The QUANTUM question answering system

Luc Plamondon, Guy Lapalme
RALI, DIRO, Université de Montréal
CP 6128, Succ. Centre-Ville, Montréal (Québec) Canada, H3C 3J7
{plamondl, lapalme}@iro.umontreal.ca

Leila Kosseim*
Concordia University
1455 de Maisonneuve Blvd. West, Montréal (Québec) Canada, H3G 1M8
kosseim@cs.concordia.ca

Abstract

We participated to the TREC-X QA main task and list task with a new system named QUANTUM, which analyzes questions with shallow parsing techniques and regular expressions. Instead of using a question classification based on entity types, we classify the questions according to generic mechanisms (which we call *extraction functions*) for the extraction of candidate answers. We take advantage of the Okapi information retrieval system for one-paragraph-long passage retrieval. We make an extensive use of the Alembic named entity tagger and the WordNet semantic network to extract candidate answers from those passages. We deal with the possibility of no-answer questions (NIL) by looking for a significant score drop between the extracted candidate answers.

1 Introduction

We shall describe here our new question answering system called QUANTUM, which stands for QUestion ANswering Technology of the University of Montreal. QUANTUM was designed specifically for the TREC-X QA-track based on our experience with XR[3], which we used last year at TREC-9 [LKL00]. We shall introduce the architecture and the performance of the version used for the main task. Then, we explain how it was adapted to the list task. We did not participate to the context task.

*Work performed while at the University of Montréal.

2 Components of questions and answers

Before we describe QUANTUM, let us consider question # 302[1] and its answer shown in Figure 1. The question is divided in three parts: a *question word*, a *focus* and a *discriminant*, and the answer has two parts: a *candidate* and a variant of the question *discriminant*.

The *focus* is the word or noun phrase that influences our mechanisms for the extraction of candidate answers (whereas the *discriminant*, as we shall see in section 3.3.2, influences only the scoring of candidate answers once they are extracted). The identification of the focus depends on the selected extraction mechanism; thus, we determine the focus with the syntactic patterns we use during question analysis. Intuitively, the focus is what the question is about, but we may not need to identify one in every question if the chosen mechanism for answer extraction does not require it.

The *discriminant* is the remaining part of a question when we remove the question word and the focus. It contains the information needed to pick the right candidate amongst all. It is less strongly bound to the answer than the focus is: pieces of information that make up the question discriminant could be scattered over the entire paragraph in which the answer appears, or even over the entire document. In simple cases, the information is found as is; in other cases, it must be inferred from the context or using world knowledge.

[1]Whenever we cite a question from a TREC competition, we indicate its number. Questions 1–200 are from TREC-8, 201–893 from TREC-9 and 894–1393 from TREC-X.

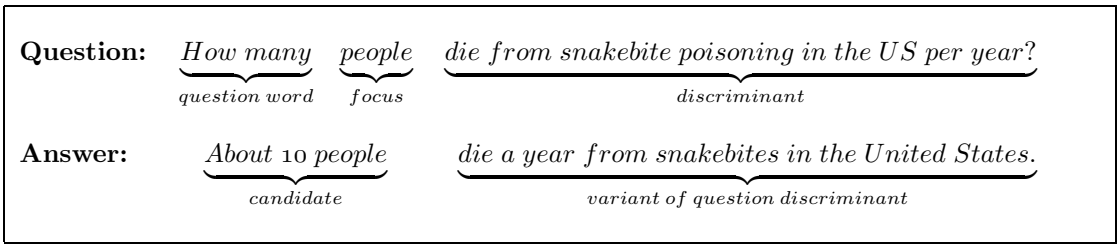

Figure 1: Example of question and answer decomposition. Question is from TREC-9 (# 302) and answer is from the TREC text collection (document LA082390-0001).

We shall use the term *candidate* to refer to a word or a small group of words, from the text collection, that the system considers as a potential answer to the question. For the purpose of TREC-X, a candidate is seldom longer than a noun phrase or a prepositional phrase.

In this article, the term *answer* designates the string that results from the expansion of a candidate to a 50-character string.

3 System architecture for the main task

The input for the QA main task is a question set and a text collection. The system must output a ranked list of five 50-character answers to each question. We describe the 5 steps that QUANTUM follows when performing this task.

3.1 Question analysis

To analyze the question, we use a tokenizer, a part-of-speech tagger and a noun-phrase chunker. These general purpose tools were developed at the RALI laboratory for purposes other than the TREC QA-track. A set of about 40 hand-made analysis patterns based on lexical form, grammatical and noun phrase tags are used to determine the most appropriate extraction functions to apply. Table 1 shows the 11 function we have implemented. Each function triggers a mechanism for the extraction of candidates in a passage that can involve the passage's syntactic structure or the semantic relations of its component noun phrases with the question focus. More formally,

$$\mathcal{C} = f(\rho, \varphi)$$

where f is the extraction function, ρ is a passage, φ is the question focus and $\mathcal{C}$ is the list of candidates found in ρ. Each element of $\mathcal{C}$ is a tuple $(c_i,\ d_i,\ s_i)$, where c_i is the candidate, d_i is the number of the document containing c_i, and s_i is the score assigned by the extraction function.

We observed that in most TREC-9 QA systems a class fits a particular type of entity that the system is able to identify: toponyms, proper nouns, animals, weights, lengths, etc. In order to pair a question with an expected type of entity, one needs to anticipate all possible question forms that could focus on this type of entity. This introduces a supplemental difficulty, given the large number of possible reformulations of a question.

However, a lexical and syntactic analysis of all possible forms of English questions — that are applicable to TREC — showed that the number of required search mechanisms is rather limited. By considering these mechanisms (our 11 functions) as classes, we facilitate the question classification task because the number of classes is small and because the classes are closely related to the syntax of questions. Even though the number of classes in such a function-based classification is smaller than in an entity-based classification, we can achieve the same level of precision by parameterizing our functions with the question focus when needed. The automated process of parameterizing a generic mechanism can suit questions about virtually any kind of entities, whereas an entity-based classification is limited to the entities it contains. At worst, the chosen function f and parameter φ could lead to a generic, non-optimal search. Yet the correct answer can still be retrieved.

3.2 Passage retrieval and tagging

The extraction of candidates is a time-consuming task. Therefore, we look for the shortest, albeit most relevant, passages of the text collection. We tried two different techniques: variable-length paragraphs retrieved with Okapi and fixed-length passages retrieved with our own algorithm based on last year's XR^3 system.

Extraction function	Example
definition(ρ, φ)	*(# 897) What is an* ***atom****?*
specialisation(ρ, φ)	*(# 910) What* ***metal*** *has the highest melting point?*
cardinality(ρ, φ)	*(# 933) How many* ***Great Lakes*** *are there?*
measure(ρ, φ)	*(# 932) How much* ***fiber*** *should you have per day?*
attribute(ρ, φ)	*(# 894) How* ***far*** *is it from Denver to Aspen?*
person(ρ)	*(# 907) Who was the first woman to fly across the Pacific Ocean?*
time(ρ)	*(# 898) When did Hawaii become a state?*
location(ρ)	*(# 922) Where is John Wayne airport?*
manner(ρ)	*(# 996) How do you measure earthquakes?*
reason(ρ)	*(# 902) Why does the moon turn orange?*
object(ρ)	Default function

Table 1: The analysis of a question determines which function to use for extracting candidates. An extraction function is a generic search mechanism, sometimes parameterized by the question focus φ. Examples of classified questions are provided with their focus in boldface.

Category	Weight
Quoted strings	20
Years	10
Named entities	10
Noun phrases of more than one word	10
Capitalized nouns	2
Common nouns	1

Table 2: Question keywords are fitted into one of these categories and their weight is set accordingly.

3.2.1 Variable-length passages with Okapi

Okapi is an information retrieval engine that has the ability to return relevant paragraphs instead of whole documents [RW98]. We feed it with the question as a query and we set it up so that it returns 30 one-paragraph-long passages (the average length of a passage, or paragraph, is 350 characters).

3.2.2 Fixed-length passages

We also tried our own passage retrieval algorithm in a different run. We first build a list of keywords from the question. Keywords are fitted into the categories listed in Table 2 and a weight is attached accordingly to each of them.

Then, the best 200 documents returned by the IR engine PRISE (provided by NIST to all participants) are scanned for the keywords. The 250-character-long strings centered around every keyword occurrence constitute our fixed-length passages. The score of a passage is the sum of the weights of all the keywords it encloses. Passages from the same documents that overlap by more than 125 characters are discarded if they both have the same score. The remaining passages are ranked according to their score and the 50 best ones are kept.

3.2.3 Passage tagging

Once we have found the most relevant passages, we run our tokenizer, our tagger and our noun phrase chunker on them because those information are needed by the candidate extraction functions. We also feed them into a named entity extractor. Last year, we used hand-built regular expressions for named-entity tagging, but this year, we used the freely available version of the Alembic Workbench system[2] developed at Mitre Corporation for the Message Understanding Conferences (MUC) [ABD+95]. Table 3 lists the types of entities that Alembic can identify.

3.3 Extraction and scoring of candidates

3.3.1 Extraction

Given the extraction function f chosen after question analysis, the question focus φ and a set of tagged passages ρ_j, candidates c_i are extracted along with their document number d_i and their score s_i (see section 3.1). During this phase, we seek the best recall rate possible, no matter whether candidates are cited in a context that matches the question discriminant. Table 4 shows some examples of what extraction functions look for.

[2]Downloaded from www.mitre.org/resources/centers/it/g063/workbench.html

Entity	Example
<PERSON>	Persons (*G. Washington*, *Mr. George Washington*), titles (*the President*)
<ORGANIZATION>	Full name of organizations and acronyms (*NATO*, *Congress*)
<LOCATION>	Toponyms (*Lake Ontario*, *North Africa*)
<DATE>	Dates (*Sep. 12, 1943*), years (*1983*), months (*February*), days (*Monday*)
<TIME>	Times (*23:03:12*, *4 a.m.*, *8 o'clock*)

Table 3: Named entities recognized by Alembic. An exhaustive description of these categories can be found in [ABD+95].

Extraction function	Example of criteria
definition(ρ, φ)	Hypernyms of φ
specialisation(ρ, φ)	Hyponyms of φ
cardinality(ρ, φ)	Pattern: NUMBER φ
measure(ρ, φ)	Pattern: NUMBER UNIT φ
attribute(ρ, φ)	Various patterns
person(ρ)	<PERSON> entities
time(ρ)	<TIME> entities
location(ρ)	<LOCATION> entities
reason(ρ)	Not implemented for TREC
manner(ρ)	Not implemented for TREC
object(ρ)	Any noun phrase

Table 4: Sample of extraction mechanisms for each extraction function.

3.3.2 Scoring

The final score of a candidate is the sum of three partial scores: the extraction score, the passage score and the proximity score.

Extraction score The score s_i awarded to a candidate by an extraction function is called the *extraction score*. It depends on the technique used for extracting a candidate. Typically, we award a higher score to a candidate extracted by the named entity extractor or by hand-made patterns; a candidate extracted because it satisfies some WordNet hypernym/hyponym relation is given a lower score because of its higher risk of introducing noise.

Passage score While the extraction score is concerned only with the form and type of a candidate, the *passage score* attempts to take into account the supplemental information brought by the question discriminant. This score is the one given to a passage during its retrieval by either Okapi or our fixed-length passage retrieval algorithm. Since the question discriminant is likely to appear in the text under a slightly different form and to be scattered over several sentences around the sought candidate, we believe that an IR engine is the best tool for measuring the concentration of elements from the discriminant in a given passage.

Proximity score The combination of the extraction score and passage score favours candidates that have the type we looked for and that are related to the question context. We also give a *proximity score* to candidates contiguous to noun phrases that contain a question keyword. By *contiguous*, we mean that they are not separated by another noun phrase. This way to measure proximity is rather crude and its effectiveness is still to demonstrate; therefore, we choose a relatively low proximity score to minimize its influence. At least, this score is helpful to break a tie between two candidates.

3.4 Candidate expansion to 50 characters

We expand a candidate by taking the 50-character document substring that is centered around it. Then, we cut off truncated words at both ends, which allows us to shift the substring to the right or to the left so that the new 50-character string contains the maximum number of complete words. The purpose is to maximize the chances that the string contains the correct candidate in the unfortunate case where QUANTUM would have guessed wrong. The effect of chance is not to be neglected since we measured a MRR score improvement of 0.3 with our last year system, XR^3, only by expanding candidates.

Candidate expansion takes place in conjunction with a redundancy elimination process. We begin by expanding our very best candidate. Then, the second best candidate is expanded only if it does not appear in the first answer. The third candidate is expanded only if it does not appear in a previous answer, and so on until we have the desired number of answers. To keep a better diversity of candidates, we eliminate duplicate candidates even if they do not come from the same document, even at the risk of eliminating a

supported occurrence of a candidate to the benefit of an unsupported one. However, we find such a probability to be very low since only 1.5 % of the TREC-9 question set had a correct but unsupported answer found by the system we used last year.

3.5 No-answer questions

Until now, we have assumed that the answer of the question could be found in the text collection. However, this might not be the case: a *NIL* answer may thus be the correct answer indeed. To deal with this, we examine our candidates to determine whether a *NIL* answer should be amongst our 5 suggestions of answers and, if so, at what rank.

Since score scales differ from question to question (particularly when different extraction functions are used), we cannot use a unique score threshold below which we can say that a *NIL* answer is more likely than a low-score answer. We have used a threshold that depends on the score drop between two candidates we have normalized it so that it can be applied to the candidates of any question.

Let a_i be the answer at rank i and δ_i^{i+j} be the score difference between a_i and its j^{th} successor a_{i+j}. We compute the normalized score drop Δ_i between a_i and a_{i+1} in the following manner:

$$\Delta_i = \frac{\delta_i^{i+1}}{\delta_i^{i+4}} = \frac{s_i - s_{i+1}}{s_i - s_{i+4}}$$

where s_i is the score of a_i. Our choice to normalize over a 5-rank score difference δ_i^{i+4} is arbitrary, though our experiments showed that the following observations still hold for normalization over different intervals.

We ran QUANTUM on the TREC-9 questions and kept all answers that were extracted (not only the 5 best). We then applied the official correction script to spot the rank r of the first correct answer (when found). We computed Δ_r to measure the normalized score difference between a correct answer and its successor, which was a wrong answer. We also computed the average Δ_i for any pair of answers. We found that the score drop between a correct answer and its successor is slightly higher than the average score drop between any answer pair. Table 5 shows that this is true for different normalization intervals.

Having that in mind, we applied the following reasoning. Suppose we have two ranked, different answers, a_i and a_{i+1}. For simplicity, suppose also that their scores are different. Since a_{i+1} has a lower score than a_i, we assume that a_{i+1} has a lower probability to be correct than a_i. If the score drop Δ_i between the two is above average, we have an additional hint that a_{i+1} is incorrect. When Δ_i reaches a threshold Δ_t, we consider that a NIL answer is more probable than a_{i+1} (and than any other a_{i+j}, where $j < 1$). Thus, we keep a_i at rank i but as a second choice, we would rather say that there is no answer than submit a_{i+1} and we insert a *NIL* between the two.

Normalization interval	Δ_r	Δ_i
δ_i^{i+4}	33 %	29 %
δ_i^{i+3}	40 %	35 %
δ_i^{i+2}	56 %	50 %

Table 5: The normalized score drop Δ_r between a correct answer and its successor is slightly higher than the average normalized score drop Δ_i between any answer and its successor, regardless of the interval. Results were obtained by running QUANTUM on the TREC-9 question set.

If the system finds less than 5 answers and no score drop justifies the insertion of a *NIL*, we add a *NIL* answer after the last answer found.

The Δ_r we computed previously between a correct answer and its successor is a lower bound for a threshold on Δ_i above which a *NIL* is inserted. We set this threshold Δ_t experimentally by creating a set of 400 questions in which we knew that 5 % of questions had no answer in the text collection (the remaining questions were from TREC-9). We then chose the threshold value Δ_t that maximized the overall MRR score on this new question set. We obtained a maximum MRR score of 0.257 with $\Delta_t = 80$ %. However, we are aware that this threshold may not be optimal if the proportion of no-answer questions in a set is not 5 %.

Our technique based on score difference suffers a major handicap: it does not allow for the insertion of a *NIL* at rank 1 because Δ_0 does not exist. The only possibility for QUANTUM to put a *NIL* at rank 1 is when no answers at all are extracted. We believe that this situation arise far less often than no-answer questions are encountered in a question set because since our extraction functions were designed to achieve a high recall rate, they are more permissive than restrictive.

3.6 Final answer

The *final answer* is defined as the rank of the answer the system would give if it were allowed only one suggestion. It can be a number from 1 to 5 or the string *UNSURE*. Since our most confident answer is always put at rank 1, the *final answer* field is set to

Run	Confident	Correct
UdeMmainOk80	456 (92 %)	56 (12 %)
UdeMmainOk60	456 (92 %)	51 (11 %)
UdeMmainQt80	456 (92 %)	39 (8 %)

Table 6: Number of questions QUANTUM was confident for (out of 493 questions) and number of confident questions to which QUANTUM found a correct answer.

1 for every question. However, when QUANTUM is unable to analyze correctly a question and thus relies on the default function to find candidates, we set *final answer* to *UNSURE*. Thus, in our system, the *final answer* indicator is merely a binary flag to express confidence. Table 6 shows QUANTUM confidence for the TREC-X questions.

4 Results to the main task

We achieved a best-score of 0.191 to the main task by using Okapi for passage retrieval and a *NIL* threshold of XX %. We submitted 3 runs:

UdeMmainOk80: This run uses Okapi for passage retrieval (length = 1 paragraph) and a Δ_t of 80% for the insertion of a NIL answer.

UdeMmainOk60: This run uses Okapi for passage retrieval (length = 1 paragraph) and a Δ_t of 60% for the insertion of a NIL answer.

UdeMmainQt80: This run uses our own fixed-length passage and a Δ_t of 80% for the insertion of a NIL answer.

Table 7 indicates the official scores for these runs. The results confirm our first intuition: a *NIL* threshold of 80 % is better than a threshold of 60 % (compare runs UdeMmainOk80 and UdeMmainOk60). Our second intuition was also confirmed: Okapi is better at retrieving relevant passages than our own fixed-length passage retrieval algorithm is (compare runs UdeMmainOk80 and UdeMmainQt80).

5 System architecture for the list task

In the list task, the number of answers to provide is specified in the question. The QUANTUM architecture for the list task differs little from the main task. Question analysis patterns are adapted to extract the number of answers from the question (in

Run	MRR	
	Lenient	Strict
UdeMmainOk80	0.197	0.191
UdeMmainOk60	0.189	0.183
UdeMmainQt80	0.145	0.137

Table 7: Official results of our 3 runs for the main task. The differences between the 3 runs reside in the passage retrieval technique used and the threshold Δ_t for insertion of *NIL* answers.

case our patterns would fail to identify that number, we take the first number smaller than 50 to appear in the question). Passage retrieval is performed using Okapi only. The extraction of candidates is done by extraction functions described above. Candidate scoring and candidate expansion are different than for the main task. Of course, no *NIL* answers insertion needs to be done. We shall describe below the techniques we tried for the two runs we submitted: UdeMlistP and UdeMlistB.

Run UdeMlistP

We use the same candidate scoring algorithm than for the main task, that is:

$$s = s_{extraction} + s_{passage} + s_{proximity}$$

However, candidates are not expanded by taking extra characters to the left and to the right. Instead, they are expanded to the right only. This is due to the fact that an unsuspected correct candidate that would appear before a known candidate in the 50-character string might, at best, make no difference in the overall accuracy and, at worst, interfere with the redundant candidate elimination algorithm.

Candidates are expanded and added to the list of answers as long as we have not reached the desired number of answers and as long as candidates are not redundant. For the purpose of the list task, a candidate is considered redundant when an identical candidate has already been expanded (note the difference with the main task, where a candidate was considered redundant if it appeared *anywhere* in an already expanded answer).

Run UdeMlistB

For this run, the scoring of candidates has been modified. Let $\mathcal{C}$ be the list of all candidates found and $\mathcal{F}$ be the list of their frequencies f sorted in decreasing order (with duplicate frequencies eliminated). The

Run	Accuracy
UdeMlistP	0.15
UdeMlistB	0.07

Table 8: Official results of our 2 runs for the list track. The differences between the 2 runs reside in the scoring of candidates.

score s' of a candidate is given by

$$s' = s^2 * log\frac{s_{passage}}{rank(f)} * n$$

where s is $s_{extraction} + s_{passage} + s_{proximity}$, $rank(f)$ is the rank, in $\mathcal{F}$, of the candidate's frequency and n is the number of candidates contained in the same document as the candidate currently scored.

Candidates are sorted again according to their new score. They are expanded to the right and they are eliminated when redundant, in the same manner as for the run UdeMlistP described above.

6 Results to the list task

As Table 8 shows, QUANTUM achieved its best score with the run UdeMlistP, which used the same scoring algorithm than its best run for the main task (UdeMmainOk80). The other run UdeMlistB reached an accuracy of 0.07 with its particular scoring algorithm.

7 Conclusion

Last year, we participated to TREC-9 for the first time with an all hand-built QA system that relied heavily on regular expressions. This year, we tried to improve the system by incorporating specialized resources (Okapi, WordNet and Alembic) and by developping a new classification based on extraction functions. These rely on semantic relations between terms in the question and in the passages and on syntactic analysis of the passages.

We did not yet evaluate the effects of each of these modifications, especially our patterns for question analysis and our function-based classification. Last year, we obtained a strict MRR of 0.179 and 0.149 for our 50-character runs. Unfortunately, this year's scores do not seem significantly higher than last year's. However, last year, when we trained our XR^3 system with the TREC-8 corpus, we obtained 0.386 and 0.331 and noticed a significant drop on the TREC-9 corpus. A similar drop this year may not indicate a decrease in performance of our system, but an increase in difficulty of the task. This still remains to be investigated.

Acknowledgments

We wish to thank Sylvain Laganière for his help on installing Okapi and making it run with the TREC document collection; and Luc Bélanger for his insight on the list task and for implementing the extraction of answer cardinality. We also would like to thank the Mitre Corporation for making Alembic Workbench available for research purposes.

This project was financially supported by a scholarship from the Quebec *Fonds pour la formation de chercheurs et l'aide à la recherche* (FCAR), the Bell University Laboratories (BUL) and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[ABD+95] J. Aberdeen, J. Burger, D. Day, L. Hirschman, P. Robinson, and M. Vilain. MITRE: Description of the Alembic System as used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference*, San Francisco, 1995. Morgan Kaufman Publishers.

[LKL00] M. Laszlo, L. Kosseim, and G. Lapalme. Goal-driven Answer Extraction. In *Proceedings of TREC-9*, pages 563–572, Gaithersburg, Maryland, 2000.

[RW98] S.E. Robertson and S. Walker. Okapi/Keenbow at TREC-8. In *Proceedings of TREC-8*, pages 151–162, Gaithersburg, Maryland, 1998.