Relation extraction and scoring in DeepQA

C. Wang
A. Kalyanpur
J. Fan
B. K. Boguraev
D. C. Gondek

Detecting semantic relations in text is an active problem area in natural-language processing and information retrieval. For question answering, there are many advantages of detecting relations in the question text because it allows background relational knowledge to be used to generate potential answers or find additional evidence to score supporting passages. This paper presents two approaches to broad-domain relation extraction and scoring in the DeepQA question-answering framework, i.e., one based on manual pattern specification and the other relying on statistical methods for pattern elicitation, which uses a novel transfer learning technique, i.e., relation topics. These two approaches are complementary; the rule-based approach is more precise and is used by several DeepQA components, but it requires manual effort, which allows for coverage on only a small targeted set of relations (approximately 30). Statistical approaches, on the other hand, automatically learn how to extract semantic relations from the training data and can be applied to detect a large amount of relations (approximately 7,000). Although the precision of the statistical relation detectors is not as high as that of the rule-based approach, their overall impact on the system through passage scoring is statistically significant because of their broad coverage of knowledge.

Introduction

The IBM DeepQA question-answering framework allows for a wide variety of candidate answer generators and scorers that enable it to generate candidate answers and compute a match (or alignment) between the question text and the supporting textual evidence found for a given candidate [1–3]. The text analytics used in these modules rely primarily on surface-level information or the predicate-argument structure (PAS) [4] of the text fragments under consideration. In this paper, we focus on analytics that go beyond explicit lexical and syntactic information and instead detect implicit semantic relations in the text.

Table 1 illustrates instances of such implicit semantic relations in Jeopardy!** data. For instance, the question "'The Screwtape Letters' from a senior devil to an under devil are by this man better known for children's books" contains an instance of the `author` relation, whose arguments are identified as "this man" and "The Screwtape Letters". Detecting such implicit relations in the question text is very useful for question answering because it enables background knowledge base to be used to find potential answers to the question (for the examples in Table 1, the relations are from the DBpedia knowledge base [5]; thus, we can look up potential answers in this resource based on the relation detected).

Another application of semantic relation detection is for passage scoring. Consider the question

> "This hockey defenseman ended his career on June 5, 2008"

and a supporting passage

> "On June 5, 2008, Wesley announced his retirement after his 20th NHL season."

The question and the passage have very few keywords in common (especially verbs and nouns), and thus, any similarity between the two computed with explicit term matching-based approaches [2] would be low. However, if a relation detector can find that the question and the

Digital Object Identifier: 10.1147/JRD.2012.2187239

Table 1 Examples of Jeopardy! questions and detected relations.

Jeopardy! question	*Relation detected (relations are from the DBpedia knowledge base)*
MOTHERS & SONS: Though only separated by one year in real life, she played mother to son Colin Farrell in "Alexander."	Starring (she, "Alexander")
THE DEVIL: "The Screwtape Letters" from a senior devil to an under devil are by this man better known for children's books.	Author (man, "The Screwtape Letters")
THE LONE REPRESENTATIVE: Michael Castle from this state with 3 counties: New Castle, Kent and Sussex.	Residence ("Michael Castle", state)

passage share a common semantic relation, e.g., `ActiveYearsEndDate` (also defined in the DBpedia knowledge base), an appropriate scoring algorithm can arrive at a higher semantic similarity score.

In this paper, we first present a high-precision approach for detecting semantic relations based on a handcrafted set of patterns over the PAS representation of text. We then present an approach based on statistical pattern discovery and aggregation, i.e., Topicalized Wide Relation and Entity eXtraction (TWREX), which achieves much higher recall relation detection by training support vector machines (SVMs) [6] on a large number of relations defined in DBpedia using training data from Wikipedia** [7].

Our relation extraction and passage-scoring components make use of several other DeepQA components, including the English Slot Grammar (ESG) parser [8], entity disambiguation and matching (EDM) [9], predicate disambiguation and matching (PDM) [9], and Answer Lookup [2]. We evaluate our relation detection at two levels. At the component level, we measure how well our approach detects semantic relations in Wikipedia data, ACE data [10], and Jeopardy! questions. At the system level, we evaluate the overall end-to-end impact of relation detection on the question-answering performance of IBM Watson*.

This paper begins with a description of our rule-based relation detection technique. It then described our statistical relation detection and scoring algorithms. The next section describes how this paper fits in the DeepQA architecture. The last two sections summarize the experimental results and provide some concluding remarks.

Rule-based relation extraction

A relation can be often expressed in multiple ways lexically and syntactically. For instance, an `authorOf` relation might be expressed not only with nouns such as "author" but also with verbs such as "write", "compose", "pen", or "publish"; furthermore, these might appear in different syntactic configurations (such as passive versus active, e.g., "Originally *written* by Alexander Pushkin as a poem"; nominalized or verbal forms of the same underlying predicate, e.g., "This 'French Connection' actor *coauthored* the 1999 novel 'Wake of the Perdido Star'"; or exploiting the semantic load of complement slots to relational nouns, e.g., "*playwright* of 'The Maids'", "'X-Files' *star*").

There is effectively an unlimited number of ways to realize a relation instance in text. In many cases, however, basic principles of economy of expression and/or conventions of genre will ensure that certain systematic ways of expressing a relation are repeatedly used [11] to the extent that patterns can be observed and categorized. The intuition behind rule-based relation detection derives from such observations, i.e., that a rule can be stated that will detect multiple instances of a pattern.

Lexical analysis of historical Jeopardy! questions gives us lists of verbs and nouns in the arts-and-entertainment (A&E) domain that are strongly indicative of deep semantic relations such as "write", "compose", "star", "play", "biography", and so forth; these are complemented by the semantic type assignment to named entities such as "David Lean" (a Director) and "Autumn Sonata" (a Film).

The choice of relations for which we have hand-created recognizers is not arbitrary. Words such as "film", "author", and "novel" are prominently at the head of the distribution of lexical answer types (the kinds of entities that questions are asking about; see [9]); furthermore, in well over 50% of questions asking for them, there is an instance of a semantic relation over what the question is asking about, which is potentially useful for structured lookup or passage scoring.

As simple examples of a relation, consider locutions such as "a Mary Shelley tale", "the Saroyan novel", "Twain's travel books", and "a 1984 Tom Clancy thriller". The key observation is that a simple pattern is at play here, interacting with a system of semantic types ([Author] and [Prose], in this case; the fact that "thriller" is ambiguous between [Prose] and [Film] should not detract from the argument), and is dependent on lexical and syntactic context. The value of such a pattern in an `authorOf` relation detector would be determined by our ability to impose semantic constraints upon the elements of candidate phrases and on the frequency of occurrence of such locutions in the domain data. Typically, relation detection for any relations would be sensitive to a number of patterns, each one, such as the one illustrated here, fairly productive in itself.

Pattern elements are typically syntactic constituents, and a quality parser can reliably identify such constituents, as well as dependency links among them; an ontology of types can be used for semantic constraint checking. In our case, a deep syntactic parser (i.e., ESG), a mapper from dependency trees to PAS, and a comprehensive semantic type system are essential enabling technologies, providing the scaffolding for effective pattern writing [4]. Still, identifying productive patterns and then developing the capability to normalize the different inputs to a canonical representation is expensive, and we have done this for relations that are (A) particularly prominent in the domain or (B) are broadly manifested across all question data.

In Jeopardy!, (A) holds for questions referring to the A&E domain, broadly construed to capture relationships between animate agents and works of art (e.g., actors in films or plays; authors of prose, poetry, or music; and characters in fictional works). Examples of (B) concern the more generic properties of many Jeopardy! lexical answer types (properties such as place or date of birth, nationality, alternative naming conventions, time stamping and temporal linking, and geospatial constraints). The following examples illustrate instances of relations for which we have manually developed rule sets.

1. Robert Redford and this "Picket Fences" star both debuted as soldiers in the 1962 drama "War Hunt." `(actorIn, actorOf, timeStamp)`
2. Born in Winsted, he practiced law in Connecticut before he wrote "Unsafe at Any Speed." `(bornWhere, authorOf)`
3. This Norwegian star of such movies as "Autumn Sonata" was actually born in Japan. `(nationalityOf, actorIn, bornWhere)`
4. The main library at the University of Northern Colorado is named for this alumnus who wrote an epic of Colorado in 1974. `(namedAfter, authorOf)`

Example 1 contains three `actorIn` and two `actorOf` relations (binding actors and films, and actors and characters, respectively), with the focus (the part of the question that is a reference to the answer; "this 'Picket Fences' star" in the example) participating in two of them. Example 2 binds the focus as argument to a `bornIn` and an `authorOf` relation: `bornIn(focus:he,Winsted)`, `authorOf(focus:he,Unsafe At Any Speed)`. Example 3 highlights three properties of the focus, namely, `actorIn`, `nationalityOf`, and `bornWhere`. Example 4 illustrates a name-sharing association between the focus and another named entity.

Relation detection patterns are written in Prolog and apply unification pattern matching over PAS [4, 12]. Unification can be traced to resolution-based theorem proving [13] and has been shown to be a particularly convenient and effective way to manipulate directed acyclic graphs. As such, it was adopted in the natural-language processing (NLP) community as the underlying representational and interpretational device for formal grammar specification and feature structure manipulation [14–16]. The Prolog language is, in effect, an efficient unification interpreter [17].

We sought parsimony in rule specification by strongly typing the relations and exploiting the lexical space of relation expression, as well as similarities in the syntax of expressing different relations (where appropriate). For instance, in the A&E domain, patterns such as

```
authorOf -> np : [Author] -> nadj
          -> [Prose]. // "thisWordsworthpoem"
```

have parallel versions for relations such as `directorOf` and `composerOf`

```
directorOf -> np : [Director] -> nadj
  -> [Film]. // "aDavidLeanclassic"
composerOf -> np : [Composer] -> nadj
  -> [Music]. // "thisPucciniopera"
```

Consequently, a metapattern for `creatorOf` can be developed and, as appropriate, specialized, depending on fine-grained semantic typing of the pattern elements.

In DeepQA, the rule-based relation detection module can detect approximately 30 relations. On average, each relation is associated with 10–20 rules.

Statistical approaches for relation extraction and passage scoring

The open-domain setting of DeepQA means that a large set of relation types must be covered in order to have a substantial impact on overall system performance. Targeting just a handful of relations would affect only a small subset of the data and is unlikely to have a significant impact. In this section, we present our statistical approach for relation detection, which is able to detect more than 7,000 relations spanning a diverse set of topics.

The organization of this section is as follows: We first describe the internal representation of relation instances, and on the basis of that, we outline the approach to automatically extract training data from Wikipedia and DBpedia. Next, we present "relation topics", i.e., the most novel part of our approach. The last two subsections discuss how relation topics are used in relation extraction and unstructured passage scoring. **Figure 1** shows a diagram of our relation topic-based relation detection component.

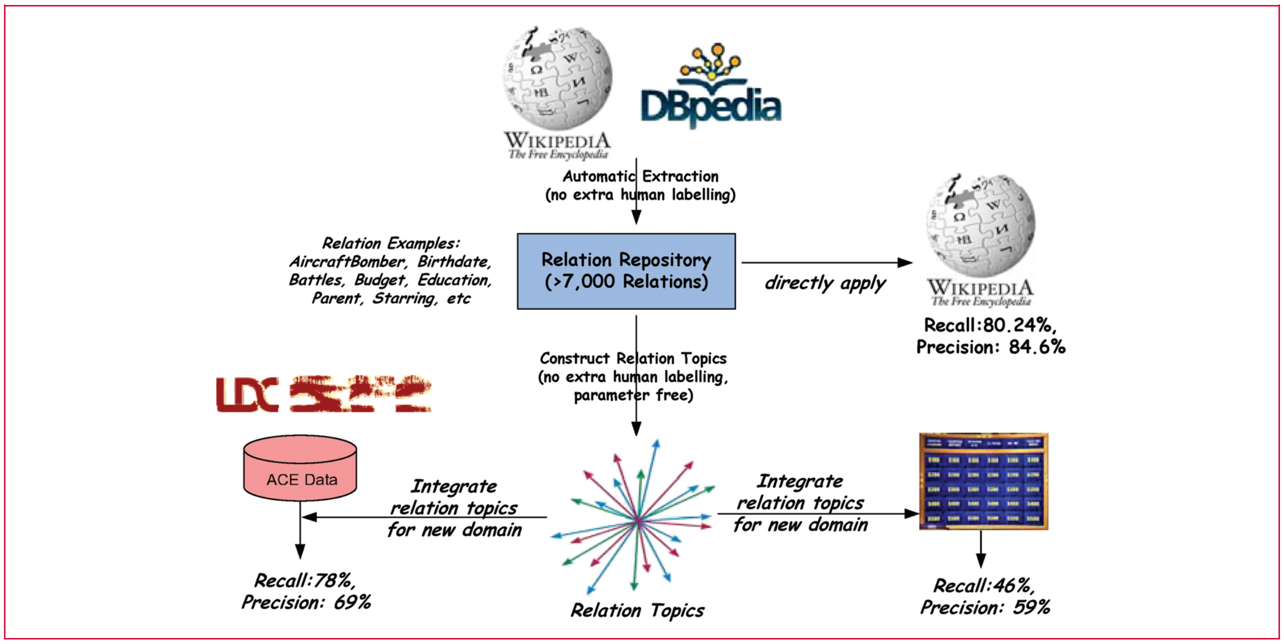

Figure 1

Statistical relation detection module.

Internal representation of relation instances

In our statistical approach, each sentence expressing a relation is represented by a set of features, including: 1) types of relation arguments; 2) syntactic features such as *subject* and words in the dependency path between two arguments; and 3) words in the whole sentence. For example, a sentence expressing `ActiveYearEndDate` relation,

"Sauve announced his retirement from the NHL in 1989",

is represented by the following features: argument$_1$type [Person, Athlete]; argument$_2$type [Year]; syntactic features in the dependency path [subj, mod_vprep, objprep]; words in the dependency path [*announce* (verb), *in* (prep)]; and words in the whole sentence [*Sauve* (noun), *announce* (verb), *his* (adj), *retirement* (noun), *from* (prep), *NHL* (noun), *in* (prep), *1989* (noun)].

In this representation, the two arguments are ordered, with the subject coming first and the object second. If the order is not given, we consider both combinations.

The YAGO (Yet Another Great Ontology) type system [18] is used to assign types to the relation arguments. Given an arbitrary entity, we retrieve its YAGO types using the EDM [9] or PDM components [9]. EDM maps the textual mention of an entity to an entity resource in a structured knowledge base. Given the uncertainty associated with the mapping process, it typically involves mapping to a collection of entities, each with an associated confidence level, i.e., roughly a measure of the probability of the accuracy of the mapping. The knowledge base used in our EDM algorithm is Wikipedia. The Wikipedia page URLs are transformed into the corresponding DBpedia URIs, which then can be mapped to YAGO types. PDM is analogous to EDM. It is used to map the lexical answer type of an entity to YAGO types.

The Slot Grammar parser ESG [8] is used to parse each sentence into a dependency tree that shows both surface structure and deep logical structure. Each tree node has a word-sense predicate with its logical arguments, a list of morphosyntactic features, and the left and right modifiers of the node. On the basis of the ESG parse, we extract a dependency path between two arguments. A dependency tree example is shown in **Figure 2**. Note that there could be multiple paths between two arguments in the tree. We only take the shortest path into consideration. Seventy-six syntactic features such as *subject* and *object* are extracted from the ESG parse for the given sentence and are used in our feature set.

Both the dependency path and the sentence features are filtered for five parts of speech, i.e., adjective, adverb, noun, preposition, and verb (thus, there is no determiner in the features for the above example).

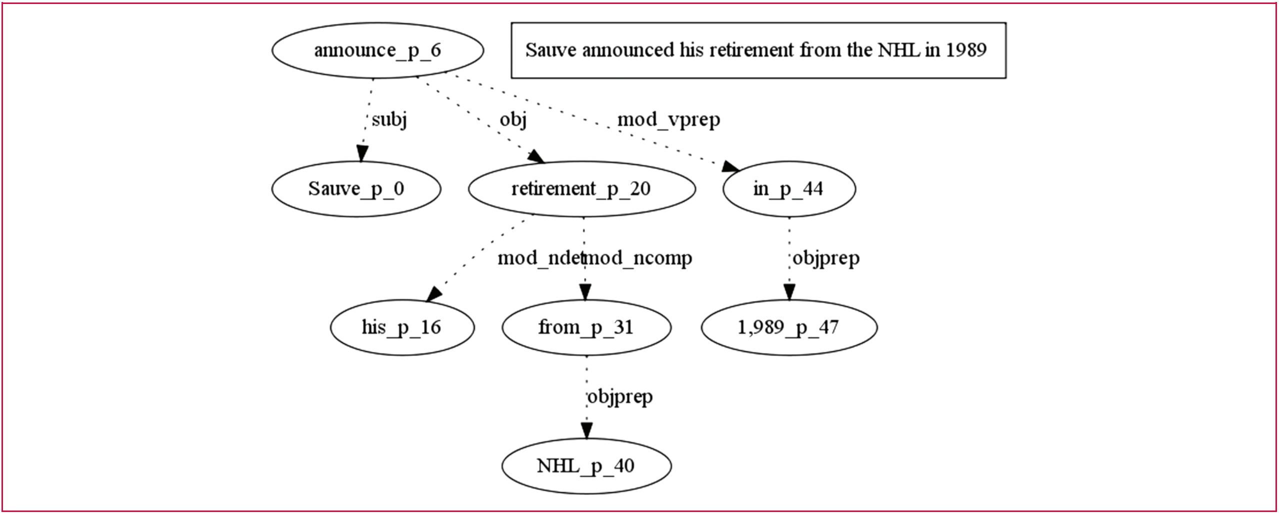

Figure 2

Example of a dependency tree.

Extracting training data from Wikipedia and DBpedia

Our training data is composed of two parts, i.e., relation instances from DBpedia and sentences describing the relations from the corresponding Wikipedia pages.

Collecting the training data

Since our relations correspond to Wikipedia infobox properties, we use an approach similar to that described in [19] to collect positive training data instances. We assume that a Wikipedia page containing a particular infobox property is likely to express the same relation in the text of the page. We further assume that the relation is most likely expressed in the first sentence on the page that mentions the arguments of the relation. For example, the Wikipedia page for "Albert Einstein" contains an infobox property "alma mater" with value "University of Zurich", and the first sentence mentioning the arguments is "Einstein was awarded a PhD by the University of Zurich", which expresses the relation. When looking for relation arguments on the page, we go beyond (sub)string matching and use link information to match entities that may have different surface forms.

Using such techniques, we have collected 620,000 examples characterizing 7,628 DBpedia relations. Our heuristic approach returns reasonably good results but brings in about 20% noise (estimated by manual inspection) in the form of false positives. For example, if someone was born and died in the same place, the sentence we extract for the `birthPlace` relation might actually express the `deathPlace` relation instead. This is a concern when building an accurate statistical relation detector. To address this issue, we have developed a keyword filter and an argument type filter to automatically remove some of the noisy data.

Retrieving types for the arguments

To get precise type information for the arguments of a DBpedia relation, we use the DBpedia knowledge base [5] and the associated YAGO type system [18]. Note that, for almost every Wikipedia page, there is a corresponding DBpedia entry that has captured the infobox properties as resource description framework (RDF) triples. Some of the triples include type information, where the subject of the triple is a Wikipedia entity and the object is a YAGO type for the entity. For example, the DBpedia entry for the entity "Albert Einstein" includes types corresponding to Scientist, Philosopher, and Violinist, etc. These YAGO types are also linked to appropriate WordNet** concepts [20], providing accurate sense disambiguation. Thus, for any entity argument of a relation we are learning, we obtain sense-disambiguated type information, which is used as a feature in the relation detection model.

Relation topics

Similar to the topics defined over words [21, 22], we define *relation topics* as multinomial distributions over the existing DBpedia relations. One relation topic example is as follows: `[ doctoraladvisor (0.683366), doctoralstudents (0.113201), candidate (0.014662), academicadvisors (0.008623), notablestudents (0.003829), college (0.003021), operatingsystem (0.002964),`

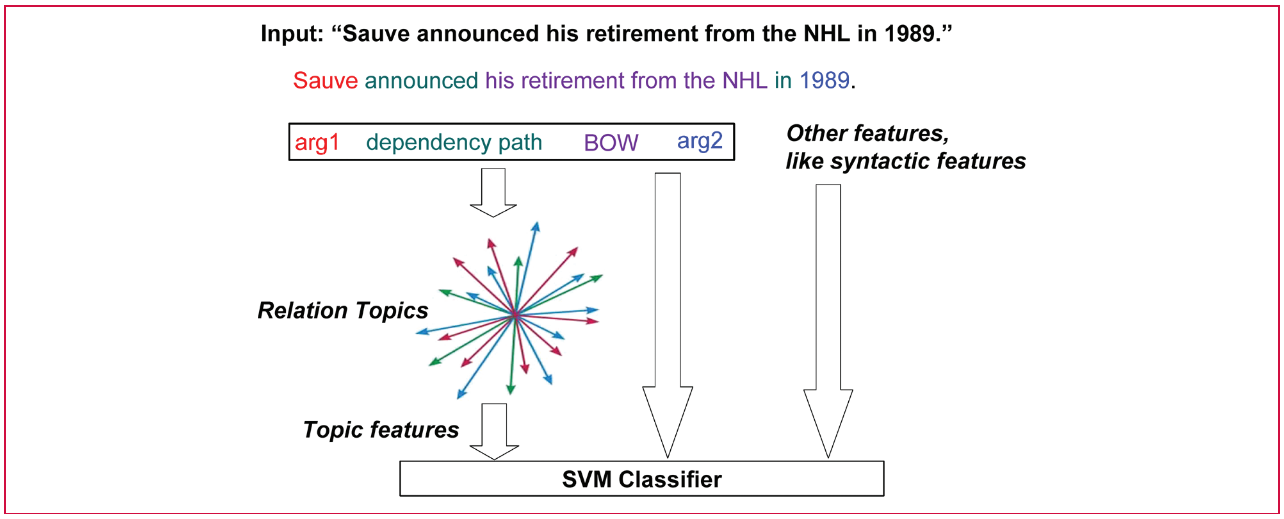
Input: "Sauve announced his retirement from the NHL in 1989."
Sauve announced his retirement from the NHL in 1989.
arg1 dependency path BOW arg2
Other features, like syntactic features
Relation Topics
Topic features
SVM Classifier

Figure 3

Features used in statistical relation extraction.

`combatant (0.002826), influences (0.002285), training (0.002148),...`], where `doctoraladvisor` is a DBpedia relation and `0.683366` is its contribution to this relation topic. The length of this topic vector is 7,628, which is the total number of DBpedia relations.

All such relation topics are automatically constructed using an unsupervised analysis of the correlations between existing relations regarding one or a set of components (argument$_1$, noun in the dependency path, etc.). Relation topics are defined at multiple scales, are human interpretable, are orthonormal to each other, and can be used as basis functions to re-represent the questions and passages. We project each relation instance onto this relation topic space, resulting in a set of features to represent the instances in the topic space. The methodology to construct relation topics, and features, is presented in [23].

Some argument types and words occur in multiple relations. They are used to model the correlations between different relations. The relation topic construction process extracts the topics from the finest level of the input data while at the same time modifying the relationship between relations to focus more on low-frequency indirect co-occurrences (between relations) for the next level.

The main motivation for the creation of relation topics is that we can project any relation instance candidate (including the candidates from unseen relations) onto the relation topic space. This step results in a set of relation topic features that reflects the relationship between the given candidate and the known relations. Such topic features are used in an SVM classifier (through the kernel function, see below) to detect relations from the relation instance candidates or compute the similarity score between the question and the passage.

Relation detection with relation topics

In this subsection, we describe our technique to do relation detection. Our model (or classifier) that does relation detection is trained using SVMs. SVM classifiers use a kernel function to model similarity among the instances. Our kernel function combines the relation topic features with three other existing kernels that match arguments, dependency paths, and the common words shared by two instances (see **Figure 3**). Details about this kernel are presented in [23].

Training

In training data from DBpedia and Wikipedia, the arguments of each relation instance are already given and the order is always from *subject* to *object*. We train one SVM model with our kernel for each relation. The negative training set for each relation is created by sampling the instances from the other relations.

Testing

Given a test instance, the testing procedure has the following four steps.

1. *Detect argument pairs*—For Jeopardy! questions, the focus and each named entity in the question make an argument pair. For an arbitrary sentence, we use ESG to extract all predicate nodes, and any predicate node pair makes a pair of relation arguments.
2. *Detect argument order*—In some situations, the dependency path tells us which argument is *subject* and

which is *object*. Otherwise, we try both combinations.

3. *Filtering*—To mitigate the noise introduced by false positives in our training data extraction heuristics and to provide a way to balance the recall and precision, we have also developed a keyword filter and an argument type filter to filter out noisy relation instances.

   On a relation-per-relation basis, we extract the most popular verbs, nouns, and prepositions (these are deemed more important than the other words to detect relations) from the dependency path of each training positive instance. For any given relation, we add the words that occur in at least 10% of positive instances or at least three times in its positive training data set to a list to create a *keyword filter*. If a given test instance does not have any key word in this list, it will be filtered out. We also designed an *argument type filter* exploiting the YAGO type hierarchy, which extracts the most common parent type in YAGO for each argument in each relation. The filter then rejects test instances that have at least one argument not under the corresponding common parent type.

   We drop the test instance if it does not pass either the keyword filter or the argument type filter (this step is optional).

4. *Apply*—Apply all our relation detectors to the test instance to extract semantic relations.

Unstructured passage scoring with relation topics

When the question and a passage share very few keywords, any similarity between them computed using term-matching-based approaches would be low. In such situations, if a relation detector can find shared relations between the question and the passage, this can lead to a higher semantic similarity score.

One way to integrate this idea in the DeepQA framework is to use our relation detectors to find all relations expressed in both the question and each given passage. The number of relations they share is then used to score the passage. This approach, however, does not work well in practice for two reasons. First, the open-domain setting of DeepQA requires that a large set of relation types be covered. In our case, we have more than 7,000 relation models that need to be applied to each given passage. Considering the huge amount of passages to be processed, this approach is not realistic in real-world applications. Second, many semantic relations are not covered in DBpedia. If a question and a passage share some significant relations that are not in DBpedia, the passage will not be appropriately scored. To address these challenges, we developed a novel algorithm for passage scoring in DeepQA by integrating relation topics extracted from Wikipedia and DBpedia data.

The information extracted from relation topics is useful for unstructured passage scoring. For example, if argument types *town, city, state*, etc. often occur in the same relations (such as `birthPlace` and `deathPlace`), the topic extraction approach will automatically learn a topic from that. When we align argument types *town* and *city* in the relation topic space, we will get a high similarity score since the projection results of them on the topic space are close to each other. Words with similar meanings will be also treated in a similar way. In the relation topic space, instances from related relations will be grouped together. Passages that only have relations remotely matching the relations expressed in the question will be still appropriately scored since the indirect relationship between relations is also captured in the topic space. Using relation topics may significantly help score passages, and it potentially expands the information brought in by each passage from making use of the knowledge extracted from the existing relation repository.

In DeepQA, the passage score represents how well a given passage matches the question and is used to rank candidate answers derived from the passages. For the sake of simplicity, we assume that the question has one focus and the passage has one candidate answer. In the following question–passage example, the question focus is *state* and the candidate is *state of Quintana Roo*.

> Question: It's Mexico's northernmost state, but part of its name means "low".

> Passage: Cabo Catoche or Cape Catoche, in the Mexican state of Quintana Roo, is the northernmost point on the Yucatan Peninsula. It is about 53 km (33 miles) north of the city of Cancun.

To align the question and the passage, we first find pairs of matching terms, which are the terms that resemble each other semantically. In this example, we have two pairs of matching terms [*Mexico's*, *Mexican*, via stem: *Mexico*] and [*northernmost*, *northernmost*]. We assume that the focus and one matching term in the question make a semantic relation. We then assume that the candidate variant and the corresponding matching term in the passage also make a relation. We project these two relation instances onto the relation topic space. The cosine similarity of the projection results is used to score the passage for the selected matching terms. In this example, we have two matching pairs. The sum of two matching scores is used as the final passage score. Since the matching terms other than the focus–candidate pair are always quite similar to each other ([*northernmost*, *northernmost*], in the example above), we do not take them into consideration when we compute cosine similarity. The passage score is used as a feature and is used for candidate answer ranking [24].

Integrating in DeepQA

The relation detection component is used in several places in the DeepQA pipeline.

Table 2 Component-level evaluation of different approaches using ACE 2004 data.

Approach	*Recall*	*Precision*	*F_1 score*
Convolution tree kernel	56.7%	72.5%	0.636
Composite kernel (linear)	67.00%	73.5%	0.701
Syntactic kernel	70.5%	69.23%	0.6986
Nguyen et al. (2009) [30]	67.00%	76.60%	0.7150
Our kernel with topic features	77.88%	69.15%	0.7324

In the question analysis step [12], we attempt to identify relations between the focus and each named entity in the question. If a relation is detected, we invoke the Answer Lookup component [2] to search against structured knowledge base (such as DBpedia) for all instances of the relation that contain the given *named entity* as one argument. The other argument of the relation is then treated as a potential candidate answer.

Relation extraction, i.e., both on the question side and on the content side, is used in supporting passage scoring, in particular, by components such as Logical Form Answer Candidate Scorer (LFACS) [3] and by the unstructured passage-scoring component described in the previous section.

Separately, semantic relations produced by the rule-based approach are also used by some other components in DeepQA. One use is in keyword search, where keywords connected to the focus by a semantic relation are weighed up in the search query [2]. Geospatial and temporal semantic relations are used to query structured knowledge bases for structured inference-based constraint satisfaction and scoring of candidate answers [25]. Additionally, the results of relations extraction over a large body of text are aggregated in the PRISMATIC knowledge base [26], which is itself used by a range of search and answer-scoring components. Semantic relations have been also enhanced to facilitate processes of semantic frame detection and frame slot instantiation [25].

Experiments

We evaluate our approaches at both the component and system levels.

Component-level evaluation

At the component level, we performed six experiments to measure the relation extraction performance on Wikipedia data, ACE data, and Jeopardy! questions.

a) *Wikipedia data*

Experiment 1—In the first experiment, we evaluate the statistical relation detection approach trained on Wikipedia data (no filter) using a Wikipedia data set held out for testing. We achieve an 81.18% F_1 score in this experiment. Details about this experimental setup are in [23].

b) *ACE data*

Experiment 2—The ACE 2004 corpus is the most popularly used benchmark data set for relation extraction. In the second experiment, we use ACE data to compare our approach against the state-of-the-art approaches on relation extraction, including convolution tree kernel [27], syntactic kernel [28], composite kernel (linear) [29], and the best kernel in [30]. The results are summarized in **Table 2**. Our statistical approach has the best performance, achieving a 73.24% F_1 score. Details about the experimental setup are in [23]. The rule-based approach is not tested in this experiment since it is not developed to process ACE relations.

c) *Jeopardy! data*

As noted earlier, our statistical relation detectors are trained on Wikipedia data. We now run four additional experiments (Experiments 3 through 6, listed below) to test the relation detection on 3,508 Jeopardy! questions. We use recall and precision to measure the performance. Measuring recall is challenging since no ground truth information about what relations are expressed in each question is given.

To create a data set to measure recall, we took the following approach. From the question text of each question, we retrieve all its named entities. Each named entity and the answer (used to replace the focus) to the question potentially participate in some semantic relations. We then check whether the answer is in the infobox of the Wikipedia page corresponding to each named entity in the question. If the answer is there, then we have a relation instance candidate. All such candidates are manually verified to remove the false positives. Using this approach, we collected 370 DBpedia relation instances. Precision is measured by manually checking whether the extracted relations are true. In this task, only the DBpedia relations that occur in the recall set are considered for precision computation. The results reported in this data set are summarized in **Table 3**.

Table 3 Component-level evaluation using Jeopardy! data.

	No. relation detectors	*Recall*	*Precision*	*F_1 score*
Rule-based approach[a]	~30 relations	41.40%	83.25%	0.5530[a]
Statistical approach (without filters)	>7,000 relations	54.28%	25% (estimated)	0.3423
Statistical approach (with filters)	>7,000 relations	45.71%	59.44%	0.5168
Filters only	>7,000 relations	83.00%	—	—

[a]The rule-based approach is evaluated on nine relations.

Experiment 3—In the third experiment, we evaluate our statistical approach trained on Wikipedia data (no filter) using Jeopardy! data. The recall for this test is 54.28%. The exact precision is not known since more than 7,000 relation instances are extracted and it is unrealistically time consuming to manually annotate all of them. We randomly chose 5% of the relation instances for annotation and find that the precision is around 25% for the sampled data.

Experiment 4—In the fourth experiment, we evaluate the statistical approach with both the keyword filter and the type filter active. The noise reduction due to filtering now makes it possible to manually annotate the results. The recall for this setting is 45.71%, precision is 59.44%, and the F_1 score is 51.68%. This shows that using filters can significantly improve precision without compromising recall.

Experiment 5—In order to assess the effects of filtering even further, we run a test to see how many positive instances are blocked by the filters; the result shows that 83% of positive instances passed the filters. The most popular failure case is due to arguments specific to Jeopardy! not being recognized by our filters. For example, the argument type filter does not know that "a hit" may represent a movie since the Wikipedia data does not have such examples.

Experiment 6—In the sixth experiment, we evaluate our *rule-based approach* using Jeopardy! data. Note that the rule-based approach is not statistical (and, hence, no training is required) since it works by applying pattern-based rules to text fragments. The relations identified by the rule-based approach do not necessarily have corresponding DBpedia relations. The recall and precision numbers in this experiment are reported only on nine relations that can be easily mapped to DBpedia relations. These nine relations are `actorIn`, `authorOf`, `bornWhen`, `bornWhere`, `composerOf`, `creatorOf`, `directorOf`, `nationalityOf`, and `performerOf`. The recall for these nine relations is 41.40%, precision is 83.25%, and the F_1 score is 55.30%. In comparison, the statistical approach with filters on the same nine relations achieved a recall of 41.27%, a precision value of 57.79%, and an F_1 score of 48.15%.

d) *Analysis*

The experiments on ACE and Wikipedia data show that our statistical relation detection approach has state-of-the-art performance at the component level. Since we do not have sufficient training data from Jeopardy!, the statistical relation detectors we use for Jeopardy! are also trained on Wikipedia.

Compared with the F_1 score from the Wikipedia test, there is a significant performance drop for the Jeopardy! task. This is due to several issues that cause a mismatch between the training data obtained from Wikipedia and the test data of Jeopardy!. First, the argument types in the Wikipedia data are given. In Jeopardy!, we have to use the EDM and PDM algorithms [9] to obtain argument types, and the accuracy of these two modules presently is approximately 70%. Second, most sentences in Wikipedia are instances of straightforward prose, whereas many Jeopardy! questions are vague and/or tricky or otherwise unusual. Third, some terms used in Jeopardy! are not used very often in Wikipedia, e.g., the term "hit" (as a "movie") discussed above. We find that using filters improves the precision and makes the relation detection models usable for Jeopardy! tasks.

Our rule-based approach is tested only on Jeopardy! data. Compared with the statistical approach, the rule-based approach has slightly worse recall but, as expected, much higher precision.

System-level evaluation

At the system level, we evaluate the impact of relation extraction on end-to-end question-answering performance through candidate answer generation and unstructured passage scoring. We apply our approaches in the context of two different configurations of the DeepQA system. The full configuration includes all components of DeepQA. The basic configuration includes all of the standard question analysis components, search, candidate generation,

Table 4 System-level impact in end–end experiments of relation detection-based candidate answer generation.

Configuration	*Accuracy improvement*
Basic configuration	54.36%
Basic + rule-based	54.79% (+0.43%)
Basic + rule-based + statistical relation detection	54.59% (+0.23%)
Full configuration	69.81%
Full + rule-based + statistical relation Detection	70.35% (+0.54%)

Table 5 System-level impact in end–end experiments of relation detection-based unstructured passage scoring.

Configuration	*Accuracy improvement*
Basic configuration	54.36%
Basic + rule-based	54.79% (+0.43%)
Basic + rule-based + statistical passage scoring	56.79% (+2.43%)
Full configuration	69.81%
Full + rule-based + statistical passage scoring	70.41% (+0.6%)

a simplified configuration for merging and ranking answers, and one answer scorer that checks answer types using a named entity detector [9]. All experiments are conducted on a previously unseen set of 3,508 Jeopardy! questions. Performances are reported on the accuracy improvement from our relation detection approaches in end-to-end experiments.

Candidate answer generation using relation extraction
In candidate answer generation, we extract relations between the *focus* and each *named entity* in the question. If a relation is detected, we invoke the Answer Lookup component to search against structured knowledge bases for all instances of the relation that contain the given named entity as one argument. The other argument of the relation is then treated as a potential candidate answer. We measure the impact of this component to the system. The results are summarized in **Table 4**. Under the basic configuration, statistical relation detection, together with the rule-based approach, achieves 0.2% accuracy improvement over the baseline system. Under the full configuration, the improvement is about 0.54%. Neither difference is statistically significant.

Unstructured passage scoring using relation extraction
In unstructured passage scoring, we use the passage score computed in the relation topic space as a new feature and apply it to candidate answer ranking. The impact of this score is reported in **Table 5**. Under the basic configuration, passage scoring using both relation topics and manual rules achieves 2.43% accuracy improvement over the baseline system. Statistical relation extraction alone contributed a 2% improvement for this case. Under the full system configuration, the improvement is approximately 0.6%. Both differences are statistically significant using McNemar's test with Yates Correction for continuity [31].

Analysis of results
The overall system-level impact of relation detection on candidate answer generation is not significant. There are several reasons for this. First, relation detectors only fire on 13.6% of the questions, and 82% of the questions in this subset have been already correctly answered without using relation detection capabilities. This means that our headroom for improvement is only approximately 90 questions. Although the relation detection module helps answer 16 more questions in this subset correctly, the overall impact is still not significant. Second, DBpedia has incomplete coverage. For example, some U.S. presidents are not labeled as U.S. citizens in DBpedia. Issues such as this lead to misleading information being supplied to the ranking model, which is consequently unable to generate some candidate answers even when the relation detector returns the correct results. Third, the overall performance of our relation detectors on Jeopardy! data is not satisfying because of the inherent mismatch between the Wikipedia-derived training data and test data. This is also confirmed in the component-level experiments.

On the other hand, the overall system-level impact of relation detection on unstructured passage scoring is statistically significant under both the basic and full configurations. In addition, we do not need to search against the knowledge base at run time since the passage score is computed in the relation topic space, which has already integrated the information extracted from the knowledge base. This significantly reduces the time and cost needed to detect relations in the DeepQA framework.

Conclusion
This paper has presented two approaches for relation extraction and unstructured passage scoring in DeepQA using handcrafted patterns and statistically derived relation topic models, respectively. The effectiveness of our approaches is demonstrated using multiple data sets at both the component and system levels.

At the component level, our relation detector outperforms the state-of-the-art approaches on ACE data, which is the most popularly used relation detection test data set.

At the system level, the candidate generator component based on relation detection does not have a significant impact on the system for three reasons: 1) Most questions that our relation detectors fire on have been already correctly answered, leaving us with little headroom to improve the system performance; 2) DBpedia has poor coverage on some relations that are of relevance; and 3) the training data from Wikipedia and the test data from Jeopardy! are quite different. More advanced domain adaptation approaches are still needed to make the performance on Jeopardy! data match the performance on Wikipedia data.

At the system level, our unstructured passage-scoring component based on relation detection has a statistically significant impact on the system, demonstrating that the information brought in by semantic relations is important to open-domain QA systems.

Comparing the two approaches in our framework, the rule-based approach has high precision and is used in many other components in the DeepQA framework. This needs to be offset against the human effort, domain knowledge, and experience needed to create rules for new relations. Statistical approaches learn how to extract semantic relations from the training data and can be applied to detect a large number of relations. It is interesting to note that, although the precision of the statistical relation detectors is not as high as that of the rule-based approach, their overall impact on the system through passage scoring is statistically significant because of broad coverage of knowledge. In the future, we will explore more advanced techniques to combine the two approaches in the DeepQA framework.

Acknowledgments

The authors would like to thank the Defense Advanced Research Projects Agency (DARPA) Machine Reading Program for its support under Air Force Research Laboratory (AFRL) prime contract no. FA8750-09-C-0172. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the view of the DARPA, AFRL, or the U.S. Government.

*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. 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.
2. 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.
3. 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.
4. 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.
5. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, “DBpedia: A nucleus for a web of open data,” in *Proc. 6th Int. Semantic Web Conf.*, Busan, Korea, 2007, pp. 11–15.
6. T. Joachims, *Making Large-Scale SVM Learning Practical*. Cambridge, MA: MIT Press, 1999.
7. Wikipedia. [Online]. Available: http://www.wikipedia.org/
8. M. McCord, “Slot grammar: A system for simpler construction of practical natural language grammars,” in *Proceedings of the International Symposium on Natural Language and Logic*, R. Studer, Ed. London, UK: Springer-Verlag, 1989, pp. 118–145.
9. J. W. Murdock, A. Kalyanpur, C. Welty, J. Fan, D. A. Ferrucci, D. C. Gondek, L. Zhang, and H. Kanayama, “Typing candidate answers using type coercion,” *IBM J. Res. Dev.*, vol. 56, no. 3/4, Paper 7, pp. 7:1–7:13, May/Jul. 2012.
10. ACE, The Automatic Content Extraction Projects. [Online]. Available: http://projects.ldc.upenn.edu/ace/
11. P. Grice, “Logic and conversation,” in *Syntax and Semantics*. New York: Academic, 1975.
12. 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.
13. J. A. Robinson, “A machine-oriented logic based on the resolution principle,” *J. ACM*, vol. 12, no. 1, pp. 23–44, Jan. 1965.
14. M. Kay, “Unification grammar,” Xerox Palo Alto Res. Center, Palo Alto, CA, Tech. Rep., 1983.
15. F. Pereira and S. Shieber, “The semantics of grammar formalisms seen as computer languages,” in *Proc. 10th Int. Conf. Comput. Linguistics*, Stanford, CA, 1984, pp. 123–129.
16. S. Shieber, “An introduction to unification-based approaches to grammar,” *The Journal of Symbolic Logic*, vol. 52, no. 4, p. 1052, 1986.
17. A. Colmerauer, “Prolog II reference manual and theoretical model,” Groupe Intelligence Artificielle, Universite Aix-Marseille, Marseille, France, Tech. Rep., 1982.
18. F. M. Suchanek, G. Kasneci, and G. Weikum, “YAGO: A large ontology from Wikipedia and WordNet,” *Web Semantics Sci., Serv. Agents World Wide Web*, vol. 6, no. 3, pp. 203–217, Sep. 2008.
19. R. Hoffmann, C. Zhang, and D. S. Weld, “Learning 5000 relational extractors,” in *Proc. 48th ACL Annu. Meet.*, 2010, pp. 286–295.
20. C. Fellbaum, *WordNet: An Electronic Lexical Database*. Cambridge, MA: MIT Press, 1998.
21. S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, “Indexing by latent semantic analysis,” *J. Amer. Soc. Inf. Sci.*, vol. 41, no. 6, pp. 391–407, Sep. 1990.
22. D. Blei, A. Ng, and M. Jordan, “Latent Dirichlet allocation,” *J. Mach. Learn. Res.*, vol. 3, no. 4/5, pp. 993–1022, Mar. 2003.
23. C. Wang, J. Fan, A. Kalyanpur, and D. Gondek, “Relation extraction with relation topics,” in *Proc. EMNLP Conf.*, 2011, pp. 1426–1436.
24. 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.
25. 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.
26. 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.
27. M. Collins and N. Duffy, “Convolution kernels for natural language,” in *Proc. Adv. NIPS*, 2001, pp. 625–632.

28. S. Zhao and R. Grishman, "Extracting relations with integrated information using kernel methods," in *Proc. 43rd ACL Annu. Meet.*, 2005, pp. 419–426.
29. M. Zhang, J. Zhang, J. Su, and G. Zhou, "A composite kernel to extract relations between entities with both flat and structured features," in *Proc. ACL 21st Int. Conf. Comput. Linguistics/44th Annu. Meet.*, 2006, pp. 825–832.
30. T. T. Nguyen, A. Moschitti, and G. Riccardi, "Convolution kernels on constituent, dependency and sequential structures for relation extraction," in *Proc. EMNLP Conf.*, 2009, pp. 1378–1387.
31. Q. McNemar, "Note on the sampling error of the difference between correlated proportions or percentages," *Psychometrika*, vol. 12, no. 2, pp. 153–157, Jun. 1947.

Received July 26, 2011; accepted for publication December 8, 2011

Chang Wang *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (wangchan@us.ibm.com)*. Dr. Wang is a Research Staff Member at the T. J. Watson Research Center. His research areas include machine learning (manifold learning, transfer learning, and representation learning) and its application in natural-language processing and information retrieval. Before joining IBM in 2010, Dr. Wang received his Ph.D. degree from the University of Massachusetts at Amherst and did his dissertation on the topic of manifold alignment-based transfer learning.

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.

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.

Branimir K. Boguraev *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (bran@us.ibm.com)*. Dr. Boguraev is a Research Staff Member in the Semantic Analysis and Integration Department at the Thomas J. Watson Research Center. He received an engineering degree in electronics from the Higher Institute for Mechanical and Electrical Engineering in Sofia, Bulgaria (1974) and a diploma and Ph.D. degrees in computer science (1976) and computational linguistics (1980), respectively, from the University of Cambridge, England. He worked on a number of U.K./E.U. research projects on infrastructural support for natural-language processing applications, before joining IBM Research in 1988 to work on resource-rich text analysis. From 1993 to 1997, he managed the natural-language program at Apple's Advanced Technologies Group, returning to IBM in 1998 to work on language engineering for large-scale, business content analysis. Most recently, he has worked, together with the Jeopardy! Challenge Algorithms Team, on developing technologies for advanced question answering. Dr. Boguraev is author or coauthor of more than 120 technical papers and 15 patents. Until recently, he was the Executive Editor of the Cambridge University Press book series *Studies in Natural Language Processing*. He has also been a member of the editorial boards of *Computational Linguistics* and the *Journal of Semantics*, and he continues to serve as one of the founding editors of *Journal of Natural Language Engineering*. He is a member of the Association for Computational Linguistics.

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.