A Comparison of Hard Filters and Soft Evidence for Answer Typing in Watson

Chris Welty, J. William Murdock, Aditya Kalyanpur, James Fan

IBM Research
cawelty@gmail.com, {murdockj,adityakal,fanj}@us.ibm.com

Abstract. Questions often explicitly request a particular type of answer. One popular approach to answering natural language questions involves filtering candidate answers based on precompiled lists of instances of common answer types (e.g., countries, animals, foods, etc.). Such a strategy is poorly suited to an open domain in which there is an extremely broad range of types of answers, and the most frequently occurring types cover only a small fraction of all answers. In this paper we present an alternative approach called TyCor, that employs soft filtering of candidates using multiple strategies and sources. We find that TyCor significantly outperforms a single-source, single-strategy hard filtering approach, demonstrating both that multi-source multi-strategy outperforms a single source, single strategy, and that its fault tolerance yields significantly better performance than a hard filter.

1 Introduction

Natural Language Question Answering (QA) systems allow users to ask questions in their own natural language, using their own terminology, and receive a concise answer [1]. While the success of Waston [2] was an important landmark, research in this area continues in the semantic web, as there is a clear gap between the complexity of logic-based semantic web representations and the capabilities of web users on a large scale [3]. A comprehensive survey can be found at [4].

Since the late 1990s, approaches to QA have always included a strong focus on typing. A common technique, possibly influenced by the way people answer questions, is to analyze the question for the type of thing being asked for, and then to restrict answers to that type. Many experimental systems have used such a *type-and-generate* approach, and rely on a process of *Predictive Annotation* [5], in which a fixed set of expected answer types are identified through manual analysis of a domain, and a background corpus is automatically annotated with possible mentions of these types before answering questions. These systems then map answer types into the fixed set used to annotate the corpus, and restrict candidate answers retrieved from the corpus to those that match this answer type using semantic search (IR search augmented with the ability to search for words tagged with some type). More recent semantic web QA systems similarly rely on the ability to map questions into ontology terms, and assume the ontology types cover all the questions and answers.

Most existing open-domain QA systems also use a single source of typing information, or in some cases two sources (where one is WordNet [6]), and a single strategy for analyzing questions for their answer type. While it is well known that multiple sources would provide more coverage, most systems avoid exploiting multiple sources because they believe it is necessary to perform ontology alignment between them.

In this paper we compare the single source, single strategy, type-and-generate approach to a multi-source, multi-strategy *generate-and-type* approach called TyCor, in which candidate answers are initially produced without use of answer type information, and subsequent stages score the degree to which the candidate answer's type can be coerced into the Lexical Answer Type (LAT) of the question. To isolate the comparison, we reduce the type-and-generate approach to hard filtering of answer candidates by type, and demonstrate that type coercion (TyCor) outperforms it on a large set of questions from the Jeopardy! quiz show, without requiring ontology alignment.

2 Background

An important part of Watson, the QA system that defeated the best human players on the American television quiz show *Jeopardy!*, is the way it identifies the type of the answer from the question and evaluates candidate answers with respect to that type. At the very beginning of the Watson project, we used a type-and-generate approach for generating candidate answers, simply because that is what we had. However, in our early analysis of the domain of questions from the TV quiz show Jeopardy!, we found three main problems that led us to consider an alternative.

To begin with, the design of the *Jeopardy!* game makes confidence an important part of performance, experts only answer when they think they know the answer and have to compete with the other players to get to be the one that answers. In order to account for the impact of confidence on performance, we adopted a metric of precision at 70% answered (P@70) with an expert level performance target of 85% P@70. In other words, our target was to perform at 85% precision on the 70% of answers in which the system was most confident.

The first problem we had with the type-and-generate approach resulted from analysis of 20K questions from our domain: we observed a very long tail of types (see 2). While the type system for our predictive annotation engine was among the largest in the community (over 100 types), these did not cover even half the questions. There were roughly 5K different type words used in the 20K questions, more than half of these occurred fewer than three times in the question set, and roughly 12% occurred once. As we continued to evaluate on hidden data, we found the 12% number to be constant: new types were being introduced at this rate (one in eight questions on average). In addition, 15% of questions did not identify the answer type with a specific word. This data seems immediately to contraindicate the use of a predetermined set of answer types.

The second problem with type-and-generate is the reliance on question analysis to map type *words* in the question to the predetermined set of recognized answer types. Human language is remarkably rich when it comes to describing types; nearly any word can be used as a type, especially in questions, e.g.:

- Invented in the 1500s to speed up the game, this *maneuver* involves 2 pieces of the same color. (Castling)

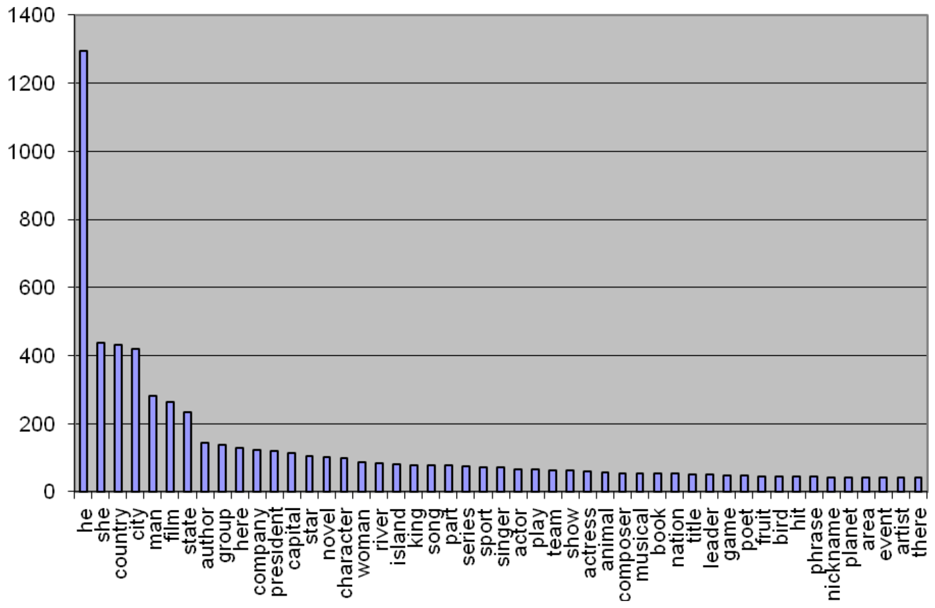

Fig. 1. The distribution of the 30 most frequent LATs in 20K Jeopardy! questions.

- The first known air mail service took place in Paris in 1870 by this *conveyance*. (Hot-air balloon)
- In 2003 this Oriole *first sacker* was elected to the Baseball Hall of Fame. (Eddie Murray)
- Freddy Krueger was introduced in this 1984 *scarefest*. (A Nightmare on Elm Street)
- When hit by electrons, a phosphor gives off electromagnetic energy in this *form*. (light)
- Whitney's patent for *this* revolutionized the garment industry. (the cotton gin)

Given this variability, one of the intuitive problems with relying on predictive annotation is that we cannot reliably map from these words, which we call the *lexical answer type* (LAT), to the predetermined set of known answer types. Even in the cases where there should be a mapping, as in the case of scarefest, or conveyance, there is no way to accurately predict for unseen questions what LATs might be used for the known types.

The third and final problem was that the type-and-generate approach itself was a single point of failure that quite simply prevented us from reaching human expert performance. The basic idea (analyze the question and determine the answer type from a predetermined set, find answers in the background corpus that have been labelled with that type, *then* process those candidates with other forms of evidence) limits the ultimate performance of the end to end system to the recall of the question analysis times the recall of the predictive annotation. In our case, the recall of our predictive annotator was estimated at about 70% *for the types it knew*, and the recall of the question analysis was similar, leaving us with a maximum performance of under 50% absolute for less that half the questions, and then we had to have an approach for the other half.

While we may have had some hope of improving the recall of question analysis and predictive annotation, these problems and the need to implement some way to handle questions in the long tail of LATs anyway, forced us to rethink the approach. It seemed to us that we needed to be open and flexible about types, treating them as a property of a question and answer combined. In other words, instead of finding candidates of the right type, we should find candidates (in some way) and judge whether each one is of the right type by examining it in context with the lexical answer type from the question. Furthermore, we needed to accommodate as many sources as possible that reflect the same descriptive diversity as these questions. Our hypothesis was that, to reach expert human performance, we needed a system design in which any part of the system could fail to produce the proper result, without preventing the overall system from succeeding.

3 TyCor

We use the term *Type Coercion* to refer to a process of determining whether it is possible to interpret a candidate answer that is consistent with the type requested by the question. Note that this term is also used by Pustejovsky [7] as a linguistic phenomenon in which a speaker imposes an abstract type onto a word by context. For example, saying that someone finished a book coerces an interpretation of ”book” as an event, while the word ”book” might be interpreted as a physical object in other contexts. Our use of the same term is largely coincidental; we describe a process of interpretation not a generative theory of language. However, we share with Pustejovsky the perspective that types can be dynamic and influenced by context.

3.1 Question Analysis

TyCor depends on getting the type word from the question, and gathers evidence for each candidate answer that it is of that type. The type word is not interpreted according to some predefined type system or ontology, unlike with type-and-generate approaches to question answering. We call the uninterpreted type word from the question the *lexical answer type* (LAT). If the LAT has been mapped into some predefined type system, we refer to the resulting type as the *semantic answer type* (SAT).

LAT recognition is easier than mapping to a semantic type, and though imperfect our LAT detection measures above .9 F1 on 20K randomly selected questions [8]. Like all parts of our system, LAT detection includes a confidence, and all type scores are combined with LAT confidence.

3.2 Candidate Generation

Our approach to candidate generation, driven by the observations discussed above, was to raise recall well above .8, and expend more effort on answer scoring to promote correct answers. A full discussion of candidate generation is beyond the scope of this paper, and can be found elsewhere [9], but improving this step to .85 recall for the top 500 answers was significant.

3.3 Answer Scoring

One of the significant differences between our approach and that of many previous QA systems is the manner in which candidate answers are generated, and that the processing of type information has been moved from candidate generation and search to answer scoring. Answer scoring is the step in which all candidates, regardless of how they were generated, are evaluated. During the answer scoring phase, many different algorithms and sources are used to collect and score evidence for each candidate answer with respect to the question. Type information is just one kind of evidence that is used for scoring, there are over 100 other dimensions of evidence including temporal and spatial constraints, n-grams, popularity, source reliability, skip-bigrams, substitutability, etc. Each answer scoring method is independent from all others, and may fail on any particular question-answer pair. The general idea is that across all methods, more will succeed for the correct answer than any other. In the end, this idea turned out to be true often enough to win.

3.4 Final Answer Merging and Ranking

With over 100 different methods for retrieving and scoring evidence for candidate answers, an overall determination of the final answer must combine the scores from each scoring algorithm for each answer in a way that weights each score as appropriate for the context given by the question [10]. In general, using a large number of blind training examples (roughly 3K questions with answers), we learn a set of context-dependent vector models in which each score corresponds to a dimension that is assigned a weight and combined using a logistic function. The contexts are mainly based on properties of the question: is there a LAT at all, is the question decomposed, etc.

An important quality of answer scores, due to the way the scores are combined, is that they exhibit monotonic behavior. That is, they should consistently increase (or decrease) with the probability of the answer being correct. For TyCor components, this requirement meant special attention had to be paid in the framework for estimating and modeling error. To accomplish this, we broke all TyCor components into four basic steps, and collected error rates for these steps. This allowed us to make more fine grained predictions of confidence in typing.

3.5 TyCor Framework

TyCor is a class of answer scoring components that take a LAT and a candidate answer, and return a probability that the candidate's type is the LAT. Note that since language does not distinguish between instantiation and subclass, the TyCor components must allow for this, and given a candidate answer that refers to a class, TyCor should give it a high score if it can be interpreted as a subclass or instance of the LAT. As mentioned above, LAT detection produces a confidence, so each (answer,LAT) score is modified by the LAT confidence.

Each TyCor component uses a source of typing information in some cases (see e.g. Lexical TyCor in the next section) this source is the answer itself and performs four steps, each of which is capable of error that impacts its confidence. A more complete description of the framework can be found in [11] and [12], but briefly:

Entity Disambiguation and Matching (EDM): The most obvious, and most error-prone, step in using an existing source of typing information is to find the entity in that source that corresponds to the candidate answer. Since the candidate is just a string, this step must account for both polysemy (the same name may refer to many entities) and synonymy (the same entity may have multiple names). Each source may require its own special EDM implementations that exploit properties of the source, for example DBpedia encodes useful naming information in the entity id. EDM implementations typically try to use some context for the answer, but in purely structured sources this context may be difficult to exploit.

Predicate Disambiguation and Matching (PDM): Similar to EDM, the type in the source that corresponds to the LAT found. In some sources this is the same algorithm as EDM, in others, type looking requires special treatment. In a few, especially those using unstructured information as a source, the PDM step just returns the LAT itself. In type-and-generate, this step corresponds to producing a semantic answer type (SAT) from the question. PDM corresponds strongly to notions of word sense disambiguation with respect to a specific source.

Type Retrieval (TR): Once an entity is retrieved from the source, if applicable the types of that entity must be retrieved. For some TyCors, like those using structured sources, this step exercises the primary function of the source and is simple. In others, like unstructured sources, this may require parsing or other semantic processing of some small snippet of natural language.

Type Alignment (TA): The results of the PDM and TR steps must then be compared to determine the degree of match. In sources containing e.g. a type taxonomy, this may include checking the taxonomy for subsumption, disjointness, etc. For other sources, alignment may utilize resources like wordnet for finding synonyms, hypernyms, etc. between the types.

3.6 Multi Source Multi Strategy TyCor

We have implemented more than ten different TyCor components for scoring candidate answers. Some of our TyCor components share algorithms but use different sources of evidence, others use different algorithms on the same sources. The algorithms mainly involve different approaches to the four TyCor steps as appropriate for the source, with an eye towards accurately accounting for the error in the steps (most notably EDM and alignment) to produce a meaningful score. The sources we use range from DB-pedia, WordNet, Wikipedia categories, Lists found on the web (e.g. list of nobel prize winners), as well as the first sentence of Wikipedia articles, lists of common male and female names, specially mined databases of people and their genders, and mined results of is-a patterns [13] from large corpora.

Generally speaking, TyCor scores range from [-1,1]; negative scores are interpreted as evidence that a candidate is not of the right type, positive scores that it is, and a 0 score is interpreted as unknown. For example, if a candidate is simply not known by a source, this doesn't constitute evidence that the answer is not of the right type. Most

TyCor methods do not even include the ability to collect negative evidence, those that do are indicated below.

A full discussion of all the components is beyond the space limitations of this paper. A brief overview follows.

Yago: Many candidate answers in our domain are titles of Wikipedia articles. Each of these is an entity in DBpedia, a linked open data source compiled automatically from Wikipedia infoboxes and article templates. Entities (articles) in DBpedia have types represented in RDF from Yago [14], a semi-automatically constructed type taxonomy based on WordNet, corpus analysis, and Wikipedia. In addition, we have added roughly 200 disjointness constraints (e.g. a Person is not a Country) at high levels in the taxonomy. Using a special purpose reasoner to check for subsumption and disjointness, Yago Tycor can produce negative evidence when a candidate matches only types that are disjoint from all the types matching the LAT.

Intro: The first sentence of all Wikipedia articles identifies some types for the entity described in the article, e.g. Tom Hanks is an American actor, producer, writer, and director. Intro TyCor utilizes a special source mined from these intro passages and scores LAT matches, using WordNet synonyms and hypernyms for Type alignment.

Gender: Uses a custom source of data mined from articles about people by determining which pronouns are most commonly used to refer to the person, scores the degree to which a candidate answer is of the appropriate gender, or not. Can produce negative evidence if the LAT indicates one gender and the answer is found to be of the other.

ClosedLAT: Certain LATs identify types with enumerable lists of instances, such as Countries, US States, US Presidents, etc. When such a list is available, this TyCor component is capable of producing a negative type score for candidate answers that are not in the list. Of course, as with everything described here, confidence is never perfect due to name matching issues and the possibility that the LAT is used in a non-standard way. For example, the mythical country of Gondor is not on our closed list, but could conceivably be the answer to a country-LAT question. Based on the domain analysis in Figure 1, we selected the 50 most frequent closed LATs and developed lists of their instances.

Lexical: Occasionally LATs specify some lexical constraint on the answer, like that it is a verb, or a phrase, or a first name, etc. Lexical TyCor uses various special-purpose algorithms based on the LAT for scoring these constraints. Passage: When a candidate answer is extracted from a passage of text, occasionally that passage includes some reference to the candidate's type, e.g. Actor Tom Hanks appeared at the premier of his new film. Passage Tycor uses WordNet for type alignment to match the LAT to the type word in the passage.

Identity: Some candidate answers contain the LAT as part of their name it can be quite embarrassing for a system to miss King Ludwig as a king just because he isn't on a list of Kings. Identity TyCor uses WordNet to match the LAT to any part of the candidate string itself.

NED: The NED TyCor uses the engine previously used for predictive annotation. Although our approach supercedes pure predictive annotation for candidate generation, it does not throw it away. The NED engine recognizes over 100 SATs, most of which are among the top 100 LATs. It uses a special purpose rule base PDM to map from LATs to SATs, and recognizes candidates as being of the right SAT mainly through large manually curated list and patterns.

WordNet: WordNet is used primarily in other TyCor components to assist in the type alignment phase, however it does contain some limited information about well known entities such as famous scientists, a few geographic and geopolitical entities, etc. It also has high coverage of biological taxonomies. This TyCor implementation has very high precision but low recall.

WikiCat: Wikipedia articles are frequently associated with categories that more or less match, in style and content, the linguist ability to say that one thing has some topic association. All these categories are stored in DBpedia. The WikiCat TyCor uses primarily the headword of all the category names for an entity, and performs type alignment using WordNet synonyms and hypernyms. Wikicat does not use the category structure (e.g. subcategory) at all, as this adds too much noise.

List: Wikipedia and many other web sources contain lists of things associated in some way, like List of Argentinean Nobel Prize Winners. We collect these lists, and like WikiCat map only the headwords to the LATs.

Prismatic: Prismatic [15] is a repository of corpus statistics such as all subject-verb-object tuples, or all subject-verb-preposition-object tuples, that allows counting queries (e.g. how many SVPO tuples with stars as the verb). Prismatic Tycor measures the frequency with which the candidate answer is directly asserted to be an instance of the lexical answer type using is-a patterns [13].

4 Experiment

We performed a series of experiments to validate our hypothesis that the type-and-generate (TaG) approach exemplified by TyCor would lead to improved performance over a generate-and-type (GaT) approach to open-domain QA. In particular, we were interested in validating that generating only candidate answers believed to be of the right semantic type before seeing the question would limit system performance.

4.1 Experimental Setup

The full QA system we ran this experiment with is the version of Watson that participated in the televised exhibition match. It performs at 71% accuracy overall and with a confidence estimation capability that allows it to perform at over 85% P@70 (precision at 70% answered). This combination of overall accuracy and confidence estimation is unprecedented in the community. The system was tested in a public display on blind questions which were held by independent auditors to ensure the test was fair and truly blind.

We ran a second set of experiments using a smaller “lite” version of Watson that removes all the answer scoring components (except, where applicable, the TyCor component being tested), relying on only the scores resulting from candidate generation methods, primarily (though not exclusively) search. This set of experiments was intended to demonstrate that the relative difference between TaG and GaT changes with the overall performance.

While the performance numbers are interesting in themselves (the lite version of the system already performs at levels that would make it one of the top 2 open domain QA systems at TREC [16]) it is simply beyond the scope of this paper to describe how they were obtained. For the purposes of this paper, within each set of experiments (full system and lite system) all aspects of the system were held constant except for the approach to answer typing. Thus it is the relative performance numbers we focus on.

For the full and lite versions of the system, we ran four experiments:

No Typing: Uses no typing components in scoring answers, although since the LAT is a search keyword, some type information does make it into candidate generation and scoring.

Tycor: Uses the full set of TyCor components (described above) as separate answer scorers.

NED Tycor: Uses only the NED based TyCor component as an answer scorer, to illustrate the effectiveness of predictive annotation as a source of evidence.

TaG: We simulate the type-and-generate approach by using the NED TyCor component as a hard filter on candidate answers that are scored. Candidates are generated in the same way as other versions of the system, but all those that do not match the SAT according to the NED annotator are filtered out. We chose this approach over using actual semantic search for candidate generation in order to hold the search engine itself constant, since the one we used in Watson did not support semantic search [9]. As can be seen from the lite version of the system, the search component alone produces 50% accuracy and 64% P@70, which is significant.

All experiments were run on 3500 blind questions from past Jeopardy! games. Candidate generation recall for all versions is the same, at 87% in the top 500.

4.2 Results

	Lite System		Full System	
	Accuracy	P@70	Accuracy	P@70
No Typing	50.0%	63.4%	65.5%	81.4%
NED TyCor	53.8%	67.8%	67.9%	84.2%
TaG	45.9%	62.1%	55.3%	74.9%
TyCor	58.5%	75.0%	70.4%	87.4%

Table 1. Relative Performance on Lite and Full QA Systems

Table 1 shows the relative performance of the four configurations of the typing components described above on the lite and full versions of the system. The lite version has no other answer scoring except search and candidate generation. The full version uses all answer scoring in the system that performed in the exhibition match. Accuracy (precision @ 100% answered), is shown along with P@70. All results within columns are different with statistical significance at $p < .01$.

The main comparisons are between the TaG and TyCor versions of the system, and between the NED TyCor and TaG, in both the lite and full systems, although it is clearly worth noting that TaG draws down the performance of the system even with no typing at all.

4.3 Analysis and Discussion

Throughout this paper we have been careful to distinguish the type-and-generate approach (TaG) from predictive annotation itself. Predictive annotation refers only to the process of annotating a background corpus with types from a predefined set of possible semantic answer types. The TaG approach is one in which questions are analyzed to produce the semantic answer type, and candidate generation produces only candidate answers which were identified as being of the right type during the predictive annotation step. Note that while there is a clear similarity between predictive annotation and named entity detection (NED), predictive annotation tends to favor recall over precision, and is generous in labeling mentions in a corpus; often mentions will have multiple type labels.

In the full system, the TaG configuration performs at 55.3% accuracy and 74.9% P@70, compared to 70.4% and 87.4% resp. for TyCor. It could be argued that the multi-source multi-strategy (hereafter, multi-strategy) aspect of TyCor accounts for this difference and not the TaG vs. generate-and-type (GaT) approaches, and that a multi-strategy approach could have been used during predictive annotation itself. If true, this would mean the only difference between the two is one of efficiency at question answering time. In reality, aspects of the multi-strategy approach make it computationally impractical to annotate the entire corpus this way, but to further isolate the comparison we ran the system with only the NED TyCor component, which is the same component used to hard filter answers in the TaG configuration.

The results demonstrate that the TaG approach is a hindrance to a high performing QA system. The idea of predictive annotation itself, however, is a helpful one, as shown by the NED TyCor rows. In both cases, having some evidence for typing, even when (as in this case) it covers roughly 60% of the questions, is more helpful than no typing information at all, provided that evidence is used in answer scoring, not as a filter. Indeed, the NED TyCor component is consistently among the top 4 performing TyCor components over large sets of questions. The main issue it has, which is more than made up for by the full complement of TyCors, is that it works only on types in the head of the LAT curve (see 2).

The main reason the GaT approach outperforms TaG is the fault tolerance of the answer scoring phase in DeepQA. It is possible for a correct answer to “win out” and become the top answer even when the NED TyCor believes the answer is of the wrong type. In TaG, when question analysis or predictive annotation make mistakes, there is no way to recover from them. This is borne out further by the difference between the full system and the lite, where there are more kinds of answer scoring and thus more ways in which other evidence can overcome typing failures; the relative and absolute differences in the full system for TaG vs. GaT are more. Interestingly, until our system reached a performance level of about 50% overall accuracy, we were not able to validate the TaG vs. GaT hypothesis experimentally, due mainly to the lack of sufficient other evidence to overcome the typing failures. This seems to indicate that, for systems performing at

50% or less, TaG is a reasonable approach. This performance level characterized all but the top performing system at the TREC QA evaluations [16].

The results also show the relative impact of the multi-strategy TyCor approach on the system. In the lite system, full TyCor adds 15.7% relative (8.5% absolute) to accuracy and 16.8% relative to P@70 (11.6% absolute). For the full system, TyCor adds 7.2% relative accuracy improvement (4.9% absolute), and 7.1% relative P@70 (6% absolute). All components in our system (there are over 100) see the same sort of diminished relative impact in the full system compared to the lite, this is due to the fact that many of the scoring components overlap. For example, another scoring component counts the n-gram frequency of terms in the question with the candidate answer. Since the LAT is one of the terms in the question, the ngram score can provide partial information about typing.

That the multi-strategy approach outperforms a single strategy for typing candidate answers should come as no surprise, but it is interesting to note that the DeepQA architecture facilitates the combination easily. Training the system only against a question-answer ground truth (ie as opposed to a task-specific ground truth of candidate answers and types), DeepQA is able to effectively combine the 14 different TyCor implementations to produce significantly better results than against any of them in isolation. Here we show only the comparison to the NED TyCor in isolation, a full set of experiments showing the relative performance of each in isolation can be found in [12].

5 Related Work

QUARTZ [17] is a QA System that uses WordNet as the background knowledge base for mapping answer types expressed in the question. This approach mitigates the type coverage issue in earlier QA systems due to the conceptual breadth of WordNet. The mapping from answer type to WordNet synset, which is essentially a Word Sense Disambiguation (WSD) problem, is done using statistical machine learning techniques. Having obtained a WordNet synset T for the answer type, the system estimates a set of complementary-types C(T) in WordNet (typically considering siblings of T). A given candidate answer is then determined to be of the correct type if it has a stronger correlation to type T than to the types in C(T), where the correlation is computed using Web data and techniques like mutual information (MI) e.g. how often does the candidate answer co-occur with the type across a collection of Web documents. In [18] the approach has been taken a step further by combining correlation-based typing scores with type information from resources such as Wikipedia, using a machine-learning based scheme to compute type validity.

Both [18] and [17] use a similar approach to type-coercion in DeepQA in that they defer type-checking decisions to later in the QA pipeline and use a collection of techniques and resources (instead of relying on classical NERs) to check for a type match between the candidate and the expected answer type in the question. However, that is where the similarity ends. A fundamental difference in our approach is that the type match information is not used as a filter to throw out candidate answers, instead, the individual TyCor scores are combined with other answer scores using a weighted vector model. Also, our type-coercion is done within a much more elaborate framework that separates out the various steps of EDM, PDM, Type Alignment etc, and the intermediate algorithms (and resources) used in these steps are far more complex and varied – having either much more precision, and/or much broader scope compared to existing work, and a precise model of error. For example, the only use of Wikipedia content for type inference in [18] is through a shallow heuristic that searches for the mention of the expected answer type on the Wikipedia page of a candidate answer (mapped using an exact string match to the page title) and makes a Yes/No decision for the type validity based on this finding. In contrast, in our Wikipedia-based TyCors we use an EDM algorithm to map the candidate answer string to a Wikipedia page using a variety of resources such as Wikipedia redirects, extracted synonym lists, link-anchor data etc, and then use different kinds of type information expressed in Wikipedia, such as lexical types in the introductory paragraphs, Wikipedia categories etc. Similarly, while [17] uses the notion of complement-type sets, which are approximated using heuristics such as sibling-types, we define explicit disjoint types in the Yago Ontology and use disjointness information to down weigh candidate answers whose types are disjoint with the LAT.

An approach that combines slightly softens the type and generate approach by applying semantic answer type constraints to passage ranking is presented in [19]. Like our system, search terms are extracted from questions for a search engine which returns ranked sets of passages. These passages are then pruned by removing all passages that do not contain terms labeled with the semantic answer type detected in the question. The approach shows improvement in passage ranking metrics, but QA performance is not evaluated. Such an approach could be used in our system, as removing passages without detected answer types in them is subtlety different than removing answers themselves; candidate answers can be generated from passages by e.g. extracting all noun phrases, whether they have the right annotation labels or not. Still, our analysis here suggests that this would probably only help for lower performing QA systems.

A similar approach to our combination of NED and WikiCat is presented in [20]. The traditional type-and-generate approach is used when question analysis can recognize a semantic answer type in the question, and falls back to Wikipedia categories for candidate generation, using it as a hard filter instead of predictive annotation. In our approach we assume any component can fail, and we allow other evidence, both from other TyCor components and from other answer scoring components, to override the failure of one particular component when there is sufficient evidence.

6 Conclusion

Answer typing is an important component in a question answering (QA) system. The majority of existing factoid QA systems adopt a type-and-generate pipeline that rely on a search component to retrieve relevant short passages from the collection of newswire articles, and to extract and rank candidate answers from those passages that match the answer type(s) identified, based on the question, from a pre-constructed and fixed set of semantic types of interest. For example, the semantic answer type for the question "What city is was 2008 World Sudoku Championship held in?" is *City*, and the candidate answer set for this question typically consists of all cities extracted from a relevant passage set by a named entity recognizer (NER). This approach suffers from two main problems.

First, restricting the answer types to a fixed and typically small set of concepts makes the QA system brittle and narrow in its applicability and scope. Such a closed-typing approach does not work for open-domain Question Answering, and in particular the Jeopardy! problem, where answer types in questions span a broad range of topics, are expressed using a variety of lexical expressions (e.g. scarefest when referring to the semantic type horror movie) and are sometimes vague (e.g. form) or meaningless (e.g. it).

Second, the QA system performance is highly dependent on the precision and recall of the NERs used, as they act as candidate selection filters, and the system has no way to recover from errors made at this stage.

We have presented an approach to handling answer types in open domain question answering systems that is more open and flexible than the commonly used type-and-generate approach. Our generate-and-type approach does not rely on a fixed type system, uses multiple strategies and multiple sources of typing information, gathers and evaluates evidence based on the type words used in the question, and is not a hard filter. Our approach is broken into four basic steps, which have allowed us to more accurately model and predict the error of typing statements, which increases the ability of the typing system to inform the confidence in final answers. We compared our approach to type-and-generate within a high performance QA system and found a significant difference in performance, both in the overall accuracy and the ability to estimate confidence.

7 Acknowledgements

Numerous people contributed to Watson

References

1. Hirschman, L., Gaizauskas, R.: Natural language question answering: the view from here. Nat. Lang. Eng. **7**(4) (December 2001) 275–300
2. Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A., Lally, A., Murdock, J.W., Nyberg, E., Prager, J., Schlaefer, N., Welty, C.: Building watson: An overview of the deepqa project. AI Magazine (2010) 59–79
3. Kaufmann, E., Bernstein, A., Fischer, L.: NLP-Reduce: A ”naive” but Domain-independent Natural Language Interface for Querying Ontologies. (2007)
4. Lopez, V., Uren, V., Sabou, M., Motta, E.: Is question answering fit for the semantic web? a survey. Semantic Web ? Interoperability, Usability, Applicability **2**(2) (September 2011) 125–155
5. Prager, J., Brown, E., Coden, A., Radev, D.: Question-answering by predictive annotation. In: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval. SIGIR ’00, New York, NY, USA, ACM (2000) 184–191
6. Miller, G.A.: Wordnet: a lexical database for english. Commun. ACM **38**(11) (November 1995) 39–41

7. Pustejovsky, J.: Type coercion and lexical selection. In: Semantics and the Lexicon. Kluwer Academic Publishers, Dordrecht, The Netherlands (1993)
8. Lally, A., Prager, J.M., McCord, M.C., Boguraev, B.K., Patwardhan, S., Fan, J., Fodor, P., Chu-Carroll, J.: Question analysis: How watson reads a clue. IBM Journal of Research and Development **56**(3.4) (may-june 2012) 2:1 –2:14
9. Chu-Carroll, J., Fan, J., Boguraev, B.K., Carmel, D., Sheinwald, D., Welty, C.: Finding needles in the haystack: Search and candidate generation. IBM Journal of Research and Development **56**(3.4) (may-june 2012) 6:1 –6:12
10. Gondek, D., Lally, A., Kalyanpur, A., Murdock, J., Duboue, P., Zhang, L., Pan, Y., Qiu, Z., Welty, C.: Finding needles in the haystack: Search and candidate generation. IBM Journal of Research and Development **56**(3.4) (may-june 2012) 14:1 –14:12
11. Kalyanpur, A., Murdock, J.W., Fan, J., Welty, C.A.: Leveraging community-built knowledge for type coercion in question answering. In Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N.F., Blomqvist, E., eds.: International Semantic Web Conference (2). Volume 7032 of Lecture Notes in Computer Science., Springer (2011) 144–156
12. Murdock, J.W., Kalyanpur, A., Welty, C., Fan, J., Ferrucci, D.A., Gondek, D.C., Zhang, L., Kanayama, H.: Typing candidate answers using type coercion. IBM Journal of Research and Development **56**(3.4) (may-june 2012) 7:1 –7:13
13. Hearst, M.A.: Automatic acquisition of hyponyms from large text corpora. In: Proceedings of the 14th conference on Computational linguistics - Volume 2. COLING '92, Stroudsburg, PA, USA, Association for Computational Linguistics (1992) 539–545
14. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: Proceedings of the 16th international conference on World Wide Web. WWW '07, New York, NY, USA, ACM (2007) 697–706
15. Fan, J., Kalyanpur, A., Gondek, D.C., Ferrucci, D.A.: Automatic knowledge extraction from documents. IBM Journal of Research and Development **56**(3.4) (may-june 2012) 5:1 –5:10
16. Voorhees, E., ed.: Overview of the TREC 2006 Conference, Gaithersburg, MD (2006)
17. Schlobach, S., Ahn, D., de Rijke, M., Jijkoun, V.: Data-driven type checking in open domain question answering. J. Applied Logic **5**(1) (2007) 121–143
18. Grappy, A., Grau, B.: Answer type validation in question answering systems. In: Adaptivity, Personalization and Fusion of Heterogeneous Information. RIAO '10, Paris, France, France, LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE (2010) 9–15
19. Aktolga, E., Allan, J., Smith, D.A.: Passage reranking for question answering using syntactic structures and answer types. In et al., P.C., ed.: ECIR 2011, LNCS 6611. (2011) 617628
20. Buscaldi, D., Rosso, P.: Mining Knowledge from Wikipedia from the question answering task. In: Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006), Genoa, Italy (2006)