Improving QA Accuracy by Question Inversion

John Prager
IBM T.J. Watson Res. Ctr.
Yorktown Heights
N.Y. 10598
jprager@us.ibm.com

Pablo Duboue
IBM T.J. Watson Res. Ctr.
Yorktown Heights
N.Y. 10598
duboue@us.ibm.com

Jennifer Chu-Carroll
IBM T.J. Watson Res. Ctr.
Yorktown Heights
N.Y. 10598
jencc@us.ibm.com

Abstract

This paper demonstrates a conceptually simple but effective method of increasing the accuracy of QA systems on factoid-style questions. We define the notion of an inverted question, and show that by requiring that the answers to the original and inverted questions be mutually consistent, incorrect answers get demoted in confidence and correct ones promoted. Additionally, we show that lack of validation can be used to assert no-answer (**nil**) conditions. We demonstrate increases of performance on TREC and other question-sets, and discuss the kinds of future activities that can be particularly beneficial to approaches such as ours.

1 Introduction

Most QA systems nowadays consist of the following standard modules: QUESTION PROCESSING, to determine the bag of words for a query and the desired answer type (the type of the entity that will be offered as a candidate answer); SEARCH, which will use the query to extract a set of documents or passages from a corpus; and ANSWER SELECTION, which will analyze the returned documents or passages for instances of the answer type in the most favorable contexts. Each of these components implements a set of heuristics or hypotheses, as devised by their authors (cf. Clarke et al. 2001, Chu-Carroll et al. 2003).

When we perform failure analysis on questions incorrectly answered by our system, we find that there are broadly speaking two kinds of failure. There are errors (we might call them *bugs*) on the implementation of the said heuristics: errors in tagging, parsing, named-entity recognition; omissions in synonym lists; missing patterns, and just plain programming errors. This class can be characterized by being fixable by identifying incorrect code and fixing it, or adding more items, either explicitly or through training. The other class of errors (what we might call *unlucky*) are at the boundaries of the heuristics; situations were the system did not do anything "wrong," in the sense of *bug*, but circumstances conspired against finding the correct answer.

Usually when *unlucky* errors occur, the system generates a reasonable query and an appropriate answer type, and at least one passage containing the right answer is returned. However, there may be returned passages that have a larger number of query terms and an incorrect answer of the right type, or the query terms might just be physically closer to the incorrect answer than to the correct one. ANSWER SELECTION modules typically work either by trying to prove the answer is correct (Moldovan & Rus, 2001) or by giving them a weight produced by summing a collection of heuristic features (Radev et al., 2000); in the latter case candidates having a larger number of matching query terms, even if they do not exactly match the context in the question, might generate a larger score than a correct passage with fewer matching terms.

To be sure, *unlucky* errors are usually *bugs* when considered from the standpoint of a system with a more sophisticated heuristic, but any system at any point in time will have limits on what it tries to do; therefore the distinction is not absolute but is relative to a heuristic and system.

It has been argued (Prager, 2002) that the success of a QA system is proportional to the impedance match between the question and the knowledge sources available. We argue here similarly. Moreover, we believe that this is true not only in terms of the correct answer, but the distracters,[1] or incorrect answers too. In QA, an *unlucky* incorrect answer is not usually predictable in advance; it occurs because of a coincidence of terms and syntactic contexts that cause it to be preferred over the correct answer. It has no connection with the correct answer and is only returned because its enclosing passage so happens to exist in the same corpus as the correct answer context. This would lead us to believe that if a

[1] We borrow the term from multiple-choice test design.

Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL, pages 1073–1080,
Sydney, July 2006.

different corpus containing the correct answer were to be processed, while there would be no guarantee that the correct answer would be found, it would be unlikely (i.e. *very* unlucky) if the same incorrect answer as before were returned.

We have demonstrated elsewhere (Prager et al. 2004b) how using multiple corpora can improve QA performance, but in this paper we achieve similar goals without using additional corpora. We note that factoid questions are usually about relations between entities, e.g. "What is the capital of France?", where one of the arguments of the relationship is sought and the others given. We can *invert* the question by substituting the candidate answer back into the question, while making one of the given entities the so-called wh-word, thus "Of what country is Paris the capital?" We hypothesize that asking this question (and those formed from other candidate answers) will locate a largely different set of passages in the corpus than the first time around. As will be explained in Section 3, this can be used to decrease the confidence in the incorrect answers, and also increase it for the correct answer, so that the latter becomes the answer the system ultimately proposes.

This work is part of a continuing program of demonstrating how meta-heuristics, using what might be called "collateral" information, can be used to constrain or adjust the results of the primary QA system.

In the next Section we review related work. In Section 3 we describe our algorithm in detail, and in Section 4 present evaluation results. In Section 5 we discuss our conclusions and future work.

2 Related Work

Logic and inferencing have been a part of Question-Answering since its earliest days. The first such systems were natural-language interfaces to expert systems, e.g., SHRDLU (Winograd, 1972), or to databases, e.g., LIFER/LADDER (Hendrix et al. 1977). CHAT-80 (Warren & Pereira, 1982), for instance, was a DCG-based NL-query system about world geography, entirely in Prolog. In these systems, the NL question is transformed into a semantic form, which is then processed further. Their overall architecture and system operation is very different from today's systems, however, primarily in that there was no text corpus to process.

Inferencing is a core requirement of systems that participate in the current PASCAL Recognizing Textual Entailment (RTE) challenge (see http://www.pascal-network.org/Challenges/RTE and .../RTE2). It is also used in at least two of the more visible end-to-end QA systems of the present day. The LCC system (Moldovan & Rus, 2001) uses a *Logic Prover* to establish the connection between a candidate answer passage and the question. Text terms are converted to logical forms, and the question is treated as a goal which is "proven", with real-world knowledge being provided by Extended WordNet. The IBM system PIQUANT (Chu-Carroll et al., 2003) used Cyc (Lenat, 1995) in answer verification. Cyc can in some cases confirm or reject candidate answers based on its own store of instance information; in other cases, primarily of a numerical nature, Cyc can confirm whether candidates are within a reasonable range established for their subtype.

At a more abstract level, the use of inversions discussed in this paper can be viewed as simply an example of finding support (or lack of it) for candidate answers. Many current systems (see, e.g. (Clarke et al., 2001; Prager et al. 2004b)) employ redundancy as a significant feature of operation: if the same answer appears multiple times in an internal *top-n* list, whether from multiple sources or multiple algorithms/agents, it is given a confidence boost, which will affect whether and how it gets returned to the end-user.

The work here is a continuation of previous work described in (Prager et al. 2004a,b). In the former we demonstrated that for a certain kind of question, if the inverted question were given, we could improve the F-measure of accuracy on a question set by 75%. In this paper, by contrast, we do not manually provide the inverted question, and in the second evaluation presented here we do not restrict the question type.

3 Algorithm

3.1 System Architecture

A simplified block-diagram of our PIQUANT system is shown in Figure 1. The outer block on the left, QS1, is our basic QA system, in which the QUESTION PROCESSING (QP), SEARCH (S) and ANSWER SELECTION (AS) subcomponents are indicated. The outer block on the right, QS2, is another QA-System that is used to answer the inverted questions. In principle QS2 could be QS1 but parameterized differently, or even an entirely different system, but we use another instance of QS1, as-is. The block in the middle is our *Constraints Module* CM, which is the subject of this paper.

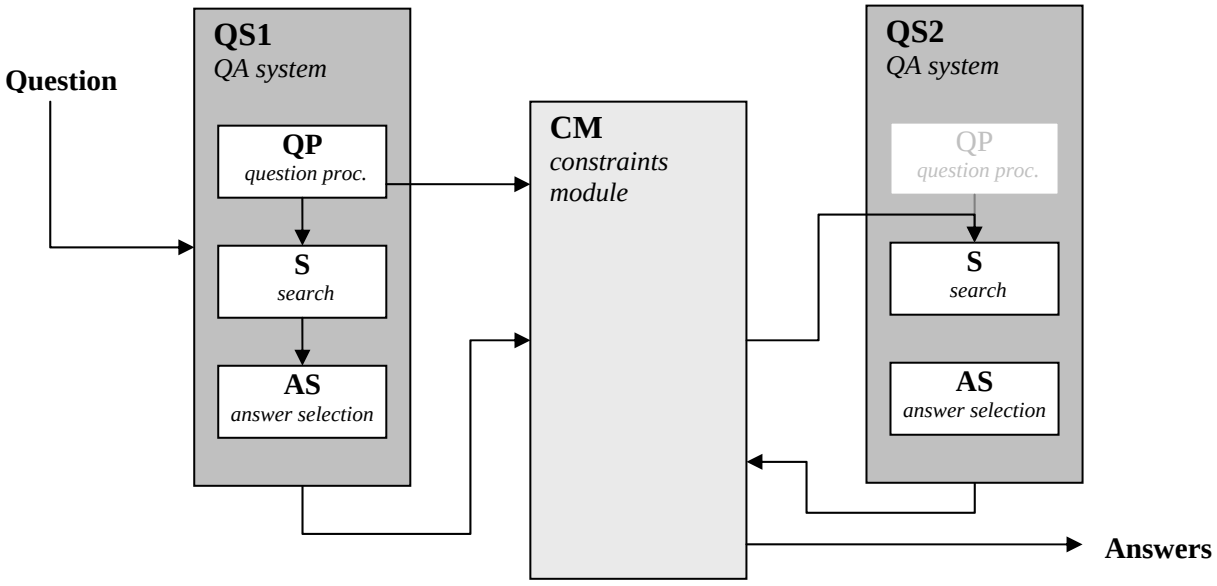

Figure 1. Constraints Architecture. QS1 and QS2 are (possibly identical) QA systems.

The Question Processing component of QS2 is not used in this context since CM simulates its output by modifying the output of QP in QS1, as described in Section 3.3.

3.2 Inverting Questions

Our open-domain QA system employs a named-entity recognizer that identifies about a hundred types. Any of these can be answer types, and there are corresponding sets of patterns in the QUESTION PROCESSING module to determine the answer type sought by any question. When we wish to invert a question, we must find an entity in the question whose type we recognize; this entity then becomes the sought answer for the inverted question. We call this entity the inverted or *pivot* term.

Thus for the question:

(1) "What was the capital of Germany in 1985?"

Germany is identified as a term with a known type (COUNTRY). Then, given the candidate answer <CANDANS>, the inverted question becomes

(2) "Of what country was < CANDANS> the capital in 1985?"

Some questions have more than one invertible term. Consider for example:

(3) "Who was the 33rd president of the U.S.?"

This question has 3 inversion points:

(4) "What number president of the U.S. was <CANDANS>?"

(5) "Of what country was <CANDANS> the 33rd president?"

(6) "<CANDANS> was the 33rd what of the U.S.?"

Having more than one possible inversion is in theory a benefit, since it gives more opportunity for enforcing consistency, but in our current implementation we just pick one for simplicity. We observe on training data that, in general, the smaller the number of unique instances of an answer type, the more likely it is that the inverted question will be correctly answered. We generated a set NELIST of the most frequently-occurring named-entity types in questions; this list is sorted in order of estimated cardinality.

It might seem that the question inversion process can be quite tricky and can generate possibly unnatural phrasings, which in turn can be difficult to reparse. However, the examples given above were simply English renditions of internal inverted structures – as we shall see the system does not need to use a natural language representation of the inverted questions.

Some questions are either not invertible, or, like "How did *X* die?" have an inverted form ("Who died of cancer?") with so many correct answers that we know our algorithm is unlikely to benefit us. However, as it is constituted it is unlikely to hurt us either, and since it is difficult to automatically identify such questions, we don't attempt to intercept them. As reported in (Prager et al. 2004a), an estimated 79% of the questions in TREC question sets can be inverted meaningfully. This places an upper limit on the gains to be achieved with our algorithm, but is high enough to be worth pursuing.

3.3 Inversion Algorithm

As shown in the previous section, not all questions have easily generated inverted forms (even by a human). However, we do not need to explicate the inverted form in natural language in order to process the inverted question.

In our system, a question is processed by the QUESTION PROCESSING module, which produces a structure called a *QFrame*, which is used by the subsequent SEARCH and ANSWER SELECTION modules. The QFrame contains the list of terms and phrases in the question, along with their properties, such as POS and NE-type (if it exists), and a list of syntactic relationship tuples. When we have a candidate answer in hand, we do not need to produce the inverted English question, but merely the QFrame that would have been generated from it. Figure 1 shows that the CONSTRAINTS MODULE takes the QFrame as one of its inputs, as shown by the link from QP in QS1 to CM. This *inverted QFrame* can be generated by a set of simple transformations, substituting the pivot term in the bag of words with a candidate answer <CANDANS>, the original answer type with the type of the pivot term, and in the relationships the pivot term with its type and the original answer type with <CANDANS>. When relationships are evaluated, a type token will match any instance of that type. Figure 2 shows a simplified view of the original QFrame for "What was the capital of Germany in 1945?", and Figure 3 shows the corresponding Inverted QFrame. COUNTRY is determined to be a better type to invert than YEAR, so "Germany" becomes the pivot. In Figure 3, the token <CANDANS> might take in turn "Berlin", "Moscow", "Prague" etc.

Keywords: {1945, Germany, capital}
AnswerType: CAPITAL
Relationships: {(Germany, capital), (capital, CAPITAL), (capital, 1945)}

Figure 2. Simplified QFrame

Keywords: {1945, <CANDANS>, capital}
AnswerType: COUNTRY
Relationships: {(COUNTRY, capital), (capital, <CANDANS>), (capital, 1945)}

Figure 3. Simplified Inverted QFrame.

The output of QS2 after processing the inverted QFrame is a list of answers to the inverted question, which by extension of the nomenclature we call "inverted answers." If no term in the question has an identifiable type, inversion is not possible.

3.4 Profiting From Inversions

Broadly speaking, our goal is to keep or re-rank the candidate answer hit-list on account of inversion results. Suppose that a question Q is inverted around pivot term T, and for each candidate answer C_i, a list of "inverted" answers $\{C_{ij}\}$ is generated as described in the previous section. If T is on one of the $\{C_{ij}\}$, then we say that C_i is *validated*. Validation is not a guarantee of keeping or improving C_i's position or score, but it helps. Most cases of failure to validate are called *refutation*; similarly, refutation of C_i is not a guarantee of lowering its score or position.

It is an open question how to adjust the results of the initial candidate answer list in light of the results of the inversion. If the scores associated with candidate answers (in both directions) were true probabilities, then a Bayesian approach would be easy to develop. However, they are not in our system. In addition, there are quite a few parameters that describe the inversion scenario.

Suppose Q generates a list of the top-N candidates $\{C_i\}$, with scores $\{S_i\}$. If this inversion method were not to be used, the top candidate on this list, C_1, would be the emitted answer. The question generated by inverting about T and substituting C_i is QT_i. The system is fixed to find the top 10 passages responsive to QT_i, and generates an ordered list C_{ij} of candidate answers found in this set.

Each inverted question QT_i is run through our system, generating inverted answers $\{C_{ij}\}$, with scores $\{S_{ij}\}$, and whether and where the pivot term T shows up on this list, represented by a list of positions $\{P_i\}$, where P_i is defined as:

$P_i = j$ if $C_{ij} = T$, for some j
$P_i = -1$ otherwise

We added to the candidate list the special answer **nil**, representing "no answer exists in the corpus."

As described earlier, we had observed from training data that failure to validate candidates of certain types (such as Person) would not necessarily be a real refutation, so we established a set of types SOFTREFUTATION which would contain the broadest of our types. At the other end of the spectrum, we observed that certain narrow candidate types such as UsState would definitely be refuted if validation didn't occur. These are put in set MUSTCONSTRAIN. Our goal was to develop an algorithm for recomputing all the original scores $\{S_i\}$ from some combination (based on either arithmetic or decision-trees) of

$\{S_i\}$ and $\{S_{ij}\}$ and membership of SOFTREFUTATION and MUSTCONSTRAIN. Reliably learning all those weights, along with set membership, was not possible given only several hundred questions of training data. We therefore focused on a reduced problem.

We observed that when run on TREC question sets, the frequency of the rank of our top answer fell off rapidly, except with a second mode when the tail was accumulated in a single bucket. Our numbers for TRECs 11 and 12 are shown in Table 1.

Top answer rank	TREC11	TREC12
1	170	108
2	35	32
3	23	14
4	7	7
5	14	9
elsewhere	251	244
% correct	34	26

Table 1. Baseline statistics for TREC11-12.

We decided to focus on those questions where we got the right answer in second place (for brevity, we'll call these second-place questions). Given that TREC scoring only rewards first-place answers, it seemed that with our incremental approach we would get most benefit there. Also, we were keen to limit the additional response time incurred by our approach. Since evaluating the top N answers to the original question with the Constraints process requires calling the QA system another N times per question, we were happy to limit N to 2. In addition, this greatly reduced the number of parameters we needed to learn.

For the evaluation, which consisted of determining if the resulting top answer was right or wrong, it meant ultimately deciding on one of three possible outcomes: the original top answer, the original second answer, or **nil**. We hoped to promote a significant number of second-place finishers to top place and introduce some **nils**, with minimal disturbance of those already in first place.

We used TREC11 data for training, and established a set of thresholds for a decision-tree approach to determining the answer, using Weka (Witten & Frank, 2005). We populated sets SOFTREFUTATION and MUSTCONSTRAIN by manual inspection.

The result is Algorithm A, where ($i \in \{1,2\}$) and

- The C_i are the original candidate answers
- The a_k are learned parameters ($k \in \{1..13\}$)
- V_i means the ith answer was validated
- P_i was the rank of the validating answer to question QT_i
- A_i was the score of the validating answer to QT_i.

Algorithm A. Answer re-ranking using constraints validation data.

1. **If** C_1 = **nil and** V_2, **return** C_2
2. **If** V_1 **and** $A_1 > a_1$, **return** C_1
3. **If not** V_1 **and not** V_2 **and** *type(T)* ∈ MUSTCONSTRAIN, **return nil**
4. **If not** V_1 **and not** V_2 **and** *type(T)* ∉ SOFTREFUTATION, **if** $S_1 > a_2$, **return** C_1 **else nil**
5. **If not** V_2, **return** C_1
6. **If not** V_1 **and** V_2 **and** $A_2 > a_3$ **and** $P_2 < a_4$ **and** $S_1\text{-}S_2 < a_5$ **and** $S_2 > a_6$, **return** C_2
7. **If** V_1 **and** V_2 **and** $(A_2 - P_2/a_7) > (A_1 - P_1/a_7)$ **and** $A_1 < a_8$ **and** $P_1 > a_9$ **and** $A_2 < a_{10}$ **and** $P_2 > a_{11}$ **and** $S_1\text{-}S_2 < a_{12}$ **and** $(S_2 - P_2/a_7) > a_{13}$, **return** C_2
8. **else return** C_1

4 Evaluation

Due to the complexity of the learned algorithm, we decided to evaluate in stages. We first performed an evaluation with a fixed question type, to verify that the purely arithmetic components of the algorithm were performing reasonably. We then evaluated on the entire TREC12 factoid question set.

4.1 Evaluation 1

We created a fixed question set of 50 questions of the form "What is the capital of *X*?", for each state in the U.S. The inverted question "What state is *Z* the capital of?" was correctly generated in each case. We evaluated against two corpora: the AQUAINT corpus, of a little over a million newswire documents, and the CNS corpus, with about 37,000 documents from the Center for Nonproliferation Studies in Monterey, CA. We expected there to be answers to most questions in the former corpus, so we hoped there our method would be useful in converting 2nd place answers to first place. The latter corpus is about WMDs, so we expected there to be holes in the state capital coverage[2], for which **nil** identification would be useful.[3]

[2] We manually determined that only 23 state capitals were attested to in the CNS corpus, compared with all in AQUAINT.

[3] We added Tbilisi to the answer key for "What is the capital of Georgia?", since there was nothing in the question to disambiguate Georgia.

The baseline is our regular search-based QA-System without the Constraint process. In this baseline system there was no special processing for **nil** questions, other than if the search (which always contained some required terms) returned no documents. Our results are shown in Table 2.

	AQUAINT baseline	AQUAINT w/constraints	CNS baseline	CNS w/constraints
Firsts (non-nil)	39/50	43/50	7/23	4/23
Total nils	0/0	0/0	0/27	16/27
Total firsts	39/50	43/50	7/50	20/50
% correct	78	86	14	40

Table 2. Evaluation on AQUAINT and CNS corpora.

On the AQUAINT corpus, four out of seven 2nd place finishers went to first place. On the CNS corpus 16 out of a possible 26 correct no-answer cases were discovered, at a cost of losing three previously correct answers. The percentage correct score increased by a relative 10.3% for AQUAINT and 186% for CNS. In both cases, the error rate was reduced by about a third.

4.2 Evaluation 2

For the second evaluation, we processed the 414 factoid questions from TREC12. Of special interest here are the questions initially in first and second places, and in addition any questions for which **nils** were found.

As seen in Table 1, there were 32 questions which originally evaluated in rank 2. Of these, four questions were not invertible because they had no terms that were annotated with any of our named-entity types, e.g. #2285 "How much does it cost for gastric bypass surgery?"

Of the remaining 28 questions, 12 were promoted to first place. In addition, two new **nils** were found. On the down side, four out of 108 previous first place answers were lost. There was of course movement in the ranks two and beyond whenever **nils** were introduced in first place, but these do not affect the current TREC-QA factoid correctness measure, which is whether the top answer is correct or not. These results are summarized in Table 3.

While the overall percentage improvement was small, note that only second–place answers were candidates for re-ranking, and 43% of these were promoted to first place and hence judged correct. Only 3.7% of originally correct questions were casualties. To the extent that these percentages are stable across other collections, as long as the size of the set of second-place answers is at least about 1/10 of the set of first-place answers, this form of the Constraint process can be applied effectively.

	Baseline	Constraints
Firsts (non-nil)	105	113
nils	3	5
Total firsts	108	118
% correct	26.1	28.5

Table 3. Evaluation on TREC12 Factoids.

5 Discussion

The experiments reported here pointed out many areas of our system which previous failure analysis of the basic QA system had not pinpointed as being too problematic, but for which improvement should help the Constraints process. In particular, this work brought to light a matter of major significance, *term equivalence*, which we had not previously focused on too much (and neither had the QA community as a whole). We will discuss that in Section 5.4.

Quantitatively, the results are very encouraging, but it must be said that the number of questions that we evaluated were rather small, as a result of the computational expense of the approach.

From Table 1, we conclude that the most mileage is to be achieved by our QA-System as a whole by addressing those questions which did not generate a correct answer in the first one or two positions. We have performed previous analyses of our system's failure modes, and have determined that the passages that are output from the SEARCH component contain the correct answer 70-75% of the time. The ANSWER SELECTION module takes these passages and proposes a candidate answer list. Since the CONSTRAINTS MODULE's operation can be viewed as a re-ranking of the output of ANSWER SELECTION, it could in principle boost the system's accuracy up to that 70-75% level. However, this would either require a massive training set to establish all the parameters and weights required for all the possible re-ranking decisions, or a new model of the answer-list distribution.

5.1 Probability-based Scores

Our ANSWER SELECTION component assigns scores to candidate answers on the basis of the number of terms and term-term syntactic relationships from the

original question found in the answer passage (where the candidate answer and wh-word(s) in the question are identified terms). The resulting numbers are in the range 0-1, but are not true probabilities (e.g. where answers with a score of 0.7 would be correct 70% of the time). While the generated scores work well to rank candidates for a given question, inter-question comparisons are not generally meaningful. This made the learning of a decision tree (Algorithm A) quite difficult, and we expect that when addressed, will give better performance to the Constraints process (and maybe a simpler algorithm). This in turn will make it more feasible to re-rank the top 10 (say) original answers, instead of the current 2.

5.2 Better confidences

Even if no changes to the ranking are produced by the Constraints process, then the mere act of validation (or not) of existing answers can be used to adjust confidence scores. In TREC2002 (Voorhees, 2003), there was an evaluation of responses according to systems' confidences in their own answers, using the Average Precision (AP) metric. This is an important consideration, since it is generally better for a system to say "I don't know" than to give a wrong answer. On the TREC12 questions set, our AP score increased 2.1% with Constraints, using the algorithm we presented in (Chu-Carroll et al. 2002).

5.3 More complete NER

Except in pure pattern-based approaches, e.g. (Brill, 2002), answer types in QA systems typically correspond to the types identifiable by their named-entity recognizer (NER). There is no agreed-upon number of classes for an NER system, even approximately. It turns out that for best coverage by our CONSTRAINTS MODULE, it is advantageous to have a relatively large number of types. It was mentioned in Section 4.2 that certain questions were not invertible because no terms in them were of a recognizable type. Even when questions did have typed terms, if the types were very high-level then creating a meaningful inverted question was problematic. For example, for QA without Constraints it is not necessary to know the type of "MTV" in "When was MTV started?", but if it is only known to be a Name then the inverted question "What <Name> was started in 1980?" could be too general to be effective.

5.4 Establishing Term Equivalence

The somewhat surprising condition that emerged from this effort was the need for a much more complete ability than had previously been recognized for the system to establish the equivalence of two terms. Redundancy has always played a large role in QA systems – the more occurrences of a candidate answer in retrieved passages the higher the answer's score is made to be. Consequently, at the very least, a string-matching operation is needed for checking equivalence, but other techniques are used to varying degrees.

It has long been known in IR that stemming or lemmatization is required for successful term matching, and in NLP applications such as QA, resources such as WordNet (Miller, 1995) are employed for checking synonym and hypernym relationships; Extended WordNet (Moldovan & Novischi, 2002) has been used to establish lexical chains between terms. However, the Constraints work reported here has highlighted the need for more extensive equivalence testing.

In direct QA, when an ANSWER SELECTION module generates two (or more) equivalent correct answers to a question (e.g. "*Ferdinand Marcos*" vs. "*President Marcos*"; "*French*" vs. "*France*"), and fails to combine them, it is observed that as long as either one is in first place then the question is correct and might not attract more attention from developers. It is only when neither is initially in first place, but combining the scores of correct candidates boosts one to first place that the failure to merge them is relevant. However, in the context of our system, we are comparing the pivot term from the original question to the answers to the inverted questions, and failure here will directly impact validation and hence the usefulness of the entire approach.

As a consequence, we have identified the need for a component whose sole purpose is to establish the equivalence, or generally the kind of relationship, between two terms. It is clear that the processing will be very type-dependent – for example, if two populations are being compared, then a numerical difference of 5% (say) might not be considered a difference at all; for "Where" questions, there are issues of granularity and physical proximity, and so on. More examples of this problem were given in (Prager et al. 2004a). Moriceau (2006) reports a system that addresses part of this problem by trying to rationalize different but "similar" answers to the user, but does not extend to a general-purpose equivalence identifier.

6 Summary

We have extended earlier Constraints-based work through the method of question inversion. The approach uses our QA system recursively, by taking candidate answers and attempts to validate them through asking the inverted questions. The outcome

is a re-ranking of the candidate answers, with the possible insertion of **nil** (no answer in corpus) as the top answer.

While we believe the approach is general, and can work on any question and arbitrary candidate lists, due to training limitations we focused on two restricted evaluations. In the first we used a fixed question type, and showed that the error rate was reduced by 36% and 30% on two very different corpora. In the second evaluation we focused on questions whose direct answers were correct in the second position. 43% of these questions were subsequently judged correct, at a cost of only 3.7% of originally correct questions. While in the future we would like to extend the Constraints process to the entire answer candidate list, we have shown that applying it only to the top two can be beneficial as long as the second-place answers are at least a tenth as numerous as first-place answers. We also showed that the application of Constraints can improve the system's confidence in its answers.

We have identified several areas where improvement to our system would make the Constraints process more effective, thus getting a double benefit. In particular we feel that much more attention should be paid to the problem of determining if two entities are the same (or "close enough").

7 Acknowledgments

This work was supported in part by the Disruptive Technology Office (DTO)'s Advanced Question Answering for Intelligence (AQUAINT) Program under contract number H98230-04-C-1577. We would like to thank the anonymous reviewers for their helpful comments.

References

Brill, E., Dumais, S. and Banko M. "An analysis of the AskMSR question-answering system." In *Proceedings of EMNLP 2002*.

Chu-Carroll, J., J. Prager, C. Welty, K. Czuba and D. Ferrucci. "A Multi-Strategy and Multi-Source Approach to Question Answering", Proceedings of the 11th TREC, 2003.

Clarke, C., Cormack, G., Kisman, D. and Lynam, T. "Question answering by passage selection (Multitext experiments for TREC-9)" in Proceedings of the 9th TREC, pp. 673-683, 2001.

Hendrix, G., Sacerdoti, E., Sagalowicz, D., Slocum J.: Developing a Natural Language Interface to Complex Data. VLDB 1977: 292

Lenat, D. 1995. "Cyc: A Large-Scale Investment in Knowledge Infrastructure." Communications of the ACM 38, no. 11.

Miller, G. "WordNet: A Lexical Database for English", Communications of the ACM 38(11) pp. 39-41, 1995.

Moldovan, D. and Novischi, A, "Lexical Chains for Question Answering", COLING 2002.

Moldovan, D. and Rus, V., "Logic Form Transformation of WordNet and its Applicability to Question Answering", Proceedings of the ACL, 2001.

Moriceau, V. "Numerical Data Integration for Cooperative Question-Answering", in EACL Workshop on Knowledge and Reasoning for Language Processing (KRAQ'06), Trento, Italy, 2006.

Prager, J.M., Chu-Carroll, J. and Czuba, K. "Question Answering using Constraint Satisfaction: QA-by-Dossier-with-Constraints", Proc. 42nd ACL, pp. 575-582, Barcelona, Spain, 2004(a).

Prager, J.M., Chu-Carroll, J. and Czuba, K. "A Multi-Strategy, Multi-Question Approach to Question Answering" in New Directions in Question-Answering, Maybury, M. (Ed.), AAAI Press, 2004(b).

Prager, J., "A Curriculum-Based Approach to a QA Roadmap"' LREC 2002 Workshop on Question Answering: Strategy and Resources, Las Palmas, May 2002.

Radev, D., Prager, J. and Samn, V. "Ranking Suspected Answers to Natural Language Questions using Predictive Annotation", Proceedings of ANLP 2000, pp. 150-157, Seattle, WA.

Voorhees, E. "Overview of the TREC 2002 Question Answering Track", *Proceedings of the 11th TREC*, Gaithersburg, MD, 2003.

Warren, D., and F. Pereira "An efficient easily adaptable system for interpreting natural language queries," *Computational Linguistics*, 8:3-4, 110-122, 1982.

Winograd, T. Procedures as a representation for data in a computer program for under-standing natural language. *Cognitive Psychology*, 3(1), 1972.

Witten, I.H. & Frank, E. *Data Mining. Practical Machine Learning Tools and Techniques*. Elsevier Press, 2005.