SINGLE AGENT ANSWERING SYSTEM
an Intelligent Agent approach to Question Answering

A MASTER THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS
COMENIUS UNIVERSITY IN BRATISLAVA

supervisor:
doc. PhDr. Ján Šefránek, CSc

Štefan Konečný
May 2010

Abstract

In this thesis we present a novel approach to Question Answering. We look on this task from perspective of an Intelligent Agent, which strives to discover in its environment facts relevant to the question it has been asked. Through the interaction with its environment - the Wikipedia online encyclopedia - it can acquire new facts and modify the relevance of the facts it already knows .

We also uncover some problems connected with this new approach and suggest solutions to overcome them. Of the connection functions and value functions we have designed, exponential connection function coupled with maximum or tresholded average seem to be the most viable. However further evaluation and research on this functions is still necessary.

We hope that work may provide inspiration for other Agent and Multi Agent Systems, capable to answer Relationship Questions and also other complex questions categories.

Acknowledgement

I would like to express my gratitude towards my supervisor Ján Šefránek, who provided me with unprecedented academic freedom and support when I have needed it the most.

I would also like say i big Thank You to my Mom, as well as the rest of my family for their ceaseless support.

I owe a lot of to my Employers Michiel Munneke and Michiel Banneke, for providing me with holiday to write this work, and all the friend and colleagues who have done my work meanwhile

Many thanks belongs to people on which work I have built upon Katrin Erk and Sebastian Padó for Shalmaneser, Simone Ponzetto for Wikirelate! and Nils Reiter for FrameNet API. Without their work, my work wold not be possible.

And thanks to You God, for all the inspiration and blessings.

List of Algorithms

List of Tables

List of Figures

Contents

Chapter 1

Introduction

Of all the epithets given to the current age, the 'Information Era' is one of the most prevalent ones. Without any doubt the quantity of information or produced nowadays is larger then ever before. This poses a natural challenge of how to find the needed information in the ever growing pile of data.

While the common search engines succeed to fulfill needs of the majority of their users, they have their limitations. As they are mostly based on keyword queries, they force the user to come up with a suitable query. Although this task is simple in most of everyday situations, the more abstract or complex the information need of the user becomes the harder it gets. In most of the cases the success or failure of the search depends on whether the information is worded similarly to the keywords the user typed in as his query.

The ambition of Question Answering (QA) is to alleviate the user of this problem. The user should be able formulate his information need as a simple question and receive an relevant answer, as if he would ask an omniscient human being. The QA system should take care of all the rest, that is, it should *understand* the information need, identify the proper context, take into account different phrasing of the possible answers, split a complex question into a simpler ones and aggregate the partial answers (possibly originating from different sources) into the final one.

Of course QA is an an area of a vivid scientific research and non of the state of art systems is close to the ideal. In this work we introduce the Single Agent Answering System (SAAS). We take a different a perspective to the QA task, compared to a Traditional

QA systems (we define this notion specifically in the upcoming section). Instead of merely identifying a passage in a document, which is the most relevant to the asked question, we attempt to build a partial word model relevant to the question. That is we try to identify the relevant entities involved in the question as well as their properties and relations between them. To achieve this goal we pursue an iterative process of detecting entities seemingly relevant to the topic, obtaining more information about them and update our knowledge accordingly to what we have learned. Our intention is to converge to a Knowledge Base containing relevant entities and relevant information about and connecting them. The system was designed specifically for answering relationship question which inquire about relations between two and more entities.

While the performance of our system is far behind the ideal QA system, as well as the current best performing QA systems, it offers a fresh perspective to the underlying topic and identifies some caveats connected to the new approach. It also proposes some partial solutions to them and evaluates their impact.

1.1 The Traditional Question Answering Approach

Although such a claim involves a great deal of simplification, most of the QA systems we are aware of exhibit a strong common pattern. Before we describe it in more detail, let us introduce a trivial question on which we shall exemplify the process:

> Who is the current president of the US?

A Traditional system as we understand it would perform the following steps:

- Identify relevant query terms. This the same step all users do facing a search engine. The most likely outcome is ’US’, ’president’, ’US president’. Most of the system will enhance the query with synonyms and different phrasing variation to get higher recall and may end with a Boolean query resembling this one:

  (’current’ OR ’present’ OR ’contemporary’) AND ... AND (’president’ OR .... OR ’head of state’) AND ... AND (’US’ OR ’UNITED STATES’ ... OR ’USA’)

- Perform question analysis. As it is a 'who?' question, we know for sure that the answer involves a person. We can thus try to detect names in answer candidate, by identyfing capitals and common first names. Some system also construct a pattern for the answer candidate, by applying some grammatical transformations to the question such as:

  The president of the US is [?].

  [?] is the president of the US.

  And if their find an answer candidate (sentence) which fits the pattern the proclaim the expresion which fills

  [?] to be the answer.

- Cast a query to your corpus (for instance web) and retrieve the set of most relevant documents
- Split the document into smaller textual units (sentences or snippets) and evaluate their relevance. Some of the common measures are the number of keywords the sentence contains, whether or how close it matches a given answer pattern.
- Order the sentences or snippets according to the measure used above. Identify an answer from the sentence (in our case find the name)
- Return the answer. This could be the answer extracted from the most valued text unit, or the most frequent one.

1.2 The Intelligent Agent approach

As already stated we shall model QA as an intelligent agent task. We shall adapt the following definition of agent for our purposes [Wooldridge and Jennings (1995)] :

> An *agent* is a computer system that is *situated* in some *environment* and that is capable of *autonomous action* in this environment in order to meet its delegated *objectives*.

More specifically for our SAAS agent the following holds:

environment - is a local copy of the Wikipedia online encyclopedia

action - is searching the environment for the information, which it deems to be most relevant to the question[1]

objective - retrieve the most the relevant information to the question

Thus or agent receives an question and start to search Wikipedia for the answer. First it analyzes the query and an complies into a Target Knowledge Base (or Target Model, TaM), which should contain all the relevant facts and ascribes a weight to them according to their relevance. Afterward the agent should select the most promising fact (the most relevant one) and explores it further by retrieving the Wikipedia page to which this fact is linked to. Next it is searches the page[2] for connections between facts in its Knowledge and other facts (which may or may not be in his Knowledge). Depending on the on the strength of the connection and relevance of the known fact it links to (in the case of a connection between two known facts we consider only the more relevant one), a new fact may be added to the Knowledge, or the relevance of a known fact may be increased. In fact the agent has two separate Knowledge Bases (KBs) the TaM described above, and a Topic Model (ToM). Both of them fulfill the same purpose and have essentially the the same functionality. The difference lies in the fact that while Target TaM gathers only facts strongly relevant to the question *itself*, whether the ToM stores facts about the question *area*, which may be only loosely related the particular question. The reason behind maintaining a ToM, is to gather more general information about the question and entities involved in it, in hope that it may lead us to the specific answer in the long run. Facts are assigned to the Knowledge Bases (KB, we shall use Knowledge and Knowledge Bases as an umbrella terms encompassing both Target and Topic Model) according to their relevance. As the relevance can be gained through a connection with a more relevant fact, a fact from ToM may be 'promoted' to the

[1]Strictly speaking most of our queries are not questions, they are rather touples of declarative sentences describing the questioners (mostly refereed to as Analyst) interest. Therefore it would be more suitable to use the term topic for them. We shall ref fer to them as questions nonetheless, in order to avoid to confusion with the Topic Model.

[2]actually due to the complexity of the processing, we process only a set of most promising snippets

Algorithm 1.1 SAAS lifecycle

```
initiate the Target Model
while !stopCriterionn
        w <- best next page(knowledge)
        snippets <- best snippets(w)
        new connections <-process (snippets)
        knowledge <- update (knowledge, new connections)
        output <- evaluate (snippets, target knowledge)
do while
return sorted output
```

TaM, if its relevance increases due to newly discovered connections. Note that because of the way we transfer relevance (see Chapter 4), this only possible, if it has a strong connection to a fact already residing in the TaM. After the KB is updated (that is both TaM and ToM), the agent assigns relevance to a set of best promising snippets[3] (according to the *updated* TaM) and stores them as anoutput (the snippets). Then the next most promising fact is chosen (take note that the updated knowledge may influence which one it is) and the whole procedure is iterated until a stopping criterion is met. We currently use a one hour timeout as our stopping criterion. An ordered list of snippet descending according to relevance is returned as output.

For now the largest difference between SAAS and a traditional system is that SAAS uses (more advanced) knowledge models, capable of deducing relevance of facts from the context. Because SAAS is mainly data driven this may be both an advantage and disadvantage, depending of quality of the data provided. As we shall see SAAS is quite easily 'lead astray' by an improperly initiated TaM falsely assigning high relevance to irrelevant facts. This effect of convergence to a local optimum (that is we converge to a Knowledge Base which assigns high value to facts related to the question but not the most related ones) or in entirely wrong direction (when the unrelated terms have much higher relevance than desirable, drifting). In fact the whole experimental part of thesis is devoted to avoid drifting, and evaluates the different method we have proposed to avoid this phenomenon.

[3]these are the same snippets which were selected for processing

This thesis shall proceed as follows. In the next chapter (Chapter 2) we shall introduce the data and software resources we use, that is the Wikirelate! [Strube and Ponzetto (2006),Ponzetto and Strube (2007, 2006)] framework to access the Wikipedia, our question test set and the parser we use process the snippets Shalmaneser [Erk and Pado (2006a)]. Next we shall explain how we represent Knowledge and how we use it (Chapter 3). Subsequently we present how the initial TaM is created and how the relevance is assigned and transferred in our representation of Knowledge (Chapter 4). In the experimental section (Chapter 5)we present the output of our system, and address how our attempts to avoid drifting influence it. Than we conclude our work in the conclusion (Chapter 6).

Chapter 2

Used Data and Technologies

2.1 Corpus

SAAS uses a local copy of Wikipedia as its solely corpus. The leading QA conference in English language, the Text REtrieval Conference, has used the AQUAINT corpus for the Relationship task in 2005 [M.Voorhees and T.Dang (2006)], and Complex 'Relationship' Question in 2006[Dang et al. (2006)]. In 2007[Dang et al. (2007)] AQUAINT2 and Blog06[CraigMacdonald and IadhOunis (2006)] were used for the Complex 'Relationship' Question. As a result, all the systems which run on a similar test data as SAAS, run on a very different corpus. As in all mentioned task the QA Systems must provide evidence for their answer (the test snippet has to be in the corpus), it would make the comparison between our system and of theirs very hard.

The main reason why we opted for Wikipedia was that it has freed us from the Information Retrieval aspects (finding the document relevant to the query) present in the Traditional Systems. Thanks to the Wikirelate! [Ponzetto and Strube (2007)] framework we could easily access the local copy of Wikipedia database through a java API, and have not had to take care of indexing and managing the corpus ourselves. On the negative sized we could only look up articles by title name and could not use full text queries (although Wikirelate! handles redirects, so we are redirected to the appropriate page). As Wikirelate! provide us only with text strings representing raw and clean text of the page, we had to develop the classes representing unit of text (sentences, sections, snippets and the overall page structure) ourselves, on the top of the API.

While Wikipedia is not usually used as a primary QA corpus, it was successfully used in an array of NLP (Natural Language Processing) tasks such as detecting synonymy [MacKinnon and Vechtomova (2007)], Named Entity disambiguation [Bunescu (2006)] and Semantic relatedness[Gabrilovich and Markovitch (2007)]. Wikirelate! itself was developed to detect semantic relatedness [Strube and Ponzetto (2006)] as well and later it was used for correference resolution [Ponzetto and Strube (2006)] . In [Ahn et al. (2004)] Wikipedia was used as an topic model for the 'other' task. In this task the system is asked a series of questions about a topic and subsequently it is asked for other relevant information about that query (which was not asked for yet). However in this case the topic model is just a set of sentences occurring in the same article and they do not identify the facts associated with the topic the way that we do.

As I final remark, Wikipedia is not well suited for all possible topics. Because it as an encyclopedia at first place it may be not useful to answer question about recent events, fashion trends and public opinions, as its strives to provide objective,neutral and high quality information. Also because of its encyclopedic nature it contains less redundancy as a news wire corpus or the web. Therefore, especially for a very specific topic, it is very important to find the right article covering the topic, as the searched information may not be repeated somewhere else.

2.2 The Test Data

Since the very beginning, SAAS was designed for answering relationship questions. The Relationship Task was introduced to TREC in 2005 and continued in 2006 and 2007 under the name Complex 'Relationship' Task. We have chosen this task as it explicitly calls for topic modeling and requires more topic 'understanding' as the factoid task. Creation of an appropriate test set is not trivial, as the relations should not be neither trivial neither impossible to find. By adding together the relationship question from the three TREC tasks, we have obtained a test set of 86 (one of the 2007 questions was not used in TREC, as the answer could not be found in the corpus. However the required information was present in

Table 2.1: Test set topics

Topic	Count	Topic	Count	Topic	Count
Military - terrorism	17	Economics	13	Politics	12
Drug trafficking	7	Science	4	Music	2
Smug ling	6	Health	11	Criminal	2
People Trafficking	5	International relations	7	Total	86

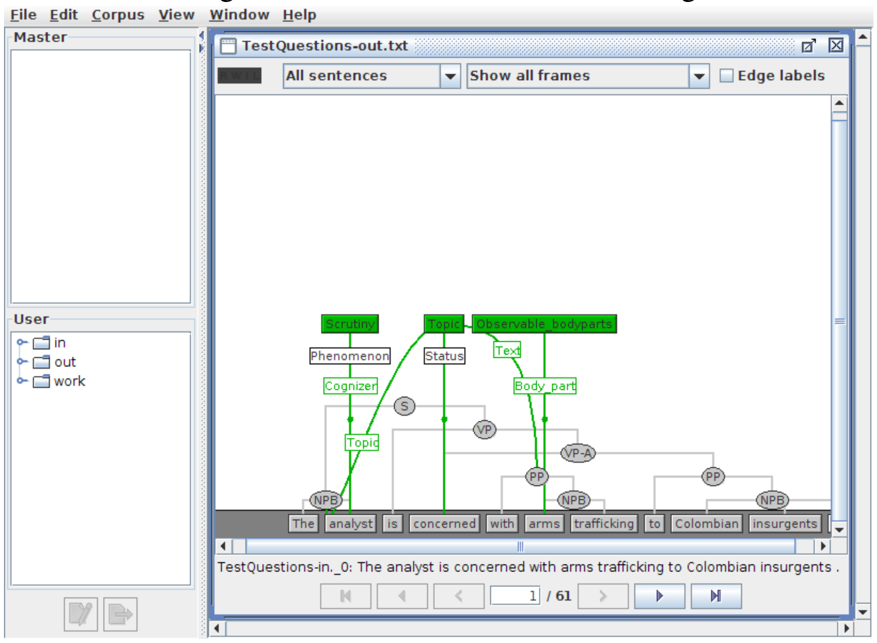
Wikipedia, even twice, once on the page of each person it relates). The topics covered in the test set are described in Table

Unfortunately due to time constraints, we could fully evaluate our experiments on eight topics only.

2.3 Shalmaneser

Shalmaneser (*Shal*low Se*man*tic Par*ser*)[Erk and Pado (2006a)] is one of the key components as the parse it produces are essential for propagating the relevance values. Shalmaneser is written in Ruby and binds together various language processing tools such as Collins parser [Collins (1997)] for parsing, Treetagger [Shmid] for lemmatization and TNT [Brants (2000)] for part-of-speech tagging. Because of its nature of using already available tools, its design is modular and easy to modify. We took advantage of this, when we removed the Fred and Rosy components. Both components are used to annotate the sentences with FrameNet elements (Fred identifies the Frames and Rosy their Frame Elements). We do not use this features for now, and their removal resulted in a noticeable speed up, from 11 minutes 37 seconds to minute and half (on 60 sentences). While Framenet relations may be very useful, as their provide more specif relations as pure syntax, performance issues open up the question whether they are suitable to use for our purposes. Shalmaneser can process plain text, Salsa/TigerXML[Erk and Pado (2006b)], or FrameNet LU XML and produces its output in SalsaTigerXML. SalsaTigerXML is an enhancement of TigerXML[A.Mengel and W.Lezius (2000)] which is an xml format to annotate sentences with syntactic information (and Salsa adds the semantic information). Both formats can be conveniently displayed through the SALTO tool[A.Burchardt (2006)], which serves also as annotation tool

Figure 2.1: SALTO in action, including Framenet annotation



TestQuestions-in._0: The analyst is concerned with arms trafficking to Colombian insurgents .

for them. All the parse trees displayed in this work are generated by SALTO. SALTOs GUI is depicted in Figure

2.4 Framenet API

Although we do not use the Framenet features, we have used Nils Reiter's [Reiter (2007)] Framenet API to access the syntactic trees in Salsa. We have extended the TreeNode classes from this API into bidirectional (maintain the child parent link as well) ones and their form the basis of all our tree based algorithms as Algorithm 3.1.

Chapter 3

Using Wikipedia Knowledge

In this chapter we shall discuss how SAAS acquires information from Wikipedia, and how this knowledge is used to assign numerical values (called *weight* and *temperature*) to sentences and snippets. We also discuss the process of selecting the best snippets(in terms of highest *temperature*) for further processing. Thus this chapter covers the following topics

1. Definition of WikiFacts (WF)s
2. Extracting WFs from sentences
3. Assigning numerical values to sentences
4. Selection of snippets for further processing based on their temperatures.

Details of assigning *relevance* values to WFs and modifying it based on connections to other WFs are covered in the next chapter.

3.1 WikiFacts

WikiFacts (WFs) are the most elementary unit of knowledge we deal with. The KBs are basically nothing, but a collection of WikiFacts. An WF is comprised of:

- address of a Wikipedia article (in normal and lemmatized version)
- a positive real number value - the *relevance*

- some helper flags for instance *followable*, *visited, inTarget...*

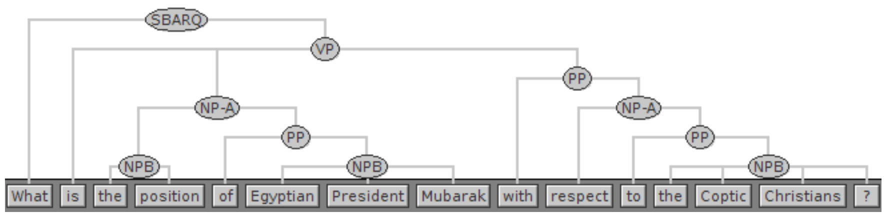
All the processing is based on WFs and their relevance values. Plain sentences (as opposed to parsed ones, for which we also have access to their parse trees) are assigned a value which we call *temperature* which is based solely on the amount of WikiFacts that the sentence contains (we have experimented with various aggregation functions for this purpose as we shall discuss in Section 3.4). Based on this property, snippets of text are selected for parsing (their *temperature* is simple the average of the *temperature* of sentences they contain). When assigning or modifying *relevance* of an WF only connections to other WFs are considered and the final *weight* assigned to an output snippet is also dependent only on how many and how relevant TaM WikiFacts it covers.

WikiFacts fall into two categories: *followable* WFs and *knowledge* WFs. While the followables represent WikiPages, we may like to visit to gain further knowledge, the *knowledge* merely link a word with a *relevance* value. Whether the fact is followable depends on the following rules:

1. WF which's address is a multiple word expression is followable
2. None of WFs which's address is in the stop word list is followable
3. Any WF which's address corresponds to an adjective, adverb or verb is *not* followable
4. WF which's address corresponds to a noun is followable
5. Single word expressions of any other word class are not considered WF candidates. That is only an adjective, adverb, noun, verb or a multiple word expression can be a WF.

Following these rules *'Red tide'* (algal bloom) is a followable WF (multiple word), 'red' is a knowledge WF (adjective) and tide would be also a followable WF (noun). We have selected the eligible word classes based on their function in language, as they mostly represent entities (nouns), their properties (adjectives), relations between entities (verbs) and the properties of these relations. On the other hand stop words (connectives, pronouns, articles,...) convey little meaning and therefore we ignore them. To identify the word class we use the Part of Speech (POS) Tag assigned by the Shalmaneser parser.

Figure 3.1: The Parse tree of our example sentence displayed by the Salto tool



All WFs not yet contained in the KB, have zero relevance. New WFs are added to KB only when this default value is increased through the Transfer of Relevance (described in section 4.1), which means that the new WF is strongly syntactically connected to a known WF (already in the KBs). This process (the transfer) thus requires syntactic information and the ability to find WFs in the (parsed) sentence. To assign temperature to sentences we resort to a simpler process for detection of WFs, to increase its speed and due to the lack of full syntactic structure (as we do not have parse tree of the sentence yet). We discuss the both procedures next, starting with the one running on parsed sentences.

3.2 Discovering WikiFacts for KB

As already stated, the WF are added to the KB's based on information obtained from the parse tree. For the sake of clarity, we shall exemplify our method on the following sentence:

> What is the position of Egyptian President Mubarak with respect to the Coptic Christians ?

The parse tree of this is displayed in Figure 3.1

The WFs are discovered through a bottom-up recursive algorithm iterating over the tree nodes. We create list of WF candidates (Candidate List - CL) for each node and the candidate list of the root node becomes the list of WFs for the whole sentence. For the leaf nodes (terminals, representing words in the sentence), the CL can either contain the word itself or is empty, if the word does not belong to an appropriate word class as discussed in the rules above. For a given non-leaf nod N, we first get the CLs of its children and merge them into an initial CL for N. Subsequently we look for (multiple word) WFs,

Table 3.1: WFs extracted from “What is the position of Egyptian President Mubarak with respect to the Coptic Christians ?”

WF address	Wiki page	Followable
Position	Position	yes
Egyptian President	President of Egypt	yes
President_Mubarak	Hosni Mubarak	yes
With_respect_to	Dependent and independent variables	yes

which’s composing words have been covered by different children of N. To this end we first retrieve the list of all words covered by any child of N (the Covered Words Set(CWS) of N, the set of all words which are descendant of N in the tree structure) in the order they have appeared in the sentence. Next we apply a sliding window of decreasing length (ranging from the size of CWS of N to the size of the least CWS of some child of N) and if all the words in the window form a title of an Wikipedia page (and these words belong to CWS’s of at least two different children), this multiple word expression is added into the CL of N. Whenever such a merged candidate is discovered, all WFs which were in the CLs of children (of N) involved in that expression are removed from the CL of N. The following scheme (Algorithm 3.1) presents a more concise representation of the procedure just described:

While the description of the procedure is quite complex, its purpose is quite simple:

1. we want to obtain the most specific WFs possible, as we believe they are more discriminative (our only indicator of specificity is the number of words involved)

2. we want to avoid unnecessary computation, therefore we consider only set of words linked by syntax (belonging to the same subtree)

3. additionally, we may discover multiple word WFs which are syntactically linked yet spatially separated in the sentence.

Following the procedure, we will find WFs enumerated in Table 3.1 in the example sentence.

Note that because we pursue generality, WFs ’Egyptian’,’President’,’Mubarak’ are not included, as they form a part of a multiple word expression. The absence of ’Coptic’,

Algorithm 3.1 Discovering WFs in a parsed sentence

```
if N is a leaf node representing a word:
        if N is not in stopword list and has the right word class
                the CL of N is the word itself
        otherwise
                CL of N is empty

if N is a non-leaf node
        create an empty CL for N

        for each child C of N
                add its CL to the CL of N
                add the CWS of C to CWS of N

        start with window w covering the whole CWS of N

        repeat
                slide the window w through CWS of N

                if the sequence s of the words is:
                        completely covered by w
                        and forms a name of wiki article
                        and these words do not belong to the CWS
                        of the same child of N
                then
                    add this sequence s to the CL of N

                        remove from CL of N the members of CL of any
                        child which has contibuted a word to s
                        (through its CWS)

        decrease the size of w, until it reaches
                the size of the minimal CWS among
                the children of N

if N is the root node
        its CL is list of all WFs found in the sentence
```

’Christians’ and ’Coptic Christians’ is also noticeable, and these WF candidates were discarded during the merging of ’with respect to’. The case of ’with respect to’ also illustrates that the WFs may link to pages which are quite unexpected the title given.

As a final notice, we prefer to use lemmatized addresses whenever possible (we obtain the lemmas from the parsing tree). That is instead of ’Coptic Christians’ we would use ’Coptic Christian’, instead of ’Oyster’s’ we use ’Oyster’ and so on. Obviously this not always possible, for instance we cannot use ’arm traffick’ in place of ’arms trafficking’. Therefore for each followable WF we check whether there exists an article for the lemmatized version for its address and we use it if it does.

The advantage of lemmatization is that it enables to one WF yield match on multiple word forms, so that ’universities’ matches with lemmatized ’university’ and ’exported’ with ’export’. We also remove the possessive suffix (’s, so we use ’Cuba’ and not ’Cuba’s’) whenever possible. To capitalize on the both operations we need to perform them on the both ends, that is we need to create the lemmatized representation in the KB and need the lemma of the word which we try to match to WF in KB. In practice for each word we look up its lemma first and if there is no match we try again with the word itself.

3.3 Finding WFs in plain sentence

As for their parsed counterparts, we would like to be able to assign a set of WFs to plain sentences. Based on the set we will calculate the temperature of the sentence and optionally select it for further syntactic processing. As the computation of temperature is essentially a preprocessing step, the detection of WFs it is based upon has to be reasonably faster. It is basically a simplified version of Algorithm 3.1. The two main differences are:

1. We set an upper limit to the size of the window to 5, and we decrease it until it reaches one

2. We completely ignore any structure in the sentence and treat it as an (ordered) list of words

Algorithm 3.2 Detecting WFs in a plain sentence

```
initialize an empty CL for the sentence

start with window w of size 5

          repeat
                    slide the window w through the sentence

                    if the sequence  of  words is:
                              completely covered by w
                              and forms a name of wiki article
                              and none of the words is marked
                              used in a WF of size greater
                              then that of w

                    then
                         add this sequence to the CL

                              mark all words as used in a WF
                              of the size w
          decrease the size of w, until it reaches 1
```

Once again, we would like to find the most specific WFs, and as a consequence if a sequence s in CL none of its (non trivial) subsequences is. The CL can still contain overlapping sequences of the same length though. In our previous example sentence ’President’ and ’Mubarak’ still would not make it to the CL , because of the ’President Mubarak’ and ’Egyptian President’ which are both present (in the CL). This time the CL also contains ’Coptic Christians’, as we do not consider syntactic relations.

As we have not access to the word lemmas in this setting, we apply stemming instead, using the Snowball Stemmer [Porter (2001)]. We also remove the possessive suffix.

3.4 Assigning values

Now as we are able to obtain WFs contained in a sentence, we can introduce a mechanism how to assign value on them. All we need to do is to retrieve the values of the WFs contained in the sentence and aggregate them together. If some of the WF is not in KB yet, it is ignored. We assign two different kind of values to the two different kinds of sentences: *temperature* to the plain sentences and *weight* to the parsed sentences. Both are internally represented by a non negative real number and serve a different purpose. While *weight* ought to represent the relevance of the sentence given the question and considers TaM WFs only, *temperature* can be understood as an *indication* of the relevance (*weight*) and takes into account WFs from *both* ToM and TaM. The aggregation functions we use are the same for both, namely

- a *simple sum* of all facts contained in the relevant KB(s)
- a *maximum* of all facts contained in the relevant KB(s)
- a *thresholded sum*, which sums only WFs, which's value exceeds a set threshold. Currently we use the average value of all facts contained in ToM as the threshold for *temperature* and average of TaM for *weight*.

Formally for a given Knowledge Base K and list of candidate WFs C (obtained from the sentence following the described procedure), we can define the three aggregation functions formally as:

$$agr_{sum}(C,K) = \sum_{w \in C} value(w,K) \tag{3.1}$$

$$agr_{max}(C,K) = \max_{w \in C} value(w,K) \tag{3.2}$$

$$agr_{threshold}(C,K) = \sum_{w \in C, w \geq threshold(K)} value(w,K) \tag{3.3}$$

where

$$threshold(K) = \frac{\sum_{w \in K} value(w,K)}{|K|} \tag{3.4}$$

and $value(w,K)$ represents the value/relevance assigned to WF w in the Knowledge Base K. To wrap it up, there are the following differences between *temperature* and *weight:*

- the different way we obtain C
- the different K:
  - $K = TaM \cup ToM$ for *temperature*
  - $K = TaM$ only for *weight*
- and the different purpose of use

There are two base classes of sentences which may yield high one. The sentences from the first class obtain their value from few highly relevant WFs, while the other one aggregates it from many WFs of lesser relevance. The avg_{max} considers the first class only, while the $avg_{threshold}$ strives to achieve that only reasonably relevant facts are considered. The avg_{sum} does not perform discrimination, and makes possible for a sentence to score high even though it contains only (many) low relevance facts.

The next largest unit of text we handle are *text snippets.* Snippets are small sets of sentences, in our experiments two to five sentences long. The value (both *weight* and *temperature*) of a snippet is calculated as an simple average of the the values of the sentences it contains. Formally for a snippet S and Knowledge Base K :

$$value(S,K) = \frac{\sum_{s \in S} agr_*(candidates(s),K)}{|S|} \tag{3.5}$$

here s stands for sentence, agr_* can be replaced by any aggregation functioned introduced above and $candidates$ represents a procedure of obtaining WFs for a given sentence.

3.5 Selection of the Best Snippets

Once again, our snippet selection procedure is straightforward. This is partially design feature, as it is still part of the prepossessing before parsing (and has to be fast). It exhibits two features worthy of notice

- it respects the Wikipedia page structure. That is the best snippets are selected based on the sections of the Wikipedia article, and the (snippet) Candidate Lists of subsections are merged to obtain a single CL for their super section
- it allow certain flexibility in the size of returned snippets

The whole algorithm is governed by two parameters: the (desired) snippet size (*size*) and the (maximum) number of snippets returned (per Wiki page and its section, denoted by *count*). In the base case (the section does not contain subsections) we simply look at sequences of consecutive sentences of size in the interval $[\lfloor \frac{size}{2} \rfloor, size]$ (preferring the longer ones), and return the (at most) count ones with highest value/average as defined in Equation 3.4. In the recursive case (section with subsections) we first retrieve a CL for the text passages not included in any subsection, next we retrieved the CL for the subsections, merge them all into one list and select the at most *count* of them with the highest value as the output. The complete process is described in algorithm 3.3

We select snippets of consecutive sentence in order to maintain some context. As SAAS is data driven, the sentences in the context can lead us to discovery of new related WFs and thereby improve our KB. Our experiments with Shalmaneser shown as that we can parse around 30 sentences in reasonable time. Originally we have used values 3 for *count* and 10 for *size.* However it has often happened that the a large snippet has contained only one WF with high value. To avoid this we use the setting *count* 6, *size* 5. Note that this still enables us to select larger snippets by selecting to shorter ones next to each other. By using the structure of Wikipedia it very likely that the selected sentences speak about the same topic and are closely related (as their belong to the same section).

Algorithm 3.3 Best Snippet selection

```
if the section does not contain subsections
        initialize and empty CL

        repeat
          repeat
                 move index i from the begining of the section
                 to its the end

                 j = the count of subsequent sentences which are
                     not used by any snippet already in CL

                 if j>(size -1)
                    calculate the the average of size sentences
                        starting with i and consider it a candidate
                        for selection

                 else if j>(size/2-1)
                        calculate the the average of j sentences
                        starting with i and consider it a candidate
                        for selection

          until the end of sentence is reached or
                there are no suitable candidates available

          add the best candidate from this iteration to the CL

        until we have count candidate or we could not find a
        candidate in the previous iteration

        return CL

else if section does contains subsections

        parent CL = get best snippets from the text
                out of all subsection

        for all subsections
                add your best snippets CL to the
                parent CL

        select at most count best snippets
        from parent CL as the CL for this section
```

Chapter 4

Relevance assignment and propagation

In the previous chapter we have described how we discover new WikiFacts and how we can use the relevance values of the known facts to assign values to sentences and snippets. In order to start up the system, two additional mechanisms necessary. The first one initializes the (target) KB while other one is necessary to propagate relevance from the known facts to the newly discovered ones. This topics will be covered in this chapter. We will first describe two approaches to initiate the TaM, one is automatic and other is semi-automatic and uses human tagged data to enhance the automatic one. Next we shall proceed towards the relevance propagation mechanism and conclude with a full summary of our system.

4.1 Automatic TaM initialization

To initiate TaM we have to do perform two different tasks identify a initial set of WFs and assign an initial relevance value to them. The task of discovering WFs was extensively covered in the previous chapter. Thus we will just run the parser on the question sentences and use Algorithm 3.1. We have considered two features, which may indicate relevance of the discovered WFs:

- the number of words contained in the address. The intuition behind this measure is that the more words an expression contains the more specific it is.

Table 4.1: An example of TaM initialization

WF	weight
Cuba	200
Angola	200
The_end	133
Combat_support	133
Presence	66
Role	66
Modern	66
Significant	66

- the frequency of a WFs in the topic sentences. The more relevant the WF is to the topic, the more often it will appear.

Formally we assign an initial weight w to a WF f for the topic T based on the following formula:

$$w(f,T) = lengt(f) * (2 * count(f,T) - 1) \tag{4.1}$$

Thus we count every but the first occurrence of f in T twice and value f proportionally to the number of words it contains. We obtain the final weight w^* by normalizing the weights obtained from Expression 4.1

$$w^*(f,T) = \frac{w(f,T)}{\sum_{f \in T} w(f,T)} * bank$$

Bank is an normalization constant to which value of all WFs from the initial Target base should add. Currently we use 1000. Albeit simple, the algorithm produces quite reasonable outputs. For instance for the following topic

> The analyst is interested in Cuba's modern role in Angola . Despite the end of combat support , is Cuba still a significant presence in Angola ?

the KB from Table 4.1 is produced. It obviously helps a lot when a topic is longer and more descriptive as we have more information to work with. Some shorter topics, such as

Are there U.S. counter-drug efforts taking place in Ecuador ?

provide us very little information to work with. Take note that while it correctly identifies the most important terms ('Cuba', 'Angola') it also assigns high value to an arbitrary expression ('The end') as well as fairly generic terms ('role','modern'). We have also ommited by purpose terms, which are characteristic to the way the questions in our datased were phrased. For the purpose of TaM initiation the words 'Analyst','interested','like' and 'know' were ignored.

4.2 Semi-automatic TaM initialization

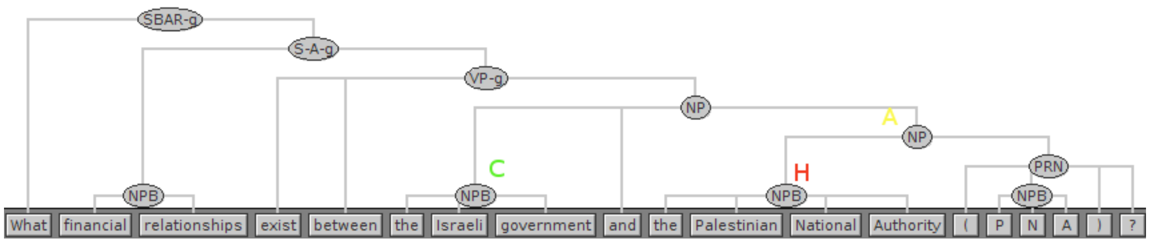
Our test topics were obtained from three different TREC Tasks, the 2005 Relationship tasks and ciQA (complex interactive Question Answering) tasks from TREC 2006 and 2007. The latter two provide additional information by labeling the entities involved in the relation. We decided to use this information in some of our experiments. The procedure for enriching the TaM is following:

1. First we run the automatic initialization
2. Then we take the entity information from the ciQA data, and distribute the *bank* evenly among all the provided entities.
3. This new WFs are added to TaM. If the TaM already contained some of them, their values are simply overwritten.

4.3 Transfer of Relevance

The knowledge is always transferred from the more relevant fact (head) to the less relevant one (child). The child can only gain relevance, if its connection to the head is strong enough. The head does not loose any of its relevance and is not influenced by the procedure. The strength of the the connection depends on a parse tree property which we call the Difference of Ancestry(DOA). First each of the WFs is associated with a non terminal node N (its covering node) in the parse tree, which fulfills the two following properties:

Figure 4.1: The Covering Node and DOA



The covering node for 'Israeli Goverment' is marked by a green C (child), Palestinian National Authority is covered by the NPB marked with the red H (head). The only different ancestor between the head and the child is the NP next to the yellow A. Thus the DOA is exactly one.

1. Each word in the WF is a descendant of N

2. For none of the children of N the previous property holds. That is the words from WF are descendants of at least two different children of N.

Let us illustrate this on a concrete example of parsing tree in Figure4.1

The covering nodes for the WFs could be obtained by a simple bottom-tree searching algorithm. However this is not necessary as we can mark the nodes already in the WF detection procedure described in the previous chapter. Let us define DOA formally. First we will need the notion of ancestry set, which is exactly what it sounds like: the set of all ancestor for the given parse tree T and a node N. Thus for a parse tree T, a head H and child C we define DOA as

$$ancestry(N,T) = \{set\,of\,all\,ancestrors\,of\,N\,in\,T\} \tag{4.2}$$

$$DOA(H,C) = |ancestry(H) \setminus ancestry(C)| \tag{4.3}$$

Thus siblings node have the least DOA, namely 0. The two covering nodes from our example have a DOA of 1 (where 'Palestinian National Authority' is the head). The DOA of 'exist' and H would be 2 and so on. The intuition behind this measure is straight forward, siblings have most in common as they are linked together by their parent. The more different parents nodes H has, the weaker the connection between H and C becomes. Note that DOA is asymmetric, that is:

$$DOA(H,C) \neq DOA(C,H)$$

The reason for this is that we regard H as the source of the knowledge. As H and its sibling are in some sense arguments of their parent, the relevance of H has direct influence on the siblings. As the distance from H to the 'least common parent' (LCP) increases, the connection becomes more uncertain. This is partly because of more nodes sharing the same distance to H that C does, and it is unclear which of them is more connected to C than the others. However C may by embedded in a more complex structure and thus its distance to LCP may not be that indicative. We adopt the assumption the child WF represents the whole subtree under C, as it is the least subtree where the WF is in its completeness. If there are multiple WFs covered by C, the process is repeated for each one of them, and each one is suggested the same value. This is once again a crude simplification, but the relevance is essentially transferred to C, and we do not now any way have to decided which of the WFs under C is more important than the others.

Based on DOA, multiple functions for transferring knowledge are possible, they should obviously decrease with DOA though. We have investigated two transfer functions a *linear* one and *an exponential* one. Also note that each node in the tree has some connection to every other node (in the worst case it goes through the root node). Therefore we would like our transfer function to decrease fast with the increase of DOA, so we do not assign a lot of relevance to a node based on an arbitrary connection. To reinforce this we set the transfer function to 0, whenever DOA is greater then 5 (we came up with this value by looking onto the parsed test trees. Obviously a more sophisticated approach would be helpful). Thus our two transfer functions are defined as follows.

$$transfer_{lin}(w_f,C) = \frac{w_f}{DOA(H,C)+1} if DOA(HC) \leq 5$$

$$transfer_{exp}(w_f,C) = \frac{w_f}{2^{DOA(H,C)}} if DOA(HC) \leq 5$$

Algorithm 4.1 Transfer of Relevance

```
for a given parsed sentence s
    for WF f from s
        iterate over all WFs g from s
            such that w_g< w_f
            calculated DOA(f,g)
            w_g'=transfer(w_f,g)
            if w_g'>w_g
                update w_g to w_g'
```

$$transfer_{exp}(w_f,C) = transfer_{lin}(w_f,C)\, if\, DOA(HC) > 5$$

where w_f is the value of the WF fact f which is covered by H.

None of these functions is cumulative. That is a WF g may set its value to $w_g = transfer_*(w_f,C)$ based on its position to w_f in the tree or keep its old value. Its value does not increase for each more relevant node w_f it is associated with. This has multiple advantages such as the convergence to a certain value (instead of increasing infinitely by finding new connections or being connected to the same facts in different sentences), and the guarantee that an fact with a high weight occurred sometimes very close to some highly relevant (most likely TaM) fact.

On the other hand, an additive transfer function would strengthen the connection with repeated co-occurrence, which can also indicate relevance. However we would have be cautious to prevent that myriad of connections to irrelevant facts out weight few connections to highly relevant facts. Because of the mentioned complications, the design of an additive function remains out of our reach for now and we use just the non additive ones.

The transfer functions are applied to the parsed sentences in the following manner (Algorithm 4.1)

Based on such updates f may be moved from ToM to TaM, if the w_f exceeds the average weight of all fact from TaM (in the time of addition, after next updates of TaM the average may increase and f may find itself bellow the threshold again, but remains in TaM nonetheless). Subsequently f may influence the the final evaluation of the returned snippets.

4.4 The detailed view on SAAS life Lifecycle

As we have discussed all the necessary details by now, we may present a more detailed version of Algorithm 1.1

Algorithm 4.2 Detailed SAAS life cycle

```
until the terminate condition is met
          select the followable WF f
      with the highest w_f from KBs

          retieve the article a
      corresponding to f

          extract the section
            structure of a

          calculate the temperature
            for each sentence of a

          retrieve the count snippets
            from a and parse them

          detect the WFs in parsed sentences
            calculate the connections
            ad new WFs and update values
      of the existing one based on
            connections

          after all snipets have been processed
            move the the WFs exceeding
            the treshold from ToM to TaM

          calculate the value of the
            parsed snippets based on TaM

          return the evaluated snippets

back to the start of the cycle
```

Chapter 5

Experiments

Before we discuss the results of our experiments themselves we should devoute some time to discussion on the length of WFs retrieved

5.1 The length of WFs

The reader may remember that in Algorithm 3.1 we resort to a pretty exhaustive search. At the root node we start with the window covering the whole sentence and decrease its size until we reach the size of the least CWS among its children. Naturally the question arises whether such an approach is necessariy, as it is extremely unlikely that an WF will cover the whole sentence. In fact the distribution of WFs in the final ToM follows the pattern obvious in the Table5.1

While the values differ from run to run, the trend remains the same among all experiments performed. There is a couple of hundreds one word WFs, tens of two word WFs and

Table 5.1: Distribution of WFs in a final TaM based on word length

Length	Frequency	Length	Frequency
1	119	4	1
2	24	5	1
3	6	>6	0
Total	151		

towards the length 5 or 6 the frequency is zero. This would justify usage of a maximum windows size of value five in Algorithm 3.1. We have run some preliminary experiments with this setting but did not experience a noticeable speed up (in terms of number of page visited before time out). It seem to be the cases, that although the search of WFs is exhaustive, the operations it performs are lightweight compared to parsing of sentences. The latter operation has more influence on the performance as the prior one.

5.2 Evaluation of experiments.

All experiments discussed here were performed with the following joint parameters:

- timeout = 1 hour
- snippet size = 5 (minimal 2)
- snippet count = 6

Thus we have varied the:

- aggregation function (*max, sum , sum threshold*)
- connection function (*linear* or *exponential*)
- usage of initial TaM enhancement

Unfortunately our system did not perform well enough to be evaluated in terms of *precision*, *recall* or *F meassure*. In any of the measures the system would score 0. The reason for this, is that while the system could find facts relevant to one or another entity (as well as same related to some words not relevant), it has never managed to find snippets connecting them. However it is fair to note that it had very limited possibilities, as it could visit only very few pages as detailed in Table expressing the number of tables visited per run and per topic.

There seem to be some trends visible from Table 5.2, in the top half (topics originating from TREC 05, no enhancement possible) the linear function visits considerable less pages than the exponential ones, while in the bottom (Enhanced topics) the difference is less apparent.

Table 5.2: Number of pages visited

	lMax	lSum	lTr	eMax	eSum	eTr
2	5	5	5	8	8	9
5	6	6	6	9	9	11
7	4	5	5	10	8	9
8	4	4	5	8	7	8
26	5	7	3	9	8	8
29	4	8	8	8	8	8
32	5	5	6	6	6	6
38	8	8	8	8	8	9

The columns represent the various experiment settings, the prefixes 'l' and 'e' represent the two connection functions while the suffixes represent different aggregation values. Last four topics (26-34) where run on topics enabling TaM Enhancement, which was used in all the experiment. The first four topics do not provide enhancement information.

In fact this is attributed to the topics themselves. Topics 32 and 38 covers connection of pharmaceutical companies to universities and psychological and emotional problems connected to obesity. Both topics are extensively covered in Wikipedia and each run returns articles highly relevant to some of the subtopics (pharmaceutical industry controversies','university','statistical testing' and 'psychology','obesity' respectively). Most likely all of the runs can discover early on the same set of relevant articles and extract the same relevant data from them, which smoothes out the difference between the runs. This hypothesis is also supported by the fact that in both topics the first pages visited provide the most relevant snippets.

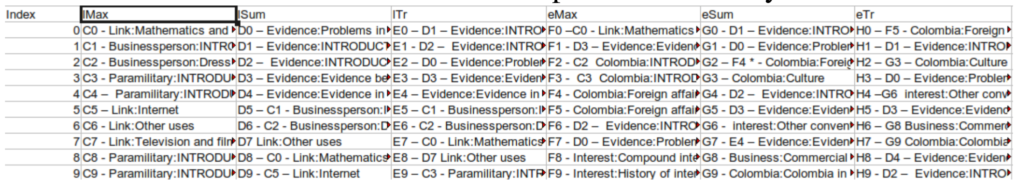
In topic 29 the same is the case, but for a diametrically different reason. The topic is concerned with human trafficking from China to US. However it uses the 'human cargo' expression to describe this process, and the Wikipedia article with this name refers to a Canadian documentary series (the lMax run performs so poorly because it picks up 'Canada') about people trafficking. With this valuable link gone, SAAS starts to investigate the economy, history and geography of this countries and adds 'Russia' as another big word player. Only the two exponential runs (eSum and eTr) maintain some connection to the original question and investigate the topic of 'transport'.

Table 5.3: TaM Enhancement vs. not Enhancement

Topic	LinEnh	LinNo	ExpEnh	ExpNo
26	7	8	8	8
29	8	10	8	10
32	5	6	6	8
38	8	8	8	10

Investigating the effect of TaM enhancement for sum aggregating function.

Table 5.4: Topic 8 - Case study



Index	lMax	lSum	lTr	eMax	eSum	eTr
0	C0 - Link:Mathematics and	D0 – Evidence:Problems in	E0 – D1 – Evidence:INTRO	F0 –C0 - Link:Mathematics	G0 - D1 – Evidence:INTRO	H0 – F5 - Colombia:Foreign
1	C1 - Businessperson:INTRO	D1 – Evidence:INTRODUCT	E1 - D2 – Evidence:INTRO	F1 - D3 – Evidence:Eviden	G1 - D0 – Evidence:Problem	H1 – D1 – Evidence:INTRO
2	C2 - Businessperson:Dress	D2 – Evidence:INTRODUC	E2 – D0 – Evidence:Problem	F2 - C2 Colombia:INTROD	G2 – F4 * - Colombia:Foreig	H2 – G3 – Colombia:Culture
3	C3 - Paramilitary:INTRODU	D3 – Evidence:Evidence be	E3 – D3 – Evidence:Eviden	F3 - C3 Colombia:INTROD	G3 – Colombia:Culture	H3 – D0 – Evidence:Problem
4	C4 – Paramilitary:INTRODU	D4 – Evidence:Evidence in	E4 – Evidence:Evidence in	F4 - Colombia:Foreign affai	G4 - D2 – Evidence:INTRO	H4 –G6 interest:Other conv
5	C5 – Link:Internet	D5 – C1 - Businessperson:I	E5 – C1 - Businessperson:I	F5 - Colombia:Foreign affai	G5 - D3 – Evidence:Eviden	H5 - D3 – Evidence:Evidenc
6	C6 - Link:Other uses	D6 - C2 - Businessperson:D	E6 - C2 - Businessperson:D	F6 - D2 – Evidence:INTRO	G6 - interest:Other conven	H6 – G8 Business:Commerc
7	C7 - Link:Television and film	D7 Link:Other uses	E7 – C0 - Link:Mathematics	F7 - D0 – Evidence:Problem	G7 - E4 – Evidence:Eviden	H7 – G9 Colombia:Colombia
8	C8 - Paramilitary:INTRODU	D8 – C0 - Link:Mathematics	E8 – D7 Link:Other uses	F8 - Interest:Compound inte	G8 - Business:Commercial	H8 – D4 – Evidence:Eviden
9	C9 - Paramilitary:INTRODU	D9 - C5 – Link:Internet	E9 – C3 - Paramilitary:INTR	F9 - Interest:History of inter	G9 - Colombia:Colombia in	H9 - D2 – Evidence:INTRO

The truth is that when not using enhancement we can process more pages then when we do (See Table 5.3). However as the most relevant snippets are discovered early on, it does not influence the snippets returned.

We cannot explain the reason of this for now. However a starking performance difference between linear and exponential function can be observed in some TREC 2005 (which seem to be harder than the other ones based on our results) topics . We will exemplify on Topic 7 investigating the links of Colombian businessman to AUC paramilitaries (see Table 5.4).

The case study shows that the linear function tends to drift away because of the word ’evidence’ used in the question. The exponential ones are also influenced by this, but manage to keep more focus on ’bussines’ and ’Colombia’. Also the linear max setting seems to avoid drifting, by focusing on ’bussines’ and ’paramilataries’. This behavior of lin max was not observed in other drifting topics, but the exponential functions perform consistently better in drifting topics than the linear ones. Specifically max exp and max threshold outperforms the exp sum. Also these to functions happen to be the only one which discover Cuba’s (historical) involvement in Angola in Topic 8 which is the only connections discovered between two entities involved.

We assume that exponential functions are less prone to drift then linear, because the are more conservative by the value transfer, and do not promote weakly connected facts that easily than the linear ones. To this end they also assign greater importance to DOA distance between head and child in the connection.

Chapter 6

Conclusion

In this thesis we have presented a novel perspective on the field of Question Answering. Instead of viewing it as a task of searching text passages in documents, we transform it to the task of an Intelligent Agent accumulating knowledge. To this end it builds a sophisticated Word Model, which it uses to detect promising sources of new valuable information. During the process of building a Knowledge Base - Word Model for the topic it also gathers facts relevant to the question. Thus it is consistently seeking for new areas and perspectives on the areas already investigated.

We have called this agent a Single Agent Answering system, and it interacts with the environment of Wikipedia, the online encyclopedia. By interacting with Wikipedia it gains new information about the topic and may reconsider the relevance of already known facts. To achieve this, we have developed a mechanism to obtain knowledge from Wikipedia in the form of WikiFacts and assign relevance value to them. We have also designed a mechanism to transfer the relevance among WFs based on their syntactic connections occurring in Wikipedia. Multiple function for governing both mechanisms were designed.

While the system behaves reasonably, it performs poorly. One reason for this is that as an data driven system, it may require more time to gain a good KB. We have also observed the problem of 'drifting', which is the phenomenon when general facts receive artificially high connection, based on rather arbitrary connections. We have discovered that the exponential connection function is less prone to this behavior than the linear one. Also the exponential connection function coupled with thresholded sum and maximum aggregation function, tends to outperform exponential connection sum aggregation function setting.

Thus we have laid a solid base in further research in (Multi) Agent based questioning system. However a lot of work needs to be done to find better connections and aggregation functions, as well as running more experiments on the existing ones.For know SAAS is far behind the state of art QA systems, but the use of stochastic search, Multi Agent methods, or simple more time may or may not change it.

Appendix A

Setting up the Wiki database

A.1 Data

In our experiments we use local copy of the Wikipedia database. This decision enables us to use WikiRelate! as an interface to Wikipedia. Our extensive usage of Wikipedia, especially in checking for WikiFacts as described in 3, makes the usage of the 'live' Wikipedia impractical, if not impossible. Due to the frequency of our requests, we would probably get black listed by Wikipedia very soon, while with local copy the verification of WFs boils down to checking a database entry.

All reported results were achieved on the data dump from 26.10.2010 available from

```
http://download.wikimedia.org/enwiki/
```

As this site lists only a couple of most recent dumps, the dump we have used is not available there anymore. We however do not depend on the specific version and any other version may be used (though the wikisimilarity.properties file in Wikirelate/config/ has to be modified). In the case of using an another version all the time and disc space usage estimates may be quite inaccurate. Of all the files available in the dump, only two are needed:

- enwiki-your_version_date-pages-articles.xml.bz2 - this archive contains the database of articles and needs to be dumped to the MediaWiki mySQL database (by default called wikidb) following the process described bellow

- enwiki-20091026-categorylinks.sql.gz - builds up the category structure upon the pages. This file is imported to the wikidb *after* pages-articles have been dumped.

Note that dumping/importing both files is necessary for Wikirelate! to run, which in turn provides the Wikipedia back end for SAAS. Also take into account that while the two files were only 5.3GB a 0.54GB respectively, the populated database will take up around 53.8GB (it is possible to reduce this amount by about 5GB by not building up the indexes, see bellow).

A.2 Hardware and System Configuration

All experiments discussed in this work, as well as the setup described bellow, were run on Fujitsu Siemens Amilo Pa 1538 notebook equipped with a Turion 64 X2 TL-50 dual core processor and 2GB of memory. The operation system used was Ubuntu 9.10 (Karmic Koala). Thus the steps describe in this setup should work on this version without problems, and may require some modifications on other versions of Ubuntu and/or other Debian distributions.

A.3 Setting up MediaWiki

First you will have to install MediaWiki, which will set up the database for the Wikipedia data and provide a web browser based front end to access your local Wikipedia copy (we recommend only viewing, see A.4). As MediaWiki depends on the LAMP stack (Linux,Apache,mySQL,PHP), we need to set it up first. [1] In Karmic Koala the LAMP stack can be installed by running

```
sudo tasksel install lamp-server
```

Other methods of installation are discussed at

```
https://help.ubuntu.com/community/ApacheMySQLPHP
```

[1]Windows users may consider to follow the instructions at http://www.blindedbytech.com/2006/08/31/how-to-install-wikipedia-for-offline-access/

Upon finishing the installation of LAMP, you may proceed to install the MediaWiki as described at

```
http://www.mediawiki.org/wiki/Manual:Installing_MediaWiki_on
_Ubuntu_7.10_via_GUI_and_Synaptic
```

We would recommend you to follow the instruction on the above website an keep the default database settings. After the MediaWiki setup is successfully finished, you may proceed to dump the database.

A.4 Dumping the Database[2]

There are many ways how to dump the wiki data into wikidb. We have used the method using the MWDumper (the third party released from the page below) as described at

```
http://www.mediawiki.org/wiki/Mwdumper
```

In our case the dumping took about 86 hours, with the settings we shall describe bellow. It is also possible to do it "over night" (in 12 hours or less), if you remove *both* auto increments *and* indexes, but we have decided to keep the indexes for the sake of faster database access. It would be also possible to dump the database without indexes and recreate them afterwards, but this would take lot of time (possibly even more than the time spared at dumping) and disc space. We also did not restore the auto increment fields, as we do not intend to modify the data. Therefore we *do not recommend* to *modify* the wikidb data as it may bring the database into an inconsistent state. The following settings have proven to be necessary for a successful dump:

1. MediaWiki tables should use CHARACTER SET=binary, which should be the case by default

2. Use default-character-set=utf8, this is already included in the query we provide

[2]based on: http://www.mediawiki.org/wiki/Mwdumper; http://www.blindedbytech.com/2006/08/31/how-to-install-wikipedia-for-offline-access/; http://arunxjacob.wordpress.com/page/3/; http://getsatisfaction.com/luci/topics/wikipedia_dump_woes;

3. Make sure you use Sun's java instead of the default gcj java in Ubuntu, this can be done by running

   ```
   sudo update-alternatives --config java
   ```

   and choosing Sun Java provided that is installed

4. Increase the java heap size, already done in the provided query

   ```
   java -Xms128m -Xmx1000m
   ```

5. Modify the mySQL's my.cnf file and increase max_allowed_packet to

   ```
   max_allowed_packet=32M
   ```

   Be aware that there are multiple my.cnf files each with different priority, so make sure that change does really take place.

6. Most importantly you have to assure that the following tables are *empty* before the dump: *page*, *revision*, *text*. This is *NOT* the case by default. If you do not delete those pages the dumping will crush at the very end.

All this being done you can launch the MWDumper from a separate terminal window by calling

```
java -Xms128m -Xmx1000m -jar mwdumper.jar
--format=sql:1.5 enwiki-20091026-pages-articles.xml.bz2 |
mysql --default-character-set=utf8 -u wikiuser -p wikidb
```

Than just use the GUI to connect to the wikidb and select the file import (the pages-articles one). You can check your progress by tracking the number of dumped pages, in our case it was a bit more then 9 216000.

The following changes in my.cnf may speed up the process a bit

1.
   ```
   # Set buffer pool size to 50-80% of your computer's memory
   innodb_buffer_pool_size=256M
   innodb_additional_mem_pool_size=20M
   ```

2.
   ```
   # Set the log file size to about 25% of the buffer pool size
   innodb_log_file_size=64M
   innodb_log_buffer_size=8M
   ```

3. Disable the log_bin, should be the case by default

4. Remove the auto increments (and indexes if you dare, see the comment at the beginning of this section).

After done all of this, we can import the categories by calling

```
mysql -u wikiuser -p password wikidb <
        enwiki-20091026-categorylinks.sql.gz
```

Now you might access your MediaWiki through your web browser. You might receive blank pages and will probably see a lot of markup characters. These issues are resolved in the next section.

A.5 Polishing the Mediawiki[3]

If you receive a blank page, the chances are that there was an PHP error which was not displayed correctly. This can be fixed by adding the following line to your MediaWiki's LocalSetting.php

```
error_reporting( E_ALL );
ini_set( 'display_errors', 1 );
define( 'MW_NO_OUTPUT_BUFFER', 1 );
```

The error will be most likely related to a low memory limit. The following setup has worked fine with us

```
ini_set( 'memory_limit', '50M' );
```

To suppress the displaying of MediaWiki markup characters, you will need the to install some MediaWiki extensions, which are available at

[3]based on: http://swingleydev.com/blog/?tag=linux; http://www.gossamer-threads.com/lists/wiki/mediawiki/184140

```
http://www.mediawiki.org/wiki/Category:Extensions
```

You can search them through the search edit box, and each extension contains an installation guide (consist mostly of copying the extension file and editing the LocalSetting.php). We recommend the following ones to install

1. ParserFunctions
2. Cite
3. CategoryTree
4. WikiHero

Now your local copy should closely resemble the live Wikipedia, apart from the absence of images.

Appendix B

Setting up Shalmaneser

B.1 Obtaining the Software and Licensing Issues

As already described in Chapter 2, Shalmaneser [Erk and Pado (2006a)] can be downloaded from the project website[1] and can be used and developed free of charge for non commercial purposes. Although its modular design enables users to plug in their own modules, we decided to use the provided custom components and pre trained classifiers. Thus we have Collins parser [Collins (1997)] for parsing, Treetagger [Shmid] for lemmatiztion and TNT [Brants (2000)] for part-of-speech tagging. While these software components are available roughly under the same conditions as Shalmaneser itself, in the case of TNT some complications occur. In order to obtain the source code you have to fax a license agreement to Thorsten Brants (the author of TNT). As the provided fax address seems to refer to his former workplace, we would advise you to contact him at his email address provided at the project webpage.[2] The Shalmaneser team is well aware of this bottleneck and plans to switch to TreeTagger and OpenNLP MAXENT(instead of TNT and Mallet [McCallum (2002)]) in their next release. Despite of their willingness to share the most current (not official) version with us, it was not fully functional at the time of writing this thesis. However we would like to encourage you to get in touch with authors of Shalmaneser if you run into complications with obtaining TNT or any other difficulties. We were also unable to use the

[1]http://www.coli.uni-saarland.de/projects/salsa/shal/index.php?nav=download

[2]http://www.coli.uni-saarland.de/~thorsten/tnt/

newer versions of Mallet (2.0.5 at the time of writing) and would therefore suggest you to use version 0.4 might you experience the same.

B.2 Minor problems

B.2.1 The missing 'IN/that' POS tag

It may happen that the Collins parser encounters the 'IN/that' POS tag in its input. As the parser's grammar does not know such a tag, this will result in the following error:

```
Frprep: Parsing
...
Loaded grammar parser: sentence.c:112:
convert_sentence: Assertion '0' failed.
Aborted Frprep: Postprocessing SalsaTigerXML data
Writing /home/pistik/.shalmaneser/1266784815_84597/frprep/
0.xml
./Pkg/CollinsInterface.rb:127:in 'each_sentence':
Error: premature end of parser file! (RuntimeError)
from ./Pkg/TabFormat.rb:108:in 'each_sentence'
from ./Pkg/CollinsInterface.rb:113:in 'each_sentence'
```

To get around this, it is sufficient to edit the sentence.c file in Collins parser's program directory and put

```
if(!(t>=0)){
   if(!strcmp("IN/that", sentence->tags[i])){
            strcpy(sentence->tags[i], "IN");
            t = find_word(sentence->tags[i],&nt_lex);
            fprintf(stdout,"changed IN/that %s tag %s\n",
                        sentence->words[i],sentence->tags[i]);
   }
}
```

immediately in front of

```
if(!(t>=0)){
        fprintf(stdout,"TAG %i %s not found\n",
                i,sentence->tags[i]);
        assert(0);
}
```

Afterward it is necessary to rebuilt the Collins parser by running its Makefile (residing also in the program directory). It should not be necessary to set up Shalmaneser again.

B.2.2 Serialization errors

You are also likely the encounter the following the error message, if you try to use the pre-trained classifiers for ROSY.

```
java.io.InvalidClassException:
        edu.umass.cs.mallet.base.pipe.Target2Label;
local class incompatible:
        stream classdesc serialVersionUID = -8390758647439705273,
localclass serialVersionUID = -461155063551297878
at java.io.ObjectStreamClass.
        initNonProxy(ObjectStreamClass.java:562)
at java.io.ObjectInputStream.
        readNonProxyDesc(ObjectInputStream.java:1583)
...
```

The problem is that the provided classifiers are in fact serialized java classes, whose serialVersionUID does not match with the currently expected ones. An easy remedy is to force the local classes to use the saved UIDs by adding the line

```
static final long serialVersionUID = -8390758647439705273L;
```

into the Mallet class

```
edu.umass.cs.mallet.base.pipe.Target2Label
```

and

```
static final long serialVersionUID = -2670393659883800085L;
```

into

```
edu.umass.cs.mallet.base.pipe.TokenSequence2FeatureSequence
```

Following this modification you will have to both rebuild Mallet (run its Makefile) and setup Shalmaneser again(rerun setup.rb and apply the Mallet patch).

Appendix C

The Test Topics

Topic 2

The analyst is concerned with arms trafficking to Colombian insurgents . Specifically , the analyst would like to know of the different routes used for arms entering Colombia and the entities involved .

Topic 5

The analyst is interested in the South American drug cartel's ability to launder money . Specifically , the analyst would like information on ties between the cartels and banks in Liechtenstein .

Topic 7

The analyst is looking for links between Colombian businessmen and paramilitary forces . Specifically , the analyst would like to know of evidence that business interests in Colombia are still funding the AUC paramilitary organization .

Topic 8

The analyst is interested in Cuba's modern role in Angola . Despite the end of combat support , is Cuba still a significant presence in Angola ?

Topic 26

What evidence is there for transport of smuggled VCDs from Hong Kong to China? The analyst is particularly interested in knowing the volume of smuggled VCDs and also the ruses used by smugglers to hide their efforts.

Topic 29

What evidence is there for transport of human cargo from China to the (U.S.)? The analyst wants to know if there is evidence of transporting human cargo from China to the (U.S.) and where the ships arrive in the (U.S.)

Topic 32

What financial relationships exist between drug companies and universities? The analyst is concerned about universities which do research on medical subjects slanting their findings, especially concerning drugs, towards drug companies which have provided money to the universities.

Topic 38

What effect do psychological or emotional problems have on obesity? The analyst would like to know if obesity, when not genetic, is triggered by deep-seated emotional problems or depression. Specifically, does the problem vanish when the underlying cause has been determined?

Bibliography

A. A. S. a. M. A.Burchardt, K.Erk. Salto a versatile multi-level annotation tool. In *Proceedings of LREC 2006*, 2006.

D. Ahn, V. Jijkoun, G. Mishne, K. MÃŒller, M. de Rijke, and S. Schlobach. Using wikipedia at the trec qa track. In *Proceedings of TREC 2004*, 2004.

A.Mengel and W.Lezius. An xml-based encoding format for syntactically annotated corpora. In *Proceedings of LREC 2000*, 2000.

T. Brants. Tnt a statistical part-of-speech tagger. In *Proceedings of ANLP 2000*, 2000.

R. Bunescu. Using encyclopedic knowledge for named entity disambiguation. In *In EACL*, pages 9–16, 2006.

M. Collins. Three generative, lexicalised models for statistical parsing, 1997.

CraigMacdonald and IadhOunis. The trec blog06 collection : Creating and analysing a blogtest collection. Technical report, Department of Computing Science, University of Glasgow, 2006.

H. T. Dang, J. Lin, and D. Kelly. Overview of the trec 2006 question answering track. In *In Proceedings of the Text REtrieval Conference*, 2006.

H. T. Dang, D. Kelly, and J. J. Lin. Overview of the trec 2007 question answering track. In *TREC*, 2007.

K. Erk and S. Pado. Shalmaneser - a flexible toolbox for semantic role assignment. In *Proceedings of LREC 2006*, 2006a.

K. Erk and S. Pado. A powerful and versatile xml format for representing role-semantic annotation. In *Proceedings of LREC 2004*, 2006b.

E. Gabrilovich and S. Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *In Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1606–1611, 2007.

I. MacKinnon and O. Vechtomova. Complex interactive question answering enhanced with wikipedia. In *TREC*, 2007.

A. K. McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

E. M.Voorhees and H. T.Dang. Overview of the trec 2005 question answering track. In *Proceedings of the Fourteenth Text REtrieval Conference(TREC2005)*, 2006.

S. P. Ponzetto and M. Strube. Exploiting semantic role labeling, wordnet and wikipedia for coreference resolution. In *HLT-NAACL*, 2006.

S. P. Ponzetto and M. Strube. An api for measuring the relatedness of words in wikipedia. In *ACL*, 2007.

M. F. Porter. Snowball: A language for stemming algorithms. http://snowball.tartarus.org/texts/introduction.html, 2001.

N. Reiter. Framenet api. http://www.cl.uni-heidelberg.de/trac/FrameNetAPI, 2007.

H. Shmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of NeMLaP 1994*.

M. Strube and S. P. Ponzetto. Wikirelate! computing semantic relatedness using wikipedia. In *AAAI*, 2006.

M. Wooldridge and N. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

Abstrakt

V tejto diplomovej práci sme predstavili inovatívny prístup k Question Answering-u. Pozeráme sa na túto úlohu z perspektívy Inteligentného Agenta, ktorý sa usiluje vo svojom prostredí nájsť informácie dôležité k zodpovedaniu otázky, ktorá mu bola položená. Vď aka svojej interakcií s prostredím - internetovou encyklopédiou Wikipédiou - môže agent nadobodnúť nové poznatky alebo prehodnotiť dôležitosť už známych faktov.

Tiež sme odhalili niektoré problémy spojené s týmto prístupom a navrhli sme postupy ako ich riešiť. Z funkcí spojenia a funkcí hodnotenia ktoré sme navrhli sa exponenciálna funkcia spojenia spojená s maximovou hodnotovou funkciu alebo hodontovou funkciou medzného priemeru javia ako najživotaschopnejšie. Stále je však potrebný ďalší výskum a vyhodnotenie týchto funkcii

Dúfame že touto prácou podnietime vznik ďalších Agentových a Multi Agentových systémov, schopných zodpovedať otázky týkajúce sa vzťahov medzi entitami, ako aj iné zložité triedy otázok.