN 0.1</td>
<td>5</td>
<td>3</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 5.8: Number of errors annotated by one of the raters, by questions considered to be Plausible, but... or Implausible.

There are a few things that jump out: first, Heilman’s run rarely lacks completeness. Most of the questions have semantic errors, and grammatical errors usually do not ruin a question. Secondly, our runs seem to lack completeness for Plausible, but... questions, but not for Implausible questions. This means the questions were mostly deemed acceptable but are missing some component that we are losing. Finally, it seems there is a pattern of introducing grammatical errors while relaxing equiv function, which could be expected.

We also asked the annotators their opinion on the process and guidelines. Two major concerns were raised by both: the first regarding the guidelines, which were not totally clear on how to differentiate each case; the other respecting what errors they could label the questions with – more than once they felt more than one error would fit. Thus, in future evaluations, we must have these comments in consideration. Adding more examples to the guidelines and allowing multiple errors are two big steps in that direction.

Considering our system’s runs, the first thing we want to note is that, from the six seeds, we were not able to generate a pattern for the fifth one. The problem was twofold: first, the Semantic Role Labeler (SRL) tokenizer had a slightly different behaviour for this sentence; secondly, the arguments and trees (dependency and constituent) are organized in ways which are not compatible. The second is more interesting to bring up for discussion, but the first one also shows how errors can slip through development.

As a reminder, the sentence was Russia is the largest country in the world by surface area. Concerning the first problem, the SRL tokenizer left area and . together, unlike most of the sentences we had seen thus far (including the remaining seeds). What this means is that the
Figure 5.3: Parse trees for the sentence *Russia is the largest country in the world by surface area* with Stanford syntactic parser (a), Stanford dependency parser (b), and MatePlus SRL (c).

argument for the predicate will include both *area* and the punctuation mark, although the second was supposed to be filtered. When our system is looking for subtrees that include all tokens in that argument, no tree will be found, because no tree contains punctuation marks, as they are properly filtered before.

The second problem lies on the structure of the arguments and the sentence itself. Figure 5.3 shows the different trees for the sentence. Take, for example, argument A1, which covers the tokens *the largest country in the world*. Our algorithm requires that a subtree is found to represent that chunk. However, there is no subtree in either tree that covers only that chunk: in the syntactic tree *the largest country* is part of a NP and *in the world* is actually two subtrees which are part of a larger subtree (PP) containing more tokens (*by surface area*). Analogously, from the dependency tree we are trimming the *Russia* child from the subtree rooted at *country*, but we are not trimming children on nodes below (*area* as child of *world*). Maybe the subtree creation needs revisiting, but it is not clear where the line should be drawn. This leads us, again, to the need of learning how to score patterns, and learn from many examples as a solution for the errors introduced by such decisions.

Regarding the QG process itself, here follow some examples of erroneous questions generated by our system, and the reason behind them.

*When was mother Newton?* was a question generated in one of the runs. The question
was generated from the pattern associated with the seed *When was Leonardo da Vinci born?*, which means *born* and *Newton* were matched. The source sentence was *When Newton was three, his mother remarried ...*, containing a similar predicate of the form \( \text{verb} \xrightarrow{\text{AM-TMP}} \text{arg} \), where the argument in the pattern is *born on April 15, 1452* and the argument in the sentence is *when Newton was three*. There are two situations here that should not be happening: first, we are comparing two words with clear different roles. Despite \( \text{equiv}_{WN} \) being able to assert a degree of relatedness between them, verbs and nouns should not be compared at all (nouns and adjectives might, on the other hand). This happens, though, because the dependency subtrees were similar. Secondly, we are comparing predicates were the verb root has different roles as well: one is the auxiliar verb (*is born*), while the other is the main verb (*remarried*).

*Where is telescope invented?* / *In optics* is another generated question, from the sentence *In optics, he invented the reflecting telescope and ...*. The pattern used comes from the seed *Where is Paris located?*/France. The predicate shared is of the form \( \text{verb} \xrightarrow{\text{AM-LOC}} \text{arg} \xrightarrow{\text{A1}} \text{arg} \). Because we were using \( \text{equiv}_{SynN} \), nouns were issued a match indiscriminately, thus matching *optics* with *France* and *telescope* with *Paris*. If the second match offers variability, the first one is inaccurate for the type of question being generated. We can also see hints of incompleteness, as the question should refer to *the reflecting telescope* or, at the least, *the telescope*. The next example digs into this problem in more detail.

*When was Newton returned?* / *1677* was generated from *In 1677, Newton returned to his work on mechanics ...*. Here we can also see that more information is necessary to make the question more natural (ignoring the wrong auxiliar verb for a moment). However, the pattern was created from a verb that did not require more details (*born*), whereas *returning* requires a complement (*to his work*). The sentence in question has an extra argument in its predicate, covering the remaining of the sentence, while the dependency tree has a specific child referring to that chunk only. In other cases, like in the previous example, the particle *the* is also a child of the token matched: *telescope \xrightarrow{\text{det}} \text{the}*. In both cases, it seems that the generated question would improve significantly by using the matched token plus one associated child span from the dependency tree. However, this type of tailored behaviour is, in practice, just the same as creating specific rules, which goes against the core idea of this work. On the
other hand, it is not clear that more patterns, examples, iterations and scores would help in this specific scenario.

What did England adopt? / The calendar was generated from ... England had not adopted the latest papal calendar .... In this example we can also see a severe error committed, where a question contradictory to the text was generated. This comes from the fact that we are only checking for existing arguments in the new sentence’s predicate, ignoring the remaining, as usually they convey more information not necessary for the question. However, the argument AM-NEG cannot be simply ignored, as seen here. This might be one scenario where we need to introduce a filtering step, as it would require many patterns and feedback to learn that sentences containing such argument cannot be used.

5.2.1.4 Discussion

In this section we started by mentioning a few datasets that can be used on the task of QG and how the task can be evaluated. We followed that by an small experiment using our current system implementation, and compared its performance against a state of the art rule-based system.

Results have shown that the approach is still lacking on some aspects, specially on how many questions are not perfect, including grammatical and semantic errors, and being incomplete. However, if we find these errors to be fixable, the system’s performance might be close to state of the art, specially considering the significantly small number of seeds used.

Regarding the evaluation itself, the annotators have shown concerns about the current guidelines, which should be improved for future evaluations. They also agreed that, although many more questions from Heilman’s system were perfect and felt natural, a significant amount of them also felt convoluted, sometimes showing too much information from the original sentence, or displaying it in a unnatural way. This fact was later confirmed by our error analysis.

The error analysis have shown a set of typical errors our system is performing, namely on the lack of information carried from the sentence to the question, either making it contain grammatical or semantic errors. We briefly discussed a few options to solve them, and our
5.2. **PATTERN APPLICATION**

concerns moving forward.

### 5.2.2 Question Answering

#### 5.2.2.1 Evaluation Metrics

Although in QA it can be more difficult to find the exact answer, the evaluation is also more straightforward: an answer is either right or wrong when comes to factoid QA\(^{11}\). In QG, however, there is no right question. As in many other natural language generation tasks, evaluation can be more subjective.

Regarding QA, we can follow the typical evaluation that systems perform, in terms of precision and recall:

\[
\text{Precision} = \frac{\# \text{Correct Answers}}{\# \text{Total Questions Answered}},
\]

\[
\text{Recall} = \frac{\# \text{Correct Answers}}{\# \text{Total Questions}},
\]

\[
F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.
\]

#### 5.2.2.2 Datasets

The datasets available for QA are mostly shared with the ones available for QG, namely SQuAD and QAD, as they contain a set of Question/Answer pairs, and the source text from where to find the answer.

Other typical datasets used in this task are TREC\(^{12}\) and CLEF\(^{13}\) corpora, but they have the downside of requiring a passage retrieval step, as they only provide a collection from where answers should be retrieved from.

---

\(^{11}\)Sometimes partial correct answers are accepted.

\(^{12}\)http://trec.nist.gov/data/qamain.html

\(^{13}\)http://www.clef-initiative.eu/dataset/corpus
5.2.2.3 Experiments

As of now, we did not perform any experiment in a QA setting.
Proposed Work

In the previous chapter we presented a small set of experiments that work as a proof of concept for our Question Generation (QG) system. The results were satisfactory given the context, but we are aiming much bigger for the end of this thesis. In this chapter we will go through a set of scenarios of evaluation that we would ideally like to see covered by this work. Mainly, we also want to make a difference in two dimensions: first, by providing a large scale dataset for development and evaluation of QG. Secondly, by proving that our system has an impact as an authoring tool for Intelligent Tutoring Systems (ITSs) and Massive Open Online Courses (MOOCs) creation. These three correspond to the three hypothesis presented in Chapter 2.

6.1 Question Answering Setting and Pattern Improvement

Not only we want to make the QG system better by itself, but we would also like to fully prove that our patterns can be used for Question Answering (QA) and, as a consequence, to learn more patterns and improve the system. This completes the view we presented in Section 2.1 and, concretely, Hypothesis 1. Improving the matching and generation process to incorporate the flaws presented in the last chapter is crucial, but in this section we will focus on the other research statements not yet met.

Given a set of epochs $E = e_1, \ldots, e_e$, the system will use a given QA dataset to validate the current patterns and learn new patterns from previously generated pairs. Each epoch $e_i$ will be evaluated independently, both in accuracy of answers given and quality of generated questions, with the goal being that the final epoch, $e_e$, shows an improvement from the initial state in $e_1$. The concrete datasets and QG evaluation procedure is still unclear but, ideally,
a goldstandard of quality questions will have been collected by then, lessening this way the
effort of such experiment. For example, SQuAD could be used as the goldstandard, but it
has the problem of new questions not being in the dataset, thus not having a reference. On
the other hand, Amazon Mechanical Turk (AMT) requires time, which makes this process
not ideal to quickly evaluate the system through a number of iterations. Therefore, a more
controlled setting would be better to incrementally develop the feedback loop.

We can also use our patterns to perform QA, and one interesting way to use our system
could be to employ it as support for an existing system, as an answer candidate extractor.
This way, we can evaluate extrinsically the quality of our patterns, if the external system is
able to improve its performance on a QA task by adding our system’s support.

6.2 Development of a Large Dataset for Question Generation Development

We already described the available datasets in Section 5.2.1 for QG. The largest one is
SQuAD [Rajpurkar et al., 2016], and can be a good test bed for our system. The dataset
contains over 100,000 questions across over 500 Wikipedia articles, crowdsourced through
AMT. Because the questions were created by humans, there is a understandable limit on
how many questions each created regarding each paragraph. In fact, workers were required
to input 5 questions (and answer them). This leaves plenty of room for unasked questions,
even if not as important as the ones in the dataset. An automated system like ours could,
thus, be an interesting tool to augment the dataset, by automatically creating more plausible
questions for each paragraph, effectively contributing for a more complete QG dataset. This
work corresponds to Hypothesis 2, presented in Section 2.2.

Questions generated this way have to be evaluated through AMT as well, not only because
of the high volume of items, but to make sure they meet the SQuAD standards. AMT provides
us a framework to easily and quickly evaluate a high volume of Question/Answer pairs, and it
will be key for our evaluation process. This is, also, another reason why our first evaluations
were done with local annotators: the guidelines to be used have to be tested and perfected
before committing a massive evaluation to AMT.
Concretely, the generated Question/Answer pairs will follow SQuAD specification, and can be used by anyone as an extension of the main dataset. Besides the AMT evaluation, another interesting way to measure the impact of the new data is to use as existing system designed for the main task (QA) and measure the performance when provided with the extended dataset, compared with the main dataset only. Another interesting point is that the new dataset can be curated to include only Question/Answer pairs which obtained a good evaluation through AMT.

6.3 Automatic Quiz Generation for Digital Course Creation

We started by motivating this research by stating that Intelligent Tutoring System and Massive Open Online Course require authoring, which imposes a limitation on their creation or, at the very least, implies a huge cost. Although we approached QG in a more generic sense throughout the document, this concrete scenario is still one of our goals. Let us say a professor wants to create a quiz about a given topic. He is looking for good questions, obviously, but he is also expecting them to be relevant. A slightly more off-topic question is not particularly relevant, even if well formulated and appropriate for the original sentence.

Let $D = S_1, \ldots S_n$ be a document composed of $n$ sentences, and $r_i = \{0, 1\}$ be the relevance score for $S_i$. Ideally, our system is looking for all questions $Q$ such that $\forall q \in Q : q.source.r = 1$. These questions can, obviously, range from bad to good, and from irrelevant to important. The assessment of their quality can also be done with help of AMT, but notice that a new factor is here involved: relevance of the questions and source sentences. This parameter is arguably more subjective than question quality, and depends a lot on the teacher’s goals. Therefore, it can be important to have someone work with us specifically for a task of this kind.

In Figure 6.1 depicts a Venn diagram representing the expected proportion of questions in the universe of possible questions to be asked regarding a document $D$, according to some generic metric of interest. In the figure one can see the circle labeled $A$ representing the total of relevant questions, and the inner circle $B$ representing the number of those identified by
the teacher. The area identified by $C$ respects to the generated questions, and shows that it is expected that it generates mostly relevant questions, maybe including some the teacher has not considered, but that irrelevant questions will be generated as well. Finally, the inner circle $D$ represents, from those questions generated, the ones which have a wrong answer or no answer.

![Venn diagram (Figure 6.1)](image)

Figure 6.1: Venn diagram depicting the expected proportion of questions according to some generic metric.

The final goal is, thus, that $D$ is as small as possible, and that $C$ overlaps as much as possible with $A$ or, at least, $B$. Notice, however, that the dimension of $C$ and $D$ will depend on the metric used. For instance, a more relaxed quality measure like Plausible, but... presented in Section 5.2.1 will result in larger circles.

Still, our system might be of interest even if the circle $C$ does not overlap quite as well with $A$ or $B$. The motivation is to diminish the efforts in content creation, and that might still be achievable.

Let $t$ be the time an expert takes to generate $N$ questions for a given document $D$, and $M = I + R + R_w + R_e$ the number of questions our system generates, where $I$ are the irrelevant questions, $R$ are the relevant questions ready to be used, $R_w$ those questions with a wrong answer, and $R_e$ those potentially relevant questions with errors. Let also $t_m$, $t_i$, $t_w$ and $t_e$ be the time the teacher needs to, respectively, go through all questions, remove irrelevant questions, fix the answer for questions $R_w$, and edit the $R_e$ questions. If $t > t_m + t_i + t_w + t_e$ we might have an valid option for an authoring tool, depending on the percentage of irrelevant
questions $\frac{I}{M}$. For example, it would not be much valuable if the teacher can save time but the end result is just a couple of relevant questions. The worst case scenario is when all times are non-zero, so the goal is to take them close to zero, that is, to present less irrelevant questions, and more relevant questions and, if with errors, errors that are minor and easy to fix. Finally, another interesting analysis is to see if any of the relevant generated questions is new, i.e., $\exists q \in R \cup R_w \cup R_e : q \notin N$.

A formative evaluation with CMU faculty and courses taught be them could be carried, which allows us to get (subjective) feedback from real professors, while measuring the relevance of the questions and time spent curating the final list, as described before. This work is directly related with Hypothesis 3, presented in Section 2.3.
Following the proposed solution for the problem in hands, and the current status of the experiments conducted, we discuss the overall schedule for the doctoral study in this chapter. The first section exposes a schedule overview of all work that needs to be done, while the second section provides a detailed scheduled for the proposed work of last chapter.

7.1 Schedule Overview

**May 2017**  We expect to have discussed this document with the jury.

**July 2017**  We intent to update the system to solve two concerns: scalability and speed. As of now, patterns are stored in file and loaded at the start, instead of being hashed and loaded as needed. This becomes a problem when the number of patterns increases drastically.

**September 2017**  We intend to incorporate in the system’s learning process the ability to evaluate the patterns created against a Question Answering (QA) dataset. These changes correspond to Section 4.4.

**December 2017**  We expect the system be able to self-regulate the iteration process through the feedback provided (either manually or automatically), and to refine and merge older patterns. This corresponds to the work presented in Section 4.5.

**February 2018**  We want to reserve some time to include new algorithms to the learning and matching process that we see fit.

**April 2018**  We expect to be able to run all experiments that will allow us to test Hypothesis 1.

**May 2018**  We plan to have Hypothesis 2 and Hypothesis 3 tested.

**Later in 2018**  We aim to deliver the final document and have the thesis discussion.
### 7.2 Detailed Schedule for Proposed Work

In this section we detail the schedule for the final proposed work. Figure 7.1 depicts a timeline for the work, which corresponds to Hypothesis 1, 2 and Hypothesis 3.

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Mar 27</th>
<th>Apr 10</th>
<th>Apr 17</th>
<th>Apr 24</th>
<th>May 1</th>
<th>May 8</th>
<th>May 15</th>
<th>May 22</th>
<th>May 29</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gather corpus of professor (H3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learn patterns</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run System EG (H2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run System Other Dataset (H1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run System professor DS (H3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMT - draft</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMT - pilot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMT - real test (H &amp; 2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process Results (H1 &amp; 2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMT - real test (H3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evaluate on professor (H3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process Results (H3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.1: Detailed schedule for the final proposed work corresponding to Hypothesis 1, 2 and Hypothesis 3.

In the figure there are two paths: one in blue, for Hypothesis 3, and one in green for the other hypothesis. One of the bottlenecks is the evaluation in Amazon Mechanical Turk (AMT), which requires a previous step of drafting and piloting the experiment. Only after concluding those we can effectively run the real test on AMT for any dataset. Another bottleneck are all interactions with real humans, namely the professor, reason why he reserved much more time for those tasks.

The figure also allows one to understand that there are a couple of tasks that can be parallelized, like running the system for multiple datasets, and evaluating those results on AMT.
In this document we presented the motivation and introduced the research topic of this Thesis, along with the contributions that are expected to result from it and the testable hypothesis.

To contextualize the proposed work, a survey on Question Answering (QA) and Question Generation (QG) was presented, with special emphasis on systems that use patterns and semantic features. Important resources that can be useful in our work were also described.

We detailed our proposed solution for the QG system, grounding it with state of the art examples and comparisons. We present both a formalization of the desired functionality and the current implementation. The proposed work consists of four parts, corresponding to each of the first research statements (Hypothesis 1), where the last one was split in two.

We also provided a first set of experiments conducted with the current version of our system. We first evaluated our pattern acquisition step, which relies on an alignment task, against state of the art aligners, and then we run our QG system on a Wikipedia article, with a small number of automatically learned patterns. The results show that our aligner is suitable for the task in hands, and that the generated questions have some quality, but that the process has still significant flaws.

In the next chapter we detailed our proposed work, showing our expectations for the final results, and how they map into the presented research statements and testable hypothesis.

Finally, we provided an overview of the main milestones of our work and the expected dates to fulfill them.
Bibliography


David Lindberg, Fred Popowich, John Nesbit, and Phil Winne. Generating natural language questions to support learning on-line. In Proceedings of the 14th European Workshop on


