Typing candidate answers using type coercion

J. W. Murdock
A. Kalyanpur
C. Welty
J. Fan
D. A. Ferrucci
D. C. Gondek
L. Zhang
H. Kanayama

Many questions explicitly indicate the type of answer required. One popular approach to answering those questions is to develop recognizers to identify instances of common answer types (e.g., countries, animals, and food) and consider only answers on those lists. Such a strategy is poorly suited to answering questions from the Jeopardy!™ television quiz show. Jeopardy! questions have an extremely broad range of types of answers, and the most frequently occurring types cover only a small fraction of all answers. We present an alternative approach to dealing with answer types. We generate candidate answers without regard to type, and for each candidate, we employ a variety of sources and strategies to judge whether the candidate has the desired type. These sources and strategies provide a set of ***type coercion*** *scores for each candidate answer. We use these scores to give preference to answers with more evidence of having the right type. Our question-answering system is significantly more accurate with type coercion than it is without type coercion; these components have a combined impact of nearly 5% on the accuracy of the IBM Watson™ question-answering system.*

Introduction

The Jeopardy!** question "In 1902 Panama was still part of this country" explicitly indicates that the correct answer is a country. To answer questions such as this one, it is important to be able to distinguish between candidate answers that are countries and those that are not. Many open-domain question-answering (QA) systems (e.g., [1–4]) adopt a *type-and-generate* approach by analyzing incoming questions for the expected *answer type*, mapping it into a fixed set of known types, and restricting candidate answers retrieved from the corpus to those that match this answer type (using type-specific recognizers to identify the candidates).

The type-and-generate approach suffers from several problems. 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 sets of questions that cover a very broad range of topics. Answer types in Jeopardy! are extremely diverse and are expressed using a variety of lexical expressions (e.g., "scarefest" when referring to horror movies) and are sometimes vague (e.g., "form") or meaningless (e.g., "it"). When questions ask for types of answers not covered by the fixed set of types, the QA system either fails to generate answers at all or uses some catchall type (e.g., "OTHER") for which the rest of the system is typically not well suited. Performance on questions whose answer types are outside the fixed type system is significantly worse than when the answer type is in the type system. Even when the system does have a type that is appropriate, the type-and-generate approach is highly dependent on the precision and the recall of the typing component. That component acts as a candidate selection filter; thus, any answer that it rejects cannot be considered at all, regardless of how much other evidence supports it.

In contrast to the type-and-generate approach, IBM Watson* uses a *generate-and-type* framework. This approach has implications not only for the typing components but also for other parts of the technology underlying Watson: DeepQA. Many of the DeepQA search and candidate generation components do not make use of type information when identifying candidate answers [5]. As a result, many candidate answers are generated without any attempt to determine whether that candidate is an instance of the type that the question is asking for. Instead, reasoning about the type of an answer is performed later in the DeepQA architecture. The portion of the DeepQA architecture

Digital Object Identifier: 10.1147/JRD.2012.2187036

0018-8646/12/$5.00 © 2012 IBM

in which candidate answers are evaluated is hypothesis and evidence scoring. The subset of hypothesis and evidence scoring that focuses on determining whether the candidate answer satisfies the answer-type requirements of the question is *type coercion* (TyCor). Results from each of the TyCor components are then treated as distinct features used by a statistical classifier, along with features from many other components; the classifier is used to generate a confidence score for each answer during DeepQA's final merging and ranking.

The term "type coercion" has been used with different meanings in other contexts. In programming languages, "type coercion" refers to the idea that one can force some value with one data type to change to a different but compatible data type by putting it into a context where that other data type is required [6]. For example, in many programming languages, the expression $0.5 + 7$ would require that the compiler or the interpreter convert the integer, i.e., 7, into a floating-point number, i.e., 7.0, before adding it to another floating-point number; that implicit change of type is called *type coercion*. Pustejovsky [7] describes a similar phenomenon in linguistics, in which a speaker can take a noun with one semantic type and imply a different but compatible semantic type by putting it into a context where that other semantic type is required. For example, the word "book" in the sentence "Bob finished a book" refers to a physical object. This sentence coerces an interpretation of "book" as an activity, i.e., the implied activity of reading the book. As with that of Pustejovsky, our paper involves forcing a particular semantic-type interpretation on some instance. However, "coercion" in our paper involves forcing an interpretation of some answer to a question based on the type of answer that the question asks for. For example, if a question asks "What novel won the Pulitzer in 1937?" and a respondent asserts "Gone with the Wind", the respondent is implicitly forcing a specific interpretation of "Gone with the Wind", i.e., the novel. Some possible answers cannot be coerced into some types. for example, there is no interpretation of "Margaret Mitchell" that refers to a novel; thus, this could not be a valid answer to this question. Our TyCor components attempt to coerce consistent interpretations of the candidate answer and the desired type. To the extent these components are able to do so, DeepQA treats the result as evidence that the answer could be correct.

Watson includes numerous TyCor components that employ different sources of typing information and different logic, but all that fit into the logical framework described in this paper. There are two inputs to each TyCor component.

- The *lexical answer types* (LATs) that the question is asking for, as identified by DeepQA's question analysis module [8]. A LAT is a text string indicating the type of answer being sought (e.g., "actor," "country," and "scarefest").
- A candidate answer from DeepQA's candidate generation module [5].

The output of each TyCor component is a numerical score indicating the extent to which that component has concluded that the candidate answer is an instance of the type indicated by the LAT. Unlike many QA systems, Watson does not use answer-typing results as a "hard filter," discarding that any answer that it cannot conclude has the desired type. Instead, each TyCor score is a distinct feature that is used by DeepQA's statistical answer ranking algorithm to assign a confidence value to each candidate answer and to rank the answers according their confidence value [9]. Correct answers in the training data are likely to have evidence that they are instances of the desired type (since they are). As a result, DeepQA's statistical answer ranking algorithm tends to prefer answers with higher TyCor scores over answers with lower TyCor scores, but this preference is not absolute, and an answer with little or no evidence that it has the desired type can be still selected as the final answer, if there is overwhelming evidence that this answer satisfies the other requirements of the question (particularly when the other candidate answers also have little or no TyCor evidence).

There is no single fixed set of types that is used by all of the TyCor components. Instead, each TyCor component is responsible for interpreting the LAT to the extent that it needs to do so. Some TyCor components have a fixed list of structured types that they are able to process; those components are not useful at all when they are not able to map the LAT to one of their types. Other TyCor components have instances tagged with types in the form of text strings; those components compute whether the LAT is consistent with the known types for a candidate answer by matching the LAT to those types linguistically (e.g., using dictionary resources to identify synonyms).

This paper begins with a discussion of answer typing in Jeopardy!. It then explains how this paper fits into the DeepQA architecture. Next, it describes the shared logical framework for TyCor. After that, it provides brief descriptions of some of our TyCor components. Finally, this paper presents evaluation results, related work, future work, and conclusions.

Answer types in Jeopardy!

In our attempt to build a QA system capable of rivaling expert human performance on answering open-domain questions, we started with 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 this approach to be problematic.

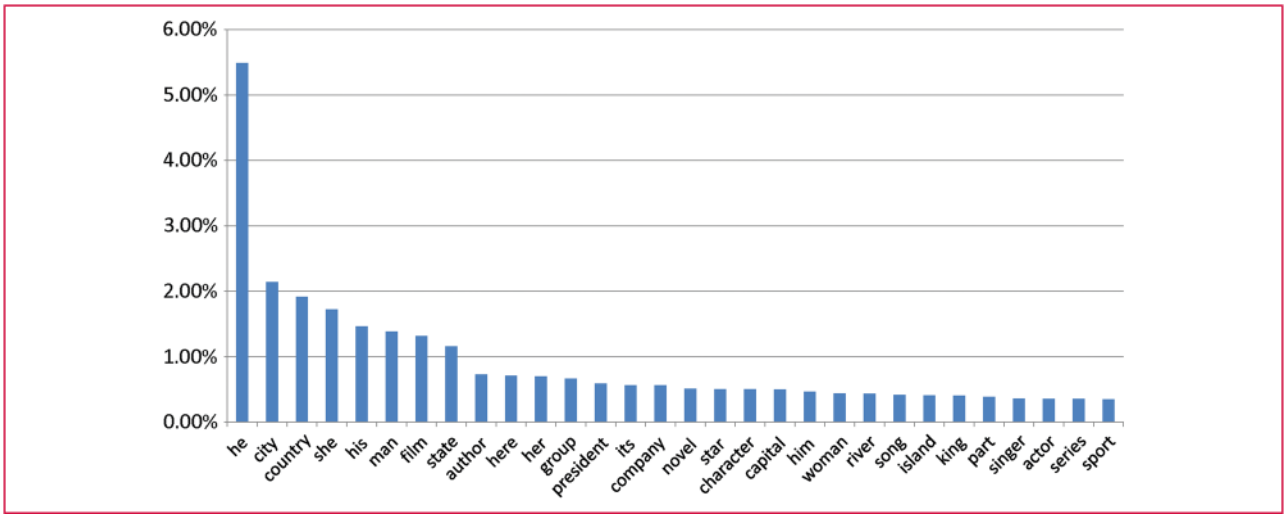

Figure 1

Distribution of the 30 most frequent lexical answer types in 20,000 Jeopardy! questions.

Human language is remarkably rich when it comes to assigning types; nearly any word can be used as a type, particularly in some questions.

- Invented in the 1500s to speed up the game, this *maneuver* involves two pieces of the same color. (Answer: "Castling")
- The first known airmail service took place in Paris in 1870 by this *conveyance.* (Answer: "hot-air balloon")
- In 2003, this Oriole *first sacker* was elected to the Baseball Hall of Fame. (Answer: "Eddie Murray")
- Freddy Krueger was introduced in this 1984 *scarefest.* (Answer: "A Nightmare on Elm Street")
- When hit by electrons, a phosphor gives off electromagnetic energy in this *form*. (Answer: "light")
- Whitney's patent for *this* revolutionized the garment industry. (Answer: "the cotton gin")

Such variability highlights one of the intrinsic problems with the type-and-generate approach, i.e., we cannot reliably predict what types the questions will ask about and what their instances are. We analyzed 20,000 past Jeopardy! questions and observed a *very long tail* of types (see **Figure 1**). Although the type system for our named entity detector was among the largest of the state-of-the-art QA systems (more than 100 types), it covered less than half the questions. Roughly 5,000 different type words were used in the 20,000 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 roughly constant; new types were being introduced at this rate (one in eight questions on the average). In addition, 15% of questions did not explicitly assert a LAT.

These observations led us to conclude that we need to be open and flexible about types, treating them as a property of question and answer combined. In other words, instead of finding candidates of the right type, we want to find candidates (in some way) and judge whether each one is of the right type by examining it in context with the answer type from the question. Furthermore, we need to accommodate sources of type and instance data that collectively reflect the same descriptive diversity as these questions.

TyCor in the DeepQA Architecture

Our TyCor capabilities fit into the DeepQA architecture [10].

Question analysis—DeepQA's question analysis includes many subcomponents that classify and extract relevant information from the question [8]. One kind of information extracted during question analysis is a *LAT*, i.e., a word in the question that indicates the type of answer being sought. LAT recognition is easier than mapping to a semantic type; although imperfect, our LAT detection has an F1 measure of 0.8 (evaluated on 3,500 randomly selected questions [8]). LAT detection includes a confidence measure, which is factored into the TyCor scores.

Candidate generation—Candidate generation in DeepQA employs a wide variety of strategies including identifying candidates in both text and structured sources [5]. In some cases, candidate generation results in a disambiguated entity, e.g., one for which a Wikipedia** URL has been

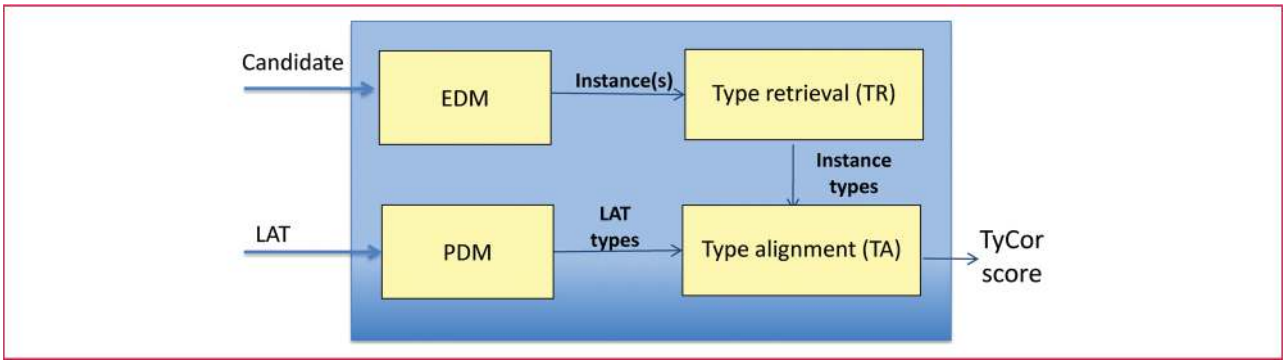

Figure 2

High-level architecture of the TyCor component involving four core steps. For example, in YAGO TyCor, given candidate "difficulty swallowing" and lexical answer type (LAT) "manifestation", EDM maps candidate to DBpedia entity "Dysphagia"; TR obtains WordNet type "Symptom" for the DBpedia instance; PDM maps LAT to WordNet concept "Condition"; and the final TR step finds a hyponymy relation between "Symptom" and "Condition" producing a positive TyCor score.

identified. For example, if the candidate answer is found in a Wikipedia document as the anchor text of some hypertext link, candidate generation can use the target of that link to provide a disambiguation for the entity. The inputs to TyCor include these candidate answers (with disambiguation results, if available) plus the LATs identified in the question analysis.

Hypothesis and evidence scoring—The TyCor components are a subset of the hypothesis- and evidence-scoring components. During the hypothesis- and evidence-scoring phase, many different algorithms and sources are used to evaluate evidence for each candidate answer.

Final-answer merging and ranking—An overall determination of the final answer must combine the scores from each scoring algorithm for each answer in a way that weighs each score as appropriate for the context given by the question [9]. The TyCor scores are among the many features used by this component.

TyCor logical framework

The TyCor answer-scoring components take as input one or more LATs and a candidate answer. They each return a score indicating the strength of the evidence that it is possible to coerce some interpretation of the candidate answer into some instance that is consistent with some interpretation of the LATs. Since language does not often distinguish between instantiation (e.g., "Secretariat was a horse") and subclassing (e.g., "A pony is a horse"), the TyCor components must allow for this; TyCor gives an answer a high score if it can be interpreted as a subclass or an instance of a LAT.

For each LAT, the TyCor component performs the four steps illustrated in **Figure 2** and described in detail below. The TyCor component then combines scores across LATs to produce a score for the candidate answer. The four steps involve using some source (e.g., a knowledge-base) to determine whether the source indicates that the answer has the desired type. This involves mapping the candidate answer and LAT to instances and types in the source, and then consulting the source to see if it claims that some instance corresponding to the candidate answer is consistent with some type corresponding to the LAT. Specifically, here are the four steps.

Entity disambiguation and matching (EDM)—EDM finds entities in the typing source that correspond to the candidate answer. EDM 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 [11] encodes useful naming information in the entity identifier (ID). EDM implementations typically try to use some context for the answer, but in purely structured sources, this context may be difficult to exploit.

Type retrieval (TR)—TR retrieves the types for each entity identified by EDM. For some TyCors, such as those using structured sources, this step exercises the primary function of the source and is simple. In unstructured sources, this may require parsing [12] or other semantic processing [13] of the natural language.

Predicate disambiguation and matching (PDM)—PDM identifies types that correspond to the LAT found. In some sources, this is the same algorithm as EDM; in others, type lookup requires special treatment because those sources encode types and instances differently. In TyCors that use unstructured information as a source, the PDM step may simply return the LAT itself. PDM strongly

corresponds to notion of word-sense disambiguation with respect to a specific source.

Type alignment—The results of the PDM and TR steps must be then compared to determine the degree of match. In sources containing a formal-type taxonomy, this may include checking the taxonomy for subsumption, disjointness, etc. For sources in which types are natural-language text, alignment may require determining whether the text of the LAT and the text of the retrieved type for the answer are consistent. This is a challenging natural-language processing (NLP) task that uses parsing [12] and depends on resources such as WordNet** [14] for finding synonyms, hypernyms, etc.

Note that EDM is skipped if candidate generation was able to produce a disambiguated candidate answer with a specified unique ID that the TyCor is able to use. For example, the Wiki-Category TyCor (described later) has types associated with specific Wikipedia entries; thus, the Wiki-Category TyCor will skip EDM when it encounters a candidate answer that was generated with a known Wikipedia URI.

EDM, PDM, and TR can each return multiple results. For example, one could start with a candidate string "Gone with the Wind" and a LAT "book"; EDM could tie the candidate to a variety of entities, one of which is a movie and one of which is a novel. We may have more or less confidence in each of those entities, and those confidence scores are factored into the TyCor score. Similarly, each of those candidates can have multiple types found in TR. Moreover, the LAT "book" could correspond to a type of published work, a phenomenon in graph theory, or something else. Type alignment attempts to align all of the types retrieved by TR (for all of the entities identified by EDM) to all of the types identifies by PDM.

We refer to TyCor as "coercing" the candidate answer to the LAT because we are not committing early to a single interpretation at the EDM, PDM, and TR stages. For example, instead of first disambiguating "Gone with the Wind" and *then* deciding if it is an instance of the type we want, we remain open to a variety of possibilities after EDM and consider the types for all of those possibilities. Thus, the type alignment step is able to "force" a particular interpretation of both the candidate and the LAT by identifying the interpretations of each that best fit each other and basing its conclusions on those interpretations.

These four steps constitute a logical framework, i.e., conceptually all of our TyCor components follow this progression of steps. Many subsets of the TyCor components share common implementations of one or more of these steps. For example, several of our TyCor sources (described in the next section) provide type information for entities defined by Wikipedia URLs; the TyCor components that use those sources share a common EDM implementation. In future work, we intend to formalize and refine this framework (see the section on future work).

TyCor components that do not find any evidence that the candidate answer has the desired type indicate a neutral result (i.e., they neither support nor refute the answer). In addition, a few TyCor components are able to identify specific evidence that the answer does *not* have the desired type. TyCor components that are able to identify negative evidence are described as such in the next section. The logic used to identify negative evidence is generally different from the logic used to identify positive evidence. There is no *a priori* reason to believe that a model should treat a strong negative TyCor score as being as bad as a strong positive TyCor score is good. Consequently, TyCor represents positive and negative scores as distinct features for use by DeepQA answer ranking [9]. TyCor components that consider negative typing evidence have two features in the final model, and these two features are constrained by the framework such that each TyCor component can have only one of these two features with a nonzero value per candidate answer. It is possible that different TyCor components produce conflicting scores, e.g., one produces positive evidence and another produces negative evidence. Generally speaking, negative typing evidence will reduce confidence in an answer, but it will not remove the answer or invalidate it. Since there is always error, it must be possible for other evidence in Watson to override negative typing evidence that may be incorrect.

TyCor sources and strategies

Watson uses a suite of more than ten different TyCor components for scoring candidate answers against type evidence. Some of our TyCor components share algorithms but use different sources, whereas 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 toward accurately accounting for the error in the steps (most notably EDM and alignment) to produce a meaningful score.

Some TyCor components have a well-defined set of structured types. Some examples of TyCor components of this sort are given.

YAGO (Yet Another Great Ontology)—Many candidate answers in our domain are titles of Wikipedia articles. Each of these is an entity in DBpedia, i.e., a linked open data source automatically compiled from Wikipedia infoboxes (i.e., information boxes) and article templates. Entities (articles) in DBpedia have types represented in a resource description framework (RDF) from YAGO [15], i.e., a semi-automatically constructed type taxonomy based on WordNet, corpus analysis, and Wikipedia. In addition, we have manually added roughly 200 disjointness constraints (e.g., "a Person is not a

Country") between high-level concepts 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.

Gender—This TyCor applies only to questions asking for a person of a specified gender. It scores the evidence that a candidate answer is of the appropriate gender using a custom source of data mined from articles about people by determining which pronouns are most unambiguously or most commonly used to refer to the person. Gender TyCor can produce negative evidence if the LAT indicates one gender and the answer is found to be of another.

Closed LAT—Certain LATs identify types with enumerable lists of instances, such as countries, U.S. states, and U.S. presidents. 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 because of name-matching issues and the possibility that the LAT is used in a nonstandard 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. Because all of the lists are believed to be complete (at least for the most obvious interpretation of the LAT), closed LAT TyCor asserts negative evidence for a candidate answer if it knows the LAT and does not have that answer on its list.

Lexical—Occasionally, LATs specify some lexical constraint on the answer, e.g., that it is a verb, or a phrase, or a first name. Lexical TyCor uses various special-purpose algorithms based on the LAT for scoring these constraints. Lexical TyCor is able to produce negative evidence for some types (e.g., it can conclude that any answer that contains no blank space is not consistent with the LAT "phrase").

Named entity detection (NED)—TyCor uses a rule-based named entity detector that was originally designed for a classical type-and-generate QA system [16]. That named entity detector recognizes instances of more than 100 structured types, most of which are among the top 100 LATs. The named entity detector identifies zero or more structured types corresponding to the LAT, and it annotates the candidate answer with zero or more structured types that it instantiates. NED TyCor determines whether any structured type that the detector found for the LAT is consistent with any structured type that the detector found for the candidate answer.

WordNet—WordNet is used in several 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 and a few geographic and geopolitical entities. It also has high coverage of biological taxonomies. WordNet TyCor uses both hyponym and instance-of links in WordNet to match the candidate answer string to the LAT. This TyCor component has very high precision but low recall.

Other TyCor components have types that are arbitrary natural-language text. Type alignment for these TyCors requires processing that natural language. We do this by aligning terms with corresponding positions in the syntactic structure of the types (as recognized by DeepQA's parsing and predicate-argument structure [12]). Once terms are aligned, we determine whether they are consistent using a variety of sources such as Wikipedia redirects and WordNet. Some examples of TyCor components of this sort are given.

Wiki-Category—Wikipedia articles are frequently tagged with explicit categories in the form of natural-language text. All of these categories are stored in DBpedia. The Wiki-Category TyCor uses the category names for an entity as types. Wiki-Category does not use the category structure (e.g., subcategory), because this adds too much noise.

Wiki-List—Wikipedia and many other Web sources contain lists of things associated in some way, such as "List of Argentinean Nobel Prize Winners". We collect these lists and use the text following "List of" as types for the instances on the list.

Wiki-Intro—By convention, the first sentence of a Wikipedia article identifies one or more types for the entity described in the article, e.g., "Tom Hanks is an American actor, producer, writer, and director" or "The lion (Panthera leo) is one of the four big cats in the genus Panthera". Wiki-Intro TyCor utilizes a special source mined from these intro passages using a variety of syntactic patterns.

Identity—Identity TyCor uses the candidate answer text as a source of typing information. For example, Identity TyCor can recognize that "the Chu River" is a river by looking at the text of the answer, without any prior knowledge of that entity.

Passage—Many candidate answers occur in a passage of text found either in primary search [5] or in supporting evidence retrieval [17]. Occasionally, that passage asserts the candidate's type, e.g., "*Christian Bale* is the first *actor* to really do Batman justice". Passage TyCor uses pattern-based relation detection [13] to identify assertions that some entity has some type and attempts to match the asserted type to the LAT.

PRISMATIC—PRISMATIC [18] is a repository of corpus statistics. PRISMATIC TyCor measures the frequency with which the candidate answer is directly asserted to be an instance of the LAT (using the same type assertion detection patterns used in Passage TyCor).

Example

The four steps of the TyCor logical framework are differently implemented in the various TyCor strategies. For example, consider a question asking for an "emperor" and the candidate answer "Napoleon." Here is how a few of the TyCor strategies handle different steps of the framework for this example.

EDM—TyCor strategies that use Wikipedia or derived sources (e.g., Wiki-List, Wiki-Category, and YAGO) try to determine what DBpedia entry "Napoleon" refers to. They find *dbpedia:Napoleon* is as a strong match. They also find a variety of other matches such as *dbpedia:Napoleon_%28card_game%29* (for which it assigns a lower score). The WordNet TyCor looks up "Napoleon" and finds three different word senses. Passage TyCor identifies occurrences of Napoleon in some supporting passage during this step. Other TyCors such as NED and Identity TyCor simply take the candidate string "Napoleon" and declare that to be the entity.

TR—Those TyCor strategies that identify a formal entity [e.g., a DBpedia URL or a WordNet synset (or set of synonyms)] during EDM are generally able to look up one or more types for that entity in some structured source. For example, YAGO TyCor is able to find formal YAGO types for the DBpedia entities identified during EDM. Similarly, WordNet TyCor finds other synsets that the synsets for Napoleon are either instances or hyponyms of; in this case, WordNet labels the primary sense of "Napoleon" as an instance of the primary sense of "emperor". NED TyCor gets a structured type from the named entity detector, e.g., it labels "Napoleon" as a NAME reference to a PoliticalLeader. Some TyCors produce types in this stage, which are not entries in a formal ontology but rather are simply text strings. For example, Wiki-List provides the type "French monarch" for *dbpedia:Napoleon* because there is a page on Wikipedia labeled "List of French monarchs" that has an entry that links to http://en.wikipedia.org/wiki/Napoleon_I_of_France, which is a Wikipedia redirect to http://en.wikipedia.org/wiki/Napoleon. Those TyCor strategies that produce text strings for types defer the issue of making sense of that type until the type alignment step.

PDM—TyCor strategies that use formal types look up those types in this step. For example, WordNet looks up the LAT "emperor" and identifies four synsets for that word. YAGO TyCor finds formal types in the YAGO ontology corresponding to the string "emperor". NED TyCor examines the NED results on the question and determines whether the LAT in the question was labeled with some named entity type. In this case, "emperor" is marked as a NOMINAL reference to a PoliticalLeader. As with TR, some TyCors implement PDM by simply returning the input string, deferring the work of making sense of that string.

Type alignment—For TyCors such as WordNet, YAGO, and NED that produce formal types in a type hierarchy from TR and PDM, the type alignment process involves aligning types in the hierarchy, i.e., checking for subsumption, disjointness, etc. For example, WordNet TyCor checks to see whether the entity type (from TR) is a hyponym of the question's desired type (from PDM). In contrast, those strategies that simply produce text strings from TR and PDM need to determine the extent to which the known entity type (from TR) implies the desired answer type (from PDM). For example, Wiki-List tries to determine whether it can conclude that a "French monarch" is an "emperor" by parsing both (e.g., finding that "monarch" is the headword of "French monarch") and matching terms using resources such as WordNet and Wikipedia redirects. Many of these strategies use a single configurable implementation of this capability but use a variety of different configurations, which provide different tradeoffs among precision and recall.

Evaluation

We have evaluated our TyCor mechanisms by comparing the effectiveness of our QA system with and without TyCor components. All experiments reported here were performed on a set of 3,508 previously unseen Jeopardy! questions.

Figure 3 shows a line graph comparing the system with all of the TyCor strategies versus the system with none, and a bar graph showing the impact of our most effective individual TyCor strategies. The horizontal axis of the line graph shows the percentage of questions answered, with preference given to questions where the confidence in the answer is highest. For example, the 70% point on the axis shows how the system performs when it attempts to answer only 70% that it is most confident of (we refer to this value as *Precision@70*). The vertical axis of the line graph indicates the fraction of those questions that are correctly answered. For example, the line for the complete Watson system with all of the TyCor components shows a precision of 0.875 at 70% of the questions answered; for 70% of the questions for which Watson was most confident in its answer, it answered 87.5% of those questions correctly. In contrast, the full Watson system without TyCor answered only 81.5% of questions correctly for the 70% for which it was most confident. The difference, i.e., 6.0%, in Precision@70 is statistically significant and is greater than the impact on the overall accuracy (i.e., Precision@100), which is 4.9%. The greater impact at 70% than 100% suggests that TyCor is even more useful for assessing confidence in Watson's answers than it is for selecting answers; when Watson is able to use its confidence score to decide which questions to attempt to answer, it benefits even more from TyCor.

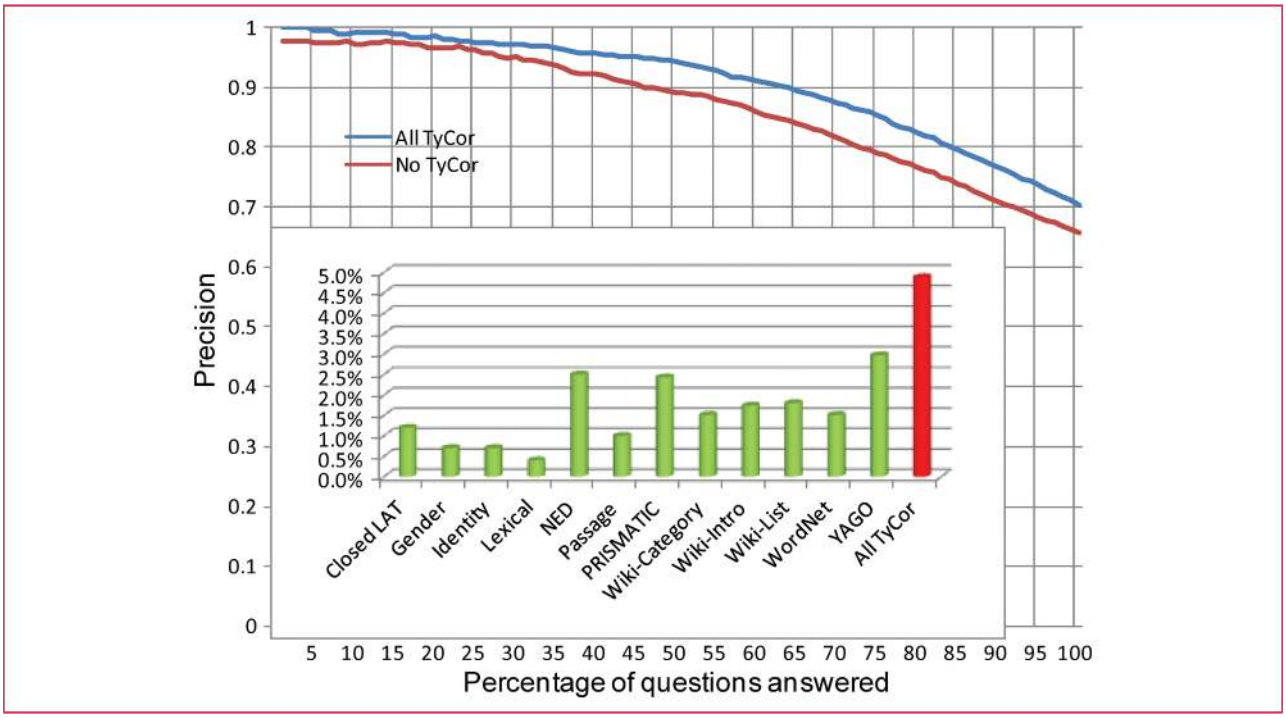

Figure 3

Impact of type coercion on the complete Watson QA system.

The bar graph shows the impact on accuracy of 12 of our most interesting TyCor components. We define *impact on accuracy* here as the difference between the accuracy of the QA system with no TyCor components and the accuracy of the system with only the specified TyCor component (accuracy is the percentage of all questions that are correctly answered). The first 12 bars show the impact of individual TyCor components, and the final bar shows the impact of all of the TyCor components together. The individual components shown have varying impact on accuracy of up to 3%, and all of them together have an impact on accuracy of 4.9%. That difference is also visible on the graph, by comparing the rightmost points on the lines with no TyCor and all TyCor.

We measure statistical significance for accuracy using McNemar's test with a correction for continuity [19], and we consider $p < .05$ to be significant. By that standard, the difference between Lexical TyCor and no TyCor is not statistically significant. This is not surprising because Lexical TyCor is a highly specialized component that addresses a small number of unusual LATs. Each of the other TyCors shows a significant impact versus no TyCor, and all TyCor shows a significant impact versus any of the TyCors alone. Precision@70 is not amenable to McNemar's test because it does not reflect a mean over a set of independent observations (e.g., raising the confidence to an answer for one question can cause some other question to drop out of the 70% with the highest confidence). Consequently, we use Fisher's randomization test [20] to assess significance for this metric.

One concern that we have with the results of ablating TyCor from the full Watson system is that some of the impact of TyCor is blunted by the existence of other scoring components in Watson that are not explicitly focused on TyCor but do implicitly correlate with answers having the correct type. For example, the Passage Term Match algorithm [17] counts the frequency with which each candidate answer co-occurs in retrieved passages with terms in the question. Since the LAT is in the question, it is one of the terms that is matched and counted by Passage Term Match. In many cases, we would expect answers that are instances of the LAT to frequently co-occur with the LAT in text; thus, we would expect the signal from the Passage Term Match feature to overlap with the signal that our TyCor components produce, blunting their measured impact in ablation studies. Consequently, we have further evaluated our TyCor components on the Watson answer-scoring baseline system [10], which is also used for other DeepQA evaluations [13, 17, 21]; that baseline includes all of the DeepQA question analysis [8] plus search and candidate generation [5], but no deep and shallow evidence scoring other than NED TyCor. Answer ranking in the Watson

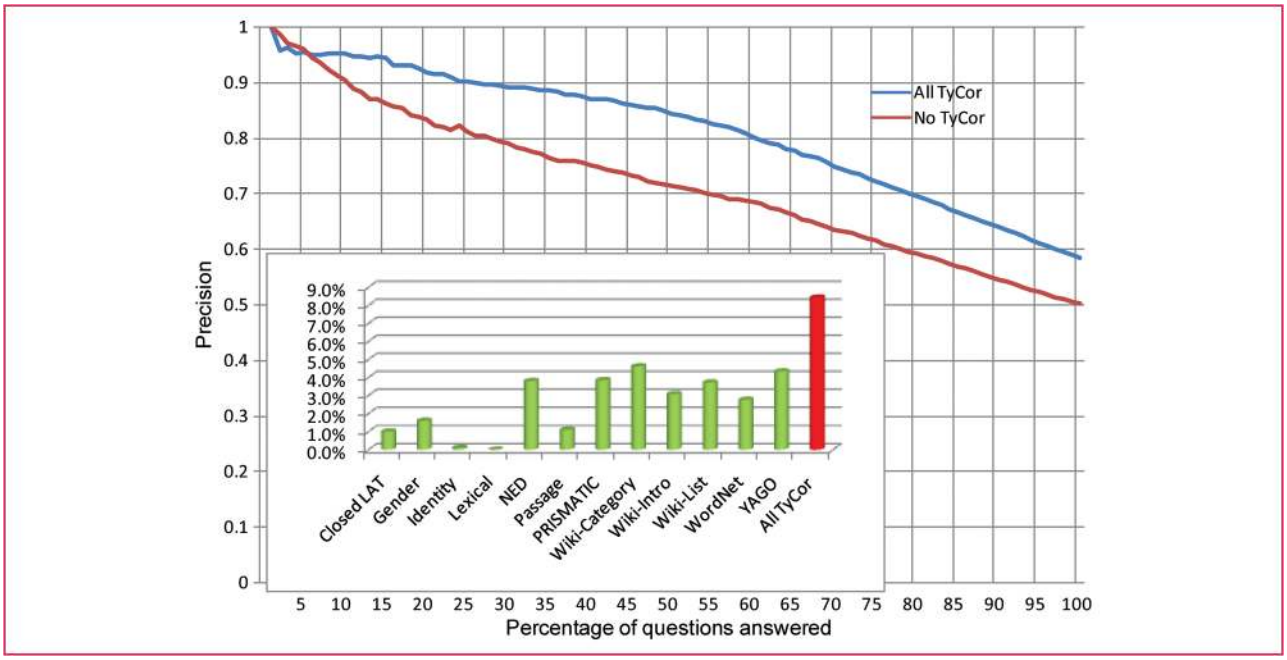

Figure 4

Impact of type coercion on the Watson answer-scoring baseline QA system.

answer-scoring baseline system uses the same statistical answer-ranking techniques as the full Watson system but has a much more restricted set of features to work with. It has all of Watson's search and candidate generation features (e.g., a rank and a score from a keyword search engine for a passage where the answer was found). However, it does not have any features that are derived from the deep analysis of supporting evidence.

Starting with the Watson answer-scoring baseline, we compare a variant with no TyCor (not even the TyCor with NED) to variants with one TyCor component added, as well as to the baseline system with all TyCor components added. **Figure 4** shows those results. TyCor has a greater impact in this simpler configuration, with some components showing more than 4% impact and the entire set showing more than 8%. Each of the components except Identity TyCor and Lexical TyCor provides a significant impact versus no TyCor, and again, all TyCor provides a significant impact versus any one TyCor.

The Watson answer-scoring baseline system includes some signal that overlaps with answer typing, because several of the search and candidate generation algorithms make use of the LAT in finding sources of candidate answers [5]. Consequently, we built an extremely simple "ultralite" baseline system as an additional point of comparison for TyCor. The ultralite system includes only text document search [5]; it does not search through any knowledge-bases or text passages. **Figure 5** shows the effectiveness of TyCor in that configuration. The effect on accuracy is even greater, with some components showing 8% to 10% and all of the components together resulting in an impact of more than 15%. Again, each of the components except Lexical TyCor provides a significant impact versus no TyCor, and again, all TyCor provides a significant impact versus any one TyCor.

As noted earlier, the NED TyCor uses a NED component that was originally built for a classical type-and-generate QA system that was consistently one of the highest performers in competitive QA evaluations [1]. As such, the comparison between the NED TyCor performance and the all TyCor performance in Figures 3–5 demonstrates the impact of our additional sources and strategies versus a traditional QA baseline. That difference is 2.4% in the full Watson system, 4.6% in the Watson answer-scoring baseline system, and 8.5% in the ultralite baseline system; those differences are all significant. We have also conducted experiments in which we completely discarded candidates that are rejected by NED TyCor, to more closely approximate a type-and-generate approach (for comparison purposes). Those experiments have shown that discarding candidates rejected by NED TyCor performs worse than using NED TyCor as the only TyCor feature and even worse than *no* TyCor at all, because it discards many correct answers that DeepQA is able to select using other features.

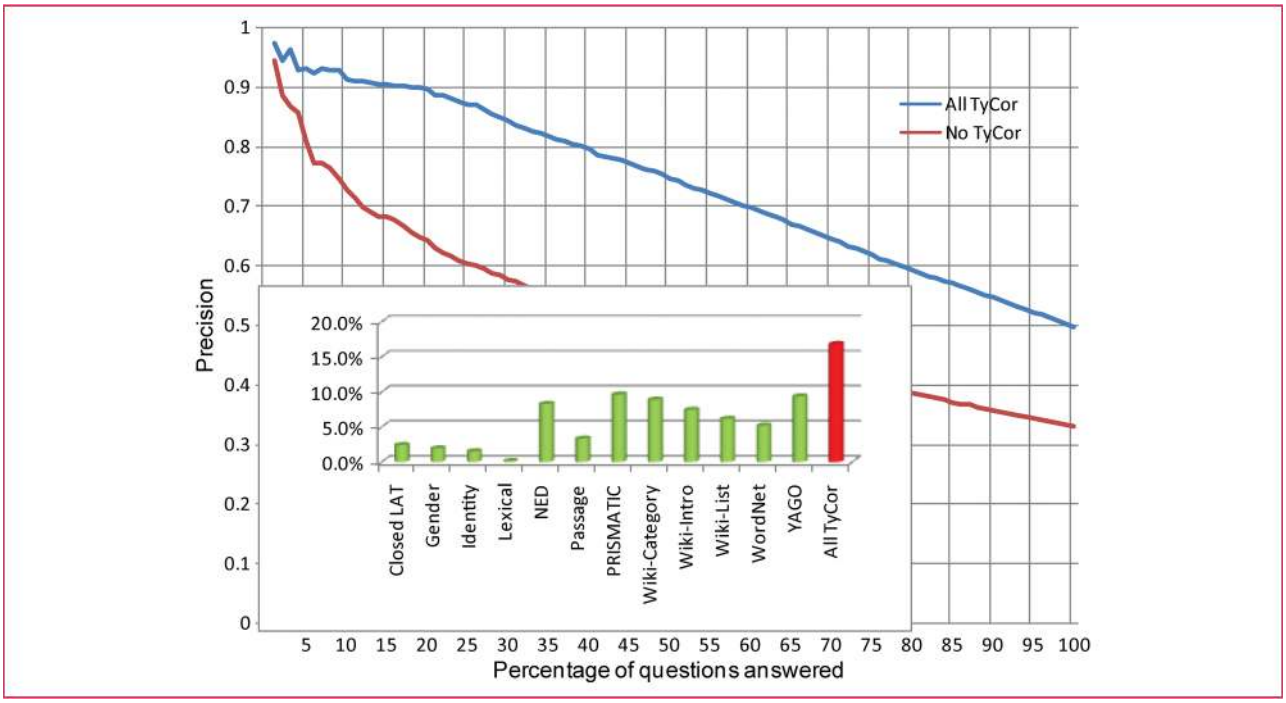

Figure 5

Impact of type coercion on the "ultralite" baseline QA system.

Related work

Our paper differs in two major respects from existing work on answer typing for QA.

- We employ a diverse collection of typing resources, including highly precise but narrow coverage hand-crafted resources (e.g., NED, closed LAT), broad community generated resources (e.g., Wiki-Category, YAGO), and resources automatically extracted from natural-language text (e.g., Wiki-Intro, PRISMATIC). This diversity allows us to cover a very wide range of types while still having very high confidence for those types that are well understood.
- We do not discard or ignore answers that we cannot coerce to the given LAT. Instead, we retain all answers; results from each TyCor are provided as a distinct feature that is used by DeepQA answer ranking.

Traditional QA systems (e.g., [1–4]) have tightly integrated capabilities for finding candidate answers and answer typing, so that only strings that appear to have the desired answer type are ever considered as candidate answers. There is some existing work that has decoupled typing and generation to some extent. QUARTZ [22] is a QA system that uses a statistical mapping from LATs to WordNet for PDM, and collocation counts for the candidate answer with synonyms of the mapped type for TR. In [23], 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 [22] and [23] are similar to our TyCor approach in which they defer type-checking decisions to the latter in the QA pipeline and use a collection of techniques and resources (instead of relying on classical NED) to check for a type match between the candidate and the expected answer type in the question. In our approach, the type match information is not used as a filter to discard candidate answers; instead, the individual TyCor scores are combined statistically with other answer scores in DeepQA answer ranking. A similar approach to the combination of our NED and Wiki-Category is presented in [24]. In that work, the traditional type-and-generate approach is used when question analysis can recognize a semantic answer type in the question and revert to Wikipedia categories. As with [22] and [23], typing is treated as a hard filter, not as supplying features for classifying answers.

Another unique characteristic of our TyCor is the framework that separates the various steps of EDM, PDM, type alignment, etc. The algorithms (and the resources) that we use to implement these steps are complex and varied—having either much more precision or much broader

scope compared with existing work. For example, the only use of Wikipedia content for type inference in [23] 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 on the basis of 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, and link-anchor data. We then use different kinds of type information expressed in Wikipedia (e.g., lexical types in the introductory paragraphs and Wikipedia categories) as types to match to the LAT. Similarly, whereas [22] uses the notion of complement-type sets, which are approximated using heuristics such as sibling types, we have explicitly identified pairs of types in YAGO that are disjoint, and we use disjointness information as evidence against candidate answers whose types are disjoint with the LAT.

Our PRISMATIC, Wiki-Intro, and Passage TyCor components use NLP analyses to extract typing information from natural-language text. The automatic detection of typing relations is a long-studied topic in NLP [25–27]. A variety of existing projects attempt to use relation detection of this sort to build large-scale entity-type knowledge-bases [28, 29]. PRISMATIC and Wiki-Intro TyCors both follow this basic approach; they differ from each other in which PRISMATIC runs over a large Web corpus, whereas Wiki-Intro draws results exclusively from the first sentence of Wikipedia articles. As a result, PRISMATIC TyCor has broader coverage and has statistics for each entity-type pair (e.g., statistics indicating how often "The Godfather" is asserted to be a "film" and how often "The Godfather" is asserted to be a novel). In contrast, Wiki-Intro has much less data, but the data that it does have is very precise and is disambiguated (attached to a specific Wikipedia URL, e.g., http://en.wikipedia.org/wiki/The_Godfather is a "film"). Existing research on mining answer types from text is closer to PRISMATIC TyCor in this respect. As described earlier, we are able to benefit from the advantages of each by posting them (along with all of our other TyCor results) as distinct features for DeepQA statistical answer ranking.

Future work

As noted in the "TyCor logical framework" section of this paper, all of our TyCor components share a common logical framework consisting of four processing subtasks. The logical framework is not implemented as a common software artifact that is shared across all TyCors. In future work, we intend to build an explicit software framework that formalizes this design.

The fact that the logical architecture is not explicit in the implementation was convenient early in our development process because many of our TyCor components address dramatically different challenges with different information requirements. For example, type alignment in YAGO TyCor takes a pair of structured types as input, and type alignment in Wiki-List TyCor takes a pair of lexical types as input; the former involves reasoning in a formal ontology, whereas the latter involves NLP. This can be addressed by a careful object-oriented design, e.g., by defining a type alignment interface with abstract input types and providing different implementations of those input types to provide the different functionality required.

As work on TyCor has matured, the lack of an explicit implementation of the logical framework has become an increasingly significant obstacle to TyCor research. For example, we would like to be able to rapidly test different approaches to combining scores across the different steps of the process; doing so now requires modifying different pieces of code in different TyCor implementations. An explicit software framework would provide a common code base that integrates the steps of the TyCor process.

Given our experience with the diverse range of TyCors in Watson, we believe that we are now ready to design and implement an explicit software framework. The key challenge in this paper will be precisely identifying the proper level of abstraction, i.e., distinguishing capabilities and data structures that are shared across all implementations from those that are specific to particular TyCor algorithms and sources. We now have a reasonable, large, and diverse set of TyCor components that can provide motivating examples for that work.

Future work on DeepQA will involve a wide variety of concrete application areas. Consequently, flexibility and rapid adaptation to new technical challenges will be essential. In some cases, that will involve plugging new knowledge sources into existing logic, and in others, the logic will also require revisions. A pluggable extensible framework for TyCor components and subcomponents will make it easier to customize DeepQA to address new challenges.

Conclusion

TyCor is an approach to finding evidence regarding whether a given candidate answer has a specified lexical type (i.e., a type characterized by natural-language text). DeepQA includes a logical framework for TyCor and a variety of specific instantiations using an assortment of structured and unstructured sources. TyCor is used in DeepQA as a form of evidence scoring; it is performed after answers are generated and before they are ranked. The TyCor components provide separate scores that are used as distinct features by DeepQA answer ranking.

Our TyCor components all instantiate a logical framework composed of four elements. *EDM* maps candidate answer strings to entities. *TR* identifies (structured or lexical) types of entities. *PDM* maps LATs to answer types. *Type alignment* determines whether the types for the candidate answer

(from TR) are consistent with the desired type for the question (as determined by PDM).

The TyCor components have a significant impact on the accuracy of a QA system applied to the Jeopardy! task. The components complement each other, as demonstrated by the fact that our QA system does better with all of the TyCor components than it does with any one of them alone.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of Jeopardy Productions, Inc., Wikimedia Foundation, or Trustees of Princeton University in the United States, other countries, or both.

References

1. J. Chu-Carroll, K. Czuba, P. A. Duboue, and J. M. Prager, "IBM's PIQUANT II in TREC2005," in *Proc. 14th TREC*, Gaithersburg, MD, 2005. [Online]. Available: http://www.mendeley.com/research/ibms-piquant-ii-trec2005/
2. H. Cui, K. Li, R. Sun, T.-S. Chua, and M.-Y. Kan, "National University of Singapore at the TREC-13 question answering main task," in *Proc. TREC*, Gaithersburg, MD, 2004.
3. S. Harabagiu, D. Moldovan, C. Clark, M. Bowden, J. Williams, and J. Bensley, "Answer mining by combining extraction techniques with abductive reasoning," in *Proc. TREC*, Gaithersburg, MD, 2003.
4. N. Schlaefer, P. Gieselmann, and G. Sautter, "The Ephyra QA System at TREC 2006," in *Proc. 15th TREC*, 2006, pp. 1–10.
5. J. Chu-Carroll, J. Fan, B. K. Boguraev, D. Carmel, D. Sheinwald, and C. Welty, "Finding needles in the haystack: Search and candidate generation," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 6, pp. 6:1–6:12, May/Jul. 2012.
6. G. Ford and R. Wiener, *Modula-2: A Software Development Approach*. Hoboken, NJ: Wiley, 1986.
7. J. Pustejovsky, "Type coercion and lexical selection," in *Semantics and the Lexicon*, J. Pustejovsky, Ed. Dordrecht, The Netherlands: Kluwer, 1993.
8. A. Lally, J. M. Prager, M. C. McCord, B. K. Boguraev, S. Patwardhan, J. Fan, P. Fodor, and J. Chu-Carroll, "Question analysis: How Watson reads a clue," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 2, pp. 2:1–2:14, May/Jul. 2012.
9. D. C. Gondek, A. Lally, A. Kalyanpur, J. W. Murdock, P. Duboue, L. Zhang, Y. Pan, Z. M. Qiu, and C. Welty, "A framework for merging and ranking of answers in DeepQA," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 14, pp. 14:1–14:12, May/Jul. 2012.
10. D. A. Ferrucci, "Introduction to 'This is Watson'," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 1, pp. 1:1–1:15, May/Jul. 2012.
11. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, "DBpedia—A crystallization point for the web of data," *J. Web Semantics Sci., Services Agents World Wide Web*, vol. 7, no. 3, pp. 154–165, Sep. 2009.
12. M. C. McCord, J. W. Murdock, and B. K. Boguraev, "Deep parsing in Watson," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 3, pp. 3:1–3:15, May/Jul. 2012.
13. C. Wang, A. Kalyanpur, J. Fan, B. K. Boguraev, and D. C. Gondek, "Relation extraction and scoring in DeepQA," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 9, pp. 9:1–9:12, May/Jul. 2012.
14. G. A. Miller, "WordNet: A lexical database for English," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995.
15. F. M. Suchanek, G. Kasneci, and G. Weikum, "YAGO: A core of semantic knowledge-unifying WordNet and Wikipedia," in *Proc. 16th Int. WWW Conf.*, Banff, Canada, 2007.
16. J. M. Prager, E. W. Brown, A. Coden, and R. Radev, "Question-answering by predictive annotation," in *Proc. SIGIR*, Athens, Greece, 2000, pp. 184–191.
17. J. W. Murdock, J. Fan, A. Lally, H. Shima, and B. K. Boguraev, "Textual evidence gathering and analysis," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 8, pp. 8:1–8:14, May/Jul. 2012.
18. J. Fan, A. Kalyanpur, D. C. Gondek, and D. A. Ferrucci, "Automatic knowledge extraction from documents," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 5, pp. 5:1–5:10, May/Jul. 2012.
19. J. L. Fleiss, *Statistical Methods for Rates and Proportions*, 2nd ed. New York: Wiley, 1981.
20. M. D. Smucker, J. Allan, and B. Carterette, "A comparison of statistical significance tests for information retrieval evaluation," in *Proc. 16th ACM CIKM*, 2007, pp. 623–632.
21. A. Kalyanpur, B. K. Boguraev, S. Patwardhan, J. W. Murdock, A. Lally, C. Welty, J. M. Prager, B. Coppola, A. Fokoue-Nkoutche, L. Zhang, Y. Pan, and Z. M. Qiu, "Structured data and inference in DeepQA," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 10, pp. 10:1–10:14, May/Jul. 2012.
22. S. Schlobach, D. Ahn, M. de Rijke, and V. Jijkoun, "Data-driven type checking in open domain question answering," *J. Appl. Logic*, vol. 5, no. 1, pp. 121–143, Mar. 2007.
23. A. Grappy and B. Grau, "Answer type validation in question answering systems," in *Proc. RIAO—Adaptivity, Personalization and Fusion of Heterogeneous Information*, Paris, France, 2010, pp. 9–15.
24. D. Buscaldi and P. Rosso, "Mining knowledge from Wikipedia for the question answering task," in *Proc. Int. Conf. Lang. Resour. Eval.*, 2006, pp. 727–730.
25. R. Amsler, "The structure of the Merriam-Webster Pocket Dictionary," Ph.D. Dissertation, Univ. Texas, Austin, TX, 1980.
26. M. Chodorow, R. Byrd, and G. Heidorn, "Extracting semantic hierarchies from a large on-line dictionary," in *Proc. 23rd Annu. Meet. Assoc. Comput. Linguistics*, 1985, pp. 299–304.
27. M. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *Proc. 14th COLING Conf.*, 1992, vol. 2, pp. 539–545.
28. S. Soderland, A. Ritter, and O. Etzioni, "What is this, anyway: Automatic hypernym discovery," in *Proc. AAAI Spring Symp. Learn. Reading Learn. Read*, 2009, pp. 1–6.
29. A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, Jr., and T. M. Mitchell, "Toward an architecture for never-ending language learning," in *Proc. 24th AAAI Conf.*, 2010, pp. 1306–1313.

Received August 18, 2011; accepted for publication November 18, 2011

J. William Murdock *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (murdockj@us.ibm.com)*. Dr. Murdock is a member of the IBM DeepQA Research Team in the T. J. Watson Research Center. In 2001, he received a Ph.D. degree in computer science from Georgia Tech, where he was a member of Ashok Goel's Design and Intelligence Laboratory. He worked as a postdoctorate with David Aha at the U.S. Naval Research Laboratory. His research interests include natural-language semantics, analogical reasoning, knowledge-based planning, machine learning, and self-aware artificial intelligence.

Aditya Kalyanpur *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (adityakal@us.ibm.com)*. Dr. Kalyanpur is a Research Staff Member at the IBM T. J. Watson Research Center. He received his Ph.D. degree in computer science from the University of Maryland in 2006. His research interests include knowledge representation and reasoning, natural-language processing, statistical data mining, and machine learning. He joined IBM in 2006 and worked on the Scalable Highly Expressive Reasoner (SHER) project that scales ontology reasoning to very large and expressive knowledge bases. Subsequently, he joined the algorithms team on the DeepQA project and helped design the Watson question-answering system. Dr. Kalyanpur has over 25 publications in leading artificial intelligence journals and conferences and several patents related to SHER and DeepQA. He has also chaired international workshops and served on W3C Working Groups.

Chris Welty *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (cawelty@gmail.com).* Dr. Welty is a Research Staff Member in the Semantic Analysis and Integration Department at the T. J. Watson Research Center. He received a Ph.D. degree in computer science from Rensselaer Polytechnic Institute in 1995. He joined IBM in 2002, after spending 6 years as a professor at Vassar College, and has worked and published extensively in the areas of ontology, natural-language processing, and the Semantic Web. In 2011, he served as program chair for the International Semantic Web Conference, and he is on the editorial boards of the *Journal of Web Semantics*, the *Journal of Applied Ontology*, and *AI Magazine*.

James Fan *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (fanj@us.ibm.com).* Dr. Fan is a Research Staff Member in the Semantic Analysis and Integration Department at the T. J. Watson Research Center, Yorktown Heights, New York. He joined IBM after receiving his Ph.D. degree at the University of Texas at Austin in 2006. He is a member of the DeepQA Team that developed the Watson question-answering system, which defeated the two best human players on the quiz show *Jeopardy!*. Dr. Fan is author or coauthor of dozens of technical papers on subjects of knowledge representation, reasoning, natural-language processing, and machine learning. He is a member of Association for Computational Linguistics.

David A. Ferrucci *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (ferrucci@us.ibm.com).* Dr. Ferrucci is an IBM Fellow and the Principal Investigator for the DeepQA Watson/Jeopardy! project. He has been at the T. J. Watson Research Center since 1995, where he leads the Semantic Analysis and Integration department. Dr. Ferrucci focuses on technologies for automatically discovering meaning in natural-language content and using it to enable better human decision making. He graduated from Manhattan College with a B.S. degree in biology and from Rensselaer Polytechnic Institute in 1994 with a Ph.D. degree in computer science specializing in knowledge representation and reasoning. He has published papers in the areas of artificial intelligence, knowledge representation and reasoning, natural-language processing, and automatic question answering.

David C. Gondek *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (dgondek@us.ibm.com).* Dr. Gondek is a Research Staff Member and Manager at the T. J. Watson Research Center. He received a B.A. degree in mathematics and computer science from Dartmouth College in 1998 and a Ph.D. degree in computer science from Brown University in 2005. He subsequently joined IBM, where he worked on the IBM Watson Jeopardy! challenge and now leads the Knowledge Capture and Learning Group in the Semantic Analysis and Integration Department.

Lei Zhang *IBM Research Division, China Research Lab, Haidian District, Beijing, 100193, China (tozhanglei@qq.com)*. Dr. Zhang received his Ph.D. degree in computer science and engineering from Shanghai Jiao Tong University in China in 2005. His research interests include knowledge representation, Semantic Web, information retrieval, and statistical machine learning. After his Ph.D. study, he worked as a Research Staff Member in the Information and Knowledge Department of IBM Research–China. Since 2005, he and his team have worked on Semantic Web technologies and semantic search and their applications in the healthcare domain. Since 2008, he and his team have worked on using structured knowledge (including Semantic Web data) to help question-answering in the DeepQA project. He is active in several academic communities and is one of the major initiators of the China Semantic Web Symposium series, which started in 2007. He has been program committee member of conferences such as WWW, IJCAI (International Joint Conferences on Artificial Intelligence), ISWC (International Semantic Web Conference), etc. More recently, he was one of the local organizers of the ISWC 2010 conference.

Hiroshi Kanayama *IBM Research Division–Tokyo, Yamato-shi, Kanagawa 2428502, Japan (hkana@jp.ibm.com).* Mr. Kanayama is a Staff Researcher in IBM Research–Tokyo. In 2000, he received a master's degree from the Graduate School of the University of Tokyo, for research on Japanese syntactic analysis. Since joining IBM Research, his research has focused on several types of deep language analysis including machine translation, sentiment analysis, and knowledge extraction from unstructured data.