Special Questions and techniques

J. M. Prager
E. W. Brown
J. Chu-Carroll

Jeopardy!™ questions represent a wide variety of question types. The vast majority are Standard Jeopardy! Questions, where the question contains one or more assertions about some unnamed entity or concept, and the task is to identify the described entity or concept. This style of question is a representative of a wide range of common question-answering tasks, and the bulk of the IBM Watson™ system is focused on solving this problem. A small percentage of Jeopardy! questions require a specialized procedure to derive an answer or some derived assertion about the answer. We call any question that requires such a specialized computational procedure, selected on the basis of a unique classification of the question, a Special Jeopardy! Question. Although Special Questions per se are typically less relevant in broader question-answering applications, they are an important class of question to address in the Jeopardy! context. Moreover, the design of our Special Question solving procedures motivated architectural design decisions that are applicable to general open-domain question-answering systems. We explore these rarer classes of questions here and describe and evaluate the techniques that we developed to solve these questions.

Introduction

One of the more challenging aspects of Jeopardy!** is its wide variety of questions. Not only do Jeopardy! questions span many subject domains but they also come in a variety of forms. The most common form by far is the Standard Jeopardy! Question. Standard Questions consist of statements expressing one or more assertions about an entity or concept, which remains unidentified in the question text. Answers are typically nouns or proper nouns but sometimes verbs or adjectives. The goal is to name the unidentified entity or concept based on relating the assertions in the question to what is expressed about the entity in some underlying body of natural-language content, e.g.,

DELICACIES*:* Star chef Mario Batali lays on the lardo, which comes from the back of this animal's neck. (Answer: "pig")

It should be noted that often not all information in a question describes or relates directly to the answer. It is the job of the question-answering system to tease out the relevant information for answering the question.

Beyond Standard Questions, however, a small fraction of the questions in Jeopardy! require a specialized computation to derive an answer or a constraint about the answer. We call any question that requires such a specialized computational procedure, selected based on a unique classification of the question, a Special Jeopardy! Question. Sample Special Questions and Standard Questions that require special processing are shown here:

Puzzle: ASTRONOMICAL RHYME TIME: Any song about earth's natural satellite. (Answer: "Moon tune")

Multiple Choice: BUSY AS A BEAVER: Of 1, 5, or 15, the rough maximum number of minutes a beaver can hold its breath underwater. (Answer: "15")

Common Bond: BOND, COMMON BOND: Fan, form, poison pen. (Answer: "letters")

Digital Object Identifier: 10.1147/JRD.2012.2187392

Fill-in-the-Blank (FITB): SCHOOL OF ROCK: History: Herman's Hermits hit number 1 in 1965 singing "I'm" this man, "I Am." (Answer: "Henry VIII")

Constraint Bearing: ARE YOU A FOOD"E"?: Escoffier says to leave them in their shells & soak them in a mixture of water, vinegar, salt, and flour. (Answer: "Escargots")

Pun bearing: HIP-HOP ON POP: From this "M.D.": "It's like this & like that & like this & uh, its like that & like this & like that & uh" (Answer: "Dr. Dre")

Special Questions fit well in the standard DeepQA processing pipeline but may have their own candidate generation or scoring components and, in some cases, their own machine learning model [1]. In the question analysis paper in this issue [2], we discuss the Special Question types, describing how they are recognized and what annotations have been created as a result. In this paper, we focus on processing techniques that have been developed to answer the most common of these types, including FITB, Puzzle (questions with constructed answers), and Multiple Choice. We also describe the special processing techniques used for Standard Questions with special aspects, namely associated lexical constraints or puns.

The FITB question type is the most frequently occurring Special Question, representing approximately 4% of all Jeopardy! questions. The next most frequent is Puzzle (2%), followed by Common Bond (0.7%) and Multiple Choice (0.5%). Overall, most of these question types are relatively infrequent. However, since Special Question types often appear as entire categories (thus representing one sixth of a Jeopardy! round), they must be addressed for any system to be consistently competitive at Jeopardy!. In this paper, we describe the processing of FITB and various subtypes of Puzzle and Multiple Choice questions. Common Bond processing is described as a variation of "missing link" detection in [3]. Approximately 13% of questions are accompanied by a lexical constraint; 1% employ a pun on the answer.

Although each Special Question type may require its own unique mechanisms and resources for candidate answer generation and evaluation, there are three general techniques that are broadly applicable across question types. These techniques include the following:

1. *Question decomposition and answer synthesis*—A subset of Puzzle questions require decomposition into parts, solution of the parts, and then an algorithmic recombination into candidate answers. We discuss in this paper decomposition as applied to Special Jeopardy! Questions; a broader discussion of decomposition in IBM Watson* is given in [4].
2. *Use of hints such as lexical constraints and puns*—Lexical constraints, e.g., 10-LETTER "W"ORDS, usually describe the length of desired answers and/or letters that are contained in desired answers. Similarly, puns, e.g., *this* "*colorful*" *river*, describe a characteristic of the answer. This kind of information is not useful when searching for candidate answers (because the text of these hints is generally not collocated with the answers in our textual resources) but is highly useful when evaluating candidate answers.
3. *Category-based revealed-answer learning*—Occasionally, the presence of category-based lexical constraints and/or Special Question types is not immediately obvious from the category or question and must be inferred from answers that have already been revealed in the given category.

In the next section, we describe these general techniques in more detail. We then describe individual Special Question types, including how the candidate answers for each type are generated and scored, and the solving strategies used for each type. In the following section, we provide evaluation results and finish with related work and conclusions.

Techniques for Special Question processing

The special processing procedures that we have developed may be divided into two broad categories: techniques that cut across question types and those that are specific to a single type. The former are described in this section, whereas the latter are covered in a later section.

Question decomposition and answer synthesis

Many Puzzle questions, including all Before & After and most Rhyme Time questions, require decomposition for successful solution. For example, the answer to

BEFORE & AFTER: The "Jerry Maguire" star who automatically maintains your vehicle's speed. (Answer: "Tom Cruise control")

is a made-up phrase that is extremely unlikely to be found in any reference corpus and therefore is generated by performing two separate searches and combining the results. This decomposition and synthesis is handled in a common way, as illustrated in **Figure 1**. In general, the stylized forms of these questions indicate how to perform the decomposition.

Specifically, decomposition for Puzzle questions is performed by matching manually developed Prolog rules against the question's predicate-argument structure to identify its subparts. This process is similar to that for Standard Question parallel decomposition, described in [4], but is specialized to the styles and formats commonly associated with the Puzzle question types. The recognized subparts (called *SubQuestionSpans*) are then used to generate intermediate candidate answers, called

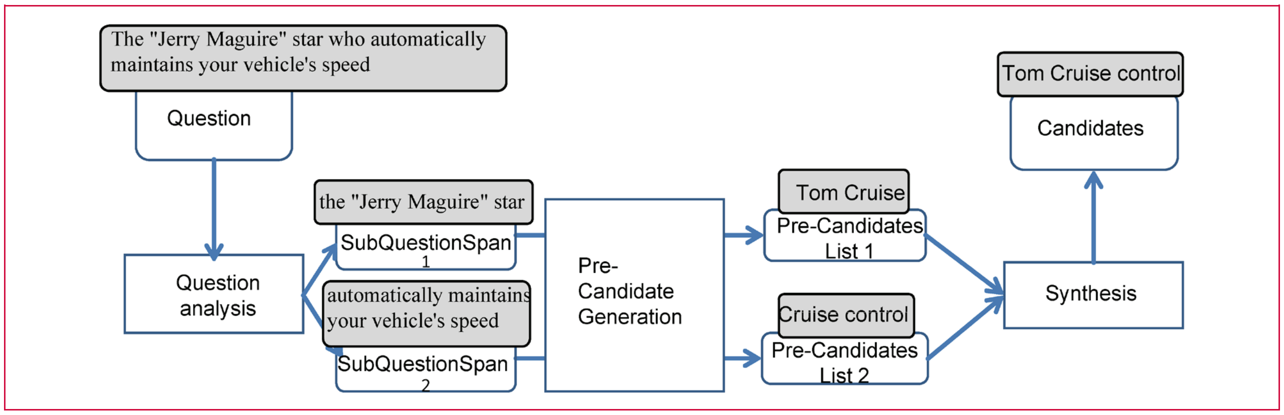

Figure 1

Decomposition flow. The general flow for the decomposition process is illustrated by the "Tom Cruise control" example. Approximately 100 precandidates are generated from either SubQuestionSpan, but only the ones that contributed to the final correct answer are shown.

precandidates, and the precandidates are brought together in a *synthesis stage* to form final candidate answers in the appropriate way (e.g., overlap for Before & After, rhyming for Rhyme Time).

Constraints and puns

A common feature of many Jeopardy! questions is that they contain hints to the answer in the form of lexical constraints and puns. Lexical constraints are typically requirements, most often found in the category but sometimes in the question text, that the answer be a certain number of letters or words long; or begin with, end with, or contain certain letters or letter sequences; rhyme with a certain word; or have other lexical properties. An estimated 13% of Jeopardy! questions, including approximately 35% of questions that are dictionary definitions of common terms, have one or more such constraints associated with them. Puns are much less frequent—about 1%—but are more challenging to process and are possibly more interesting because of the limited work to date in the field of computational humor (however, see, e.g., [5]). Although such properties are not used to generate candidates, they are used to score existing candidate answers on the basis of their degree of match with the recognized pun element. These hints in principle apply to any question type, but in practice, they occur mostly in Standard Jeopardy! Questions, both as suggestions to help the answering process and to ensure uniqueness when the answer may be ambiguous.

Lexical constraints

A lexical constraint might be instantiated:

STARTS & ENDS WITH "A": In 1981 this country acted as go-between to secure the release of the 52 Americans held by Iran. (Answer: "Algeria")

or may be instantiated only partially:

ONLY ONE VOWEL: Proverbially, you can be "flying" this or be this "and dry". (Answer: "high")

The second example is interesting in several respects: it has a lexical constraint, is decomposable, and is a FITB.

Lexical constraints may also be vague or ambiguous, e.g., quoted terms in the category indicate that the quoted text string is part of every answer, but it is often unclear (even to humans) whether the string has to start the answer, or start any word in the answer, or can be anywhere in the answer. Proper interpretation requires knowledge of the conventions typically used in Jeopardy!. Careful observation of the Jeopardy! style of questions and associated answers helped us fashion the constraint processing to be as narrow as possible without penalizing correct answers. On the basis of this analysis, we have developed a rule-based recognizer to identify the most frequently observed constraints, which we discuss in the next section.

The syntax of lexical-constraint-bearing categories is not always strictly English (THE L"ONE"LIEST NUMBER, IN THE "GN"OW, GOING INTO O__T) and sometimes borders on that of word problems (ONE O, THEN 2 O's). Our evaluation of our recognition algorithm on unseen questions showed that our existing recognizer fails to identify a small set of constraints expressed in unanticipated ways. However, since constraints typically occur in categories, patterns common in revealed answers in the category are useful for inferring constraints in recall failure cases. Therefore, we developed an inference process that complements our recognition process, which we discuss in the section "Learning from Revealed Answers."

Table 1 Sample constraints.

Constraint class	*Sample category*	*Sample answer*
Alliteration	ALLITERATIVE ARTISTS	Pablo Picasso
Blank	ON ____	On Deck
DoubleLetter	DOUBLE “H”	Fishhook
CommonEnding	-OID	Celluloid
NLetterRepetitions	“U”2	Futures
Rhyming	RHYMES WITH JOCK	Frock
SubStringDisjunction	“CHURCH” AND “STATE”	Churchkey
SyllableCount	MONOSYLLABLES	Ain’t

Constraint objects

Constraint processing is handled by the Constrainer component, which consists of *constraint classes* and management infrastructure. Each constraint class handles a different kind of constraint and has the following capabilities:

- Recognition and instantiation of a constraint class from the question.
- Inference of a constraint class from revealed answers in the same category.
- Testing a candidate for passing or failing the constraint.

Constraint classes are manually developed to cover the most common cases of lexical constraints. For example, one class consists of constraints that specify the length in characters for the answers and another of constraints that specify substring requirements for the answers. Instantiated constraint objects can work in conjunction to ensure that all the constraints are satisfied. For example, from the category 10-LETTER “W”ORDS, two constraint objects are constructed. One passes only answers that contain ten letters, and the other passes only answers that begin with the letter “w”; these are used together to determine whether candidate answers satisfy the entire category constraint.

There are a few dozen different constraint classes; a selection of these, along with categories that trigger them and answers that pass, is shown in **Table 1**. All of these particular examples derive from the category, but some instances are occasionally found in the question text, e.g., “Like young canines, baby rats are known by this *one-syllable* term”.

For each question, the Constrainer process is run after question analysis, and every constraint object analyzes the question to see if it should become active. An *ActiveConstraintSet* of zero or more active constraint objects will be constructed as a result. In the answer-scoring process, each candidate answer produced by Watson [6] is tested by the ActiveConstraintSet and is given features to indicate whether no lexical constraints exist for this question or whether it passed or failed if constraints are identified.

Puns

As mentioned in the paper on question analysis [2], puns that occur in the Jeopardy! category are usually solely for entertainment purposes—to dress up a potentially dry subject in a catchy phrase, such as “THE MALE IS IN THE CZECH”, about Czechoslovakian men—but most often do not need to be understood or processed in any different way to answer the questions themselves. Puns in the questions, on the other hand, are often clear hints at the answers and benefit from being directly addressed.

The most common repeating form of puns that we have recognized—and the only kind that we attempt to solve—is in the form of a quoted phrase modifying the question focus. We identified four subclasses, listed here with an example of each:

SUBSTRING: SLOGANEERING: In 1987 using the slogan “Catch the wave”, Max Headroom pitched this “new” soft drink. (Answer: “New Coke”)

SYNONYM: GEOLOGY: Due to its luster, German miners gave this “pleasant” rock its name, which means “spark”. (Answer: “gneiss”, whose homophone “nice” is a synonym of “pleasant”)

INSTANCE: *F*: An opera from 1900 gave wing to this popular “insect” composition by Rimsky-Korsakov. (Answer: “Flight of the Bumblebee”)

ASSOCIATION: I DO KNOW JACK: Marilyn Monroe made her major TV debut in 1953 on this “stingy” comedian’s TV show. (Answer: “Jack Benny”)

Lemmatization is performed on the pun term and the answer candidate, and when either is a multiword term, matching is attempted with individual terms as well as the whole string.

Homophones of each are included in the match operation (see the SYNONYM example above where "pleasant" is matched with "gneiss" through "nice"). For the SUBSTRING type, the pun term is a substring of the answer. For SYNONYM, the pun term is synonymous with an answer term, using WordNet** [7]. For INSTANCE, an answer term is an instance or subtype of the pun term, using YAGO TyCor [8]. For ASSOCIATION, there is a strong association (but not through any of the previous three kinds) between an answer term and the pun term; we test for such association by measuring the degree of correlation between the two terms using a corpus of n-grams extracted from a collection of English text consisting of encyclopedia and newswire articles [3]. This degree of association is compared with an empirically determined threshold to decide whether an association pun relationship exists.

Of course, it is not known in advance which of these types of pun match is in play; therefore, each is attempted, and if any test succeeds, a pun is asserted. This results in a PUN feature being generated for Watson's machine learning phase.

Learning from revealed answers

Constraint and question class inference

Jeopardy! provides a form of feedback within a category by making the correct answer available to the players after each question is played. Watson uses these *revealed answers* to improve its analysis results, such as for lexical answer type (LAT) detection [2] and identifying certain category-question relationships [10]. In this paper, we describe how revealed answers are used in constraint recognition and inference, as well as in question classification.

Watson's revealed-answer constraint learning allows it to infer a constraint that was missed when analyzing the category or to remove a constraint that was incorrectly detected or inferred for the category.

A constraint object inferred on the basis of revealed answers must satisfy the following two conditions: 1) The constraint object must pass all answers so far revealed in the category; and 2) the a priori probability of this constraint is low enough for Watson to confidently hypothesize the presence of the constraint. In other words, we are looking for situations where it is very unlikely that the revealed answers would happen by chance to satisfy the conditions for a (nonoperative) constraint. Condition 2 is established by a formula based on Bayesian inference that uses statistics derived from a large corpus (~75,000) of previously seen Jeopardy! answers; this corpus provides the prior probability that a conditioned constraint applies to a random answer. We used this to compute the probability that a constraint is operative given the (1, 2, 3, or 4) revealed answers so far in the category. When this probability exceeds an empirically determined threshold (in the Watson system, this threshold is set to 96%), the constraint is made active for the remaining questions in the category. As with constraints activated by regular recognition, retraction will occur if future revealed answers do not pass the constraint.

Some Special Question types also undergo the same treatment: the type of the question and the solving mechanism required for it are of a "classical" well-known variety, but the usually clear category indicator is disguised using some kind of wordplay, such as using "WHIRLED CAPITALS" to mean "WORLD CAPITAL ANAGRAMS". Without some additional detection mechanism, the type of the category will not be recognized; hence, five questions will potentially be lost. The same revealed-answer learning process used for lexical constraints is used for Special Question type inference.

In **Table 2**, we give examples of learning of both a lexical constraint and a Puzzle type (anagram) that had occurred during the Watson development period. Column 4 in the table gives the estimated probability that the constraint/question type is in play at this point, assuming a top-to-bottom order of question selection, and column 5 indicates whether that probability exceeded the threshold and, hence, whether the constraint/type was triggered. For the first group of questions, the system learned after three revealed answers that answers had to contain four letter "I"s. For the second group, it learned that the answer was an anagram of all or a section of the question after having seen just two answers.

Revealed-answer learning is in place for two reasons: 1) to react to situations that have been observed in training data and for which programmed mechanisms are in place, but because of unexpected phraseology in test data, the occurrences are not recognized; and 2) bugs or other deficiencies in the recognition process. As might be expected over time, occurrences of the second diminished, but the former continued to occur at a low level. In the section on evaluation, we show how this learning can complement recognition of constraints.

Selected Special Question types

In this section, we discuss the processing of three of the most common Special Jeopardy! Question types: Puzzle, Multiple Choice, and FITB.

Puzzle

Puzzle questions are those questions for which, in general, one would not expect any text corpus to contain the answer along with the facts asked about in the question. Instead, the answer must be constructed. The most obvious type is possibly Math, where the question is a mathematical expression that must be evaluated, but there are also Before & After, Rhyme Time, Anagram, and Hidden Words. We

Table 2 Examples of categories with solving hints/directions, which DeepQA did not recognize directly but could learn from revealed answers.

Category	*Question*	*Answer*	*Probability*	*Triggered*
I-I-I-I	It's the inflammation of the gums	Gingivitis	.0062	No
	On Dec. 10, 1817 it entered the Union as the 20th state	Mississippi	.0089	No
	In this early '90s video game by Sid Meier, you built "an empire to stand the test of time"	Civilization	.5917	No
	It's the process of converting data into a form that can be used by computers	Digitization	.9957	Yes
	Voters who "take" this can propose their own constitutional amendment	Initiative	.9999	Yes
WHIRLED CAPITALS	Non old	London	.00017	No
	In vane	Vienna	.9453	No
	I rasp	Paris	.9999	Yes
	Ape rug	Prague	.9999	Yes
	In silk, eh?	Helsinki	.9999	Yes

had observed that many Puzzle subtypes occurred only once or twice in a large set—for example, those involving computing Scrabble** scores of words or converting Roman numerals. We are not going to address these here; in addition, the processing of these questions, once recognition has occurred, is relatively straightforward. We instead describe what we have observed to be some of the most common Puzzle subtypes, namely Before & After, Rhyme Time, Math, Anagrams, and Hidden Words.

For the most part, puzzle types are indicated by the category, but occasionally, they are specified in the question text itself. Below are some examples of puzzle types that we discuss in this paper:

BEFORE & AFTER: The "Jerry Maguire" star who automatically maintains your vehicle's speed. (Answer: "Tom Cruise control")

RHYME TIME: A class that teaches you all about equines. (Answer: "horse course")

MATH: Twenty-three plus eight minus sixteen. (Answer: "15")

SHAKESPEAREAN ANAGRAMS: She's "one girl" King Lear should have been leery of. (Answer: "Goneril")

HIDDEN U.S. STATES: Linguistically, did ahoy enter the English language from sailors? (Answer: "Idaho")

Each of these puzzle types requires a special recognition and solving strategy, although some of them share a common infrastructure. In this section, we focus on how candidate answers for different types of puzzle questions are generated. To facilitate scoring of candidate answers, each candidate is given a puzzle rank and a puzzle score feature. A candidate answer's puzzle rank feature value is its rank in the candidate list when sorted by puzzle score. The features are used in the final merging component [1] to help compute the overall rank of each answer. An example of the advantage of using a combination of Puzzle and other features is shown in

ANAGRAMS: It's the only kitchen appliance Rosetta knows how to use (Answer: "toaster")

"Toaster" is not the only anagram of one of the question words, but it is the only one that is an instance of the LAT "appliance"; therefore, only it will generate a good TyCor score [8].

Before & After and Rhyme Time

A characteristic of all Before &After and the large majority of Rhyme Time answers is that they are made-up phrases that do not naturally occur in any text and must be synthesized. Before & After questions always contain definitions or descriptions of two different entities that must be overlapped to give the answer. The majority of Rhyme Time questions contain definitions or descriptions of two different entities that must rhyme. Consider the following examples where each SubQuestionSpan is underlined:

BEFORE & AFTER: 1961 invasion of Cuba wrapped in a tasty pastry. (Answer: "Bay of pigs in a blanket")

AUTHORS' RHYME TIME: Ian's Scandinavian rodents. (Answer: "Fleming's lemmings")

To solve these questions, the system runs Hypothesis Generation on each SubQuestionSpan to generate the precandidates and then synthesizes final candidate answers as follows:

- For Before & After, only multiword precandidates are retained. Full candidate answers are synthesized by pairing precandidates generated from different SubQuestionSpans in which the trailing word(s) from one precandidate matches the leading word(s) from the other precandidate. For example, "Tom Cruise" and "Cruise control" would result in the candidate answer "Tom Cruise control". The puzzle score is calculated by taking the product of the reciprocal of each precandidate answer's rank, multiplied by the number of words in the overlap sequence (which is typically one).
- A similar calculation happens for Rhyme Time, except that the pair of precandidates are tested for rhyme and not overlap. The output candidate is the concatenation of the two precandidates. The order of the precandidates in the composed answer is determined by which one corresponds to the main clause or head noun in the question—that one becomes the head noun of the answer with the other precandidate prepended as a modifier. In the case of the possessive-plural construction as presented in the example above, the answer's syntax directly mirrors the question's syntax. The puzzle score is calculated by taking the product of the reciprocal of each precandidate answer's rank, multiplied by the pair's rhyme-score.[1]

If the category has a modifier (e.g., *SUPREME COURT* BEFORE & AFTER, *EDIBLE* RHYME TIME), then the precandidate generation process is run twice. In the first pass, the modifier is appended to the left SubQuestionSpan, and in the second to the right one, as the modifier may be applicable to different spans in different questions.

About 10% of Rhyme Time questions are not puzzle questions, but rather have naturally occurring rhyming phrases as answers. For example,

RHYME TIME: The pacifistic influence of the 1960s hippie movement. (Answer: "Flower Power")

These are not syntactically distinguishable from the Rhyme Time questions of the puzzle variety. Thus, for Rhyme Time, an additional path is used whereby the question text is treated as a search string, and the candidate answers are constrained to be rhyming phrases.

Math

There are three kinds of Math questions that are considered "special," as illustrated here:

1. MATH CLASS: It's the square root of the square root of 16. (Answer: "2")
2. GEOMETRY: If 3 angles in a quadrilateral are 40°, 140° and 80° then the other angle must be this many. (Answer: "100")
3. YOU DO THE MATH: The number of legs on a spider plus the number of legs on a fly. Answer: "14")

The kind exemplified by question no. 1 above is explicit arithmetic expressions to be evaluated, sometimes with surrounding language such as "It is . . ." or ". . . equals this". Such math expressions are evaluated by a Definite-Clause Grammar (DCG) that parses the expression and produces a parse tree. The DCG recognizes any mix of alphabetic and numeric representations of numbers and operators. The tree is traversed and recursively evaluated to a number.

The second kind, as in question no. 2, describes a situation, usually geometric, for which substitution of given numbers into an unspecified formula is required. The component that handles these consists of approximately 20 geometrical and arithmetic objects (e.g., diameter, triangle, and sequence) and a similar number of formulas relating values of these objects to a computed result. This set was developed to cover the math level observed in past games—clearly not all of geometry or arithmetic. Question analysis determines which objects are given values in the question and which are asked for. The formula that computes the answer to question no. 2 corresponds to the following English rule:

If the diameter is mentioned and the circumference has a defined value, compute the circumference's value divided by pi.

The third form, as illustrated in question no. 3, asks for "the number of X" or "the number of X in Y", possibly combined with another instance of such or a literal number (e.g., "BODY COUNT: Usual number of ribs divided by 8").

[1]The rhyme score is in the range of 0–1 and is calculated by a utility that has access to a phonetic dictionary (the *Carnegie Mellon Pronouncing Dictionary* cmudict.0.6, with extensions) and calculates what contiguous fraction of the phonemes match, going from right to left. Out-of-vocabulary words have their phonetic scheme estimated by lexical similarity to words in the phonetic dictionary. For RHYME TIME, only precandidate pairs that have a mutual rhyme score of greater than 0.35 are considered to rhyme and to generate an output candidate.

The combining is assumed to be one of $+$, $-$, $\times$, and /. The presence of a combining operator and a second argument is another example of the need for question decomposition.

Two techniques are used for calculating the value of the "number of" phrases. One approach performs a lookup that returns a ratio between two units of measurements. If that approach fails, a search against an n-gram corpus is used to identify the integer most closely associated with the terms in the question (e.g., "legs" and "spider").

Anagrams and Hidden Words

We observed from our development data that answers to Anagrams and Hidden Words, whether common words or proper names, are covered by one of two resources: 1) titles of Wikipedia** documents—sometimes via redirection—or 2) entities found in the instance lists maintained by R2, our named entity recognizer. This observation enabled us to use a simple generate-and-test regimen based on these resources.

We generated two resources for these question types. The typed instance sets (TISs) are dynamically extracted word lists from R2 when the category has a modifier that we recognize as the common-word description or (rough) synonym of a semantic type. Thus, the category "ANIMAL ANAGRAMS" triggers the instance lists of type *Animal*; the category "ANAGRAMMED THEORETICAL PHYSICISTS" triggers *Scientist*. The untyped instance set (UIS) is a list of all of the Wikipedia titles.

For Anagrams, a series of heuristics is applied in order to determine the section of the question that provides the anagram letters:

- The entire question, if it is less than a certain length.
- The span that precedes or follows a colon to the beginning or end of the question, if less than a certain length.
- Any word or quoted phrase in the question.

If the letters in the Anagram span are determined to be a rearrangement of the letters in any term in an instance set, then that term is generated as a candidate. This operation is first performed with the TIS if the category specifies a type that triggers an instance list. If no candidates are found, the processing is performed with the UIS. In the case of more than one resulting candidate, the list is sorted with longer entries first; in the case of same-length entries, the term with the lower inverse document frequency is preferred.

In Hidden Words questions, the answer is a word formed from letters that appear consecutively in the question, but not as a word per se, and are assumed to cross word boundaries. In the example given earlier, the answer "Idaho" is found inside "... d*id aho*y ...". For Hidden Words, any term in an instance set that is a substring of the question with spaces removed but is not a substring of the original question becomes a candidate. The candidate list is sorted in the same way as for Anagrams above.

Multiple Choice

Multiple Choice questions provide the correct answer in the category or question text as a choice among three or more possible answers. The rest of the category or question text provides a question or relation that must be solved to select the correct answers. Here are some examples:

THE SOUTHERNMOST NATION: India, Italy, Iceland. (Answer: "India")

BRANSON, HANSON OR MANSON: A city in the Ozarks. (Answer: "Branson")

In the first example, the category gives a relation that must be used to select the correct answer. In this particular case, the relation is a geospatial superlative relation applied to all candidate answers, which are countries, in aggregation. In the second example, the question text gives a question that, on its own, can return more than one valid answer, but in combination with the multiple choices given in the category, selects the correct answer from among the choices.

To solve Multiple Choice questions, Watson must detect the Multiple Choice question (i.e., classify the question as a Multiple Choice question), identify each of the answer choices (which may appear in the category or the question text), and then use the rest of the category or question text to select the best answer. Multiple Choice classification and identification of the answer choices occurs during question analysis using rule-based analysis and heuristics. This is described in more detail in [2].

After question analysis, Watson routes Multiple Choice questions to a Multiple Choice candidate generation component. This component generates each answer choice as a candidate answer. To score these candidates, it checks if the category or question text contains a known relation to solve. The current set of known relations is limited to geospatial relations that ask about relative size, population, or location (e.g., Northernmost Capital). If the system detects one of these relations, it performs entity identification on the answer choices to map them to the actual geographical entities, finds the geographical entities in DBpedia [10], extracts the appropriate attributes from the DBpedia entry (e.g., latitude and longitude), and invokes an appropriate reasoning method to solve the relation. The Multiple Choice answer selected by the relation is assigned a Multiple Choice Feature score of 1, and the other Multiple Choice candidates are assigned a 0. If no relation is detected, all Multiple Choice candidates will receive a feature score of 1 to distinguish them from candidates generated via other means.

During final merging and ranking [1], we use a machine learning model trained specifically for Multiple Choice questions that emphasizes the Multiple Choice feature. Answer score features generated by the rest of the analytics in Watson are then used to distinguish the correct answer from among the Multiple Choice candidates.

Fill-in-the-Blank

FITB questions come in a few different styles, but in general, they require completing a phrase, a quote, or a title. The missing words to be returned in the response may be explicitly represented with a blank or more implicitly represented as a LAT adjacent to a quoted phrase. Here are some examples:

> COMPLETES THE PLAY TITLE: Howard Sackler's "The Great White __". (Answer: "Hope")
>
> INVERTEBRATES: It completes the expression "happy as a" this "at high tide". (Answer: "clam")
>
> DOUBLE, DOUBLE: The classic name for the double promontories at the Strait of Gibraltar is "the Pillars of" this mythic hero. (Answer: "Hercules")

The first example requires completing the title of a play, the second example requires completing a common expression, and the third example requires finding the missing word in the name of a famous physical object.

Watson solves a FITB question by first detecting the FITB question during question analysis (i.e., classifying the question as a FITB) and identifying the missing portion of the quote or phrase that must be found. The missing portion is typically represented as a "blank," such as underscores, or as a LAT adjacent to a quoted phrase. In the latter case, the LAT is often just a pronominal, e.g., "this", but it may also be a specific type or class, e.g., "this mythic hero".

Watson runs the rest of FITB processing as a candidate answer generator, operating on the results of primary passage search [6]. For each primary search passage, the FITB candidate answer generator searches the passage text for the quoted phrase or relevant context from the question. This search requires some fuzzy matching in case the question text does not exactly match the wording in the passage. When a match is found, the FITB candidate answer generator generates candidate answers by extracting the text from the passage that corresponds to the "blank" in the question text. This text may produce multiple candidate answers as longer and longer phrases are extracted as candidate answers, up to sentence boundaries.

The candidate answers receive a FITB rank feature and a FITB boundary feature for use in the final merging and ranking model. The rank feature is based on the search rank of the passage from which the candidate answer was extracted, and the boundary feature characterizes the boundary between the matched question text and the extracted text corresponding to the blank, recognizing that certain boundaries (e.g., nothing separating the matched question text and candidate answer, all within quotes) are stronger indicators than other boundaries (e.g., a semicolon separating matched question text and candidate answer).

Evaluation

Special Questions

We evaluate two aspects of processing for the Special Questions described in this paper—the accuracy of each Special Question type classifier, and the question-answering accuracy of the end-to-end system on each Special Question type. To evaluate the performance of the Special Question type classifiers, we generated a gold standard for question type classification by manually annotating the type of all questions in a test set of 3,571 blind questions, extracted from 66 Jeopardy! games after removing audiovisual questions. The results (including F measure, accuracy, and Precision@70[2]) are shown in **Table 3**.

From the table, we see that the performance of the Special Question type classifiers is fairly consistent for the measured types. Generally, the Special Question types are quite rare in this test set; FITB is the most common, at 3.8%. Since our test set was constructed from randomly selected Jeopardy! games, these frequencies reflect the actual distribution of these question types in our data set. Moreover, the Puzzle type contains many subtypes, including a large number that have been observed in our development data a very small number of times (possibly never before this particular test set). We did not develop solvers or recognizers for these rare types; therefore, the Puzzle recognition recall (0.525) was low. On the other hand, the recognition precision for the more common types that we did have solvers for was high (0.977).

Question-answering accuracy of the more common Puzzle types was evaluated on a different blind set of 10,000 questions in order to obtain a large enough set of questions for each Special Question type. The accuracy for questions recognized as Before & After was 39% (29/75), Rhyme Time 30% (15/49), and Anagrams 80% (16/20). The low values for Before & After and Rhyme Time reflect the fact that

[2]Precision@N is the precision of the system if it answers only the top N% of questions for which it is most confident in its top answer.

Table 3 Selected Special Question recognition and question-answering accuracy on 3,571 questions (FITB, Fill-in-the-Blank.)

	Puzzle	*FITB*	*Multiple-Choice*
Number detected	43	142	20
Number in gold standard	80	135	19
Number correct	42	96	13
Recognition precision	0.977	0.676	0.650
Recognition recall	0.525	0.711	0.684
F_1	0.683	0.693	0.667
Frequency in gold standard	0.023	0.038	0.005
Answering accuracy	0.512	0.803	0.6
Answering Precision@70	0.645	0.970	0.571

Table 4 Before & After failure modes and frequencies.

Category of failure	*Detail*	*Percentage of questions*
Question analysis	Parse problem	4%
	Incorrect decomposition	11%
Search	No recall in one search	19%
	No recall in both searches	5%
Algorithm deficiency	Question does not cleanly break into left and right parts	3%
	Pre-candidates do not overlap on word boundaries	2%
Scoring	Correct answer not in first place	5%
Other		5%

many operations all have to work successfully to get a correct answer. A failure analysis of 184 Before & After questions culled from a sample of 30,000 questions revealed the failure causes listed in **Table 4**. This analysis revealed that there were several different causes, but the most prominent were as follows:

1. Search failure (24%)—Further analysis revealed that this was mostly due to the individual SubQuestionSpans often being underspecified (not enough information to determine a unique answer) and/or not containing a focus. We experimented with an approach that took a high-scoring precandidate from one SubQuestionSpan, "peeled off" the word on the edge, and added it to the other search. This gave encouraging results but made the response time unacceptably slow.
2. Incorrect decomposition (11%)—Rules that rely on syntax alone (e.g., noun phrase plus relative clause will split at the relative pronoun) do not seem to be sufficient to correctly match all cases.

Constraints and puns

To evaluate the impact of the Constrainer component, we ablated it from the system, retrained all of the models, and ran this "baseline" system on the test set of 3,571 blind questions used above. The results are reported in **Table 5**. On the entire test set, the Constrainer improves accuracy by 1.8% (from 0.687 to 0.705) and Precision@70 by 2.4% (from 0.855 to 0.879). If we consider just the 561 questions in the test set where a constraint is detected, the Constrainer improves accuracy by 11.4% (from 0.636 to 0.750) and Precision@70 by 12.5% (from 0.814 to 0.939). These are statistically significant via McNemar's test with a two-tailed P value less than 0.0001. These results clearly show that for those questions with a lexical constraint, the Constrainer provides a substantial performance improvement.

Table 5 Evaluation of Constrainer on 3,571 question test set.

	Baseline (no constrainer)	*With constrainer*
	All 3,571 questions	
Accuracy	0.687	0.705 (+1.8%)
Precision@70	0.855	0.879 (+2.4%)
	561 questions where a constraint was detected	
Accuracy	0.636	0.750 (+11.4%)
Precision@70	0.814	0.939 (+12.5%)

Table 6 Effect of revealed-answer learning of constraints in 122 categories in 3,571-question test set.

Property	*Count* (%)
Forwards constrainer passed correct answer	119 (98)
Inferred constrainer passed correct answer	119 (98)
Both constrainers passed correct answer	118 (97)
Forwards constrainer rejected incorrect answer	98 (80)
Inferred constrainer rejected incorrect answer	101 (83)
Either constrainer rejected incorrect answer	114 (93)

This is not surprising as lexical constraints provide both important clues for and restrictions on the correct answer.

End-to-end system testing on 23,000 questions showed a 2.5% accuracy improvement on the 1% of questions where the pun feature was active.

Revealed-answer learning

We performed an experiment to measure to what extent revealed learning could compensate for recognition deficiencies without hurting performance with overaggressive inference.

For the same blind test set of 3,571 questions, we manually identified 122 categories with a lexical constraint. For each such category, we established the following:

- The last question in the category (so that if taken in top-bottom order, there would be four revealed answers available).
- The correct answer.
- A manually constructed answer of the type expected by the question, but not observing the stated constraint.

For each of these questions, we generated the "Forwards" Constrainer by analysis of the category string, and the Inferred Constrainer from the revealed answers. We measured to what extent these constrainers individually and combined

- Passed the correct answer, and
- Failed the wrong answer.

The results are reported in **Table 6**. In summary, by including revealed-answer learning, we lost one correct answer but rejected 16 wrong answers that the *Forwards Constrainer* had passed. Although the individual constrainer performances at rejecting wrong answers were similar to each other, there was sufficient lack of overlap that the combination was significantly more powerful than either individually.

Related work

This paper focuses on special techniques in the Watson system for handling question classes highly unique to Jeopardy!. While to the best of our knowledge, no other system has been developed in this domain, the techniques that we have presented in this paper were designed with component and architectural applicability to general question answering in mind. We discuss related techniques for these general-purpose components in this section.

The question decomposition and synthesis architecture for solving Before & After and Rhyme Time questions is similar to earlier work on question decomposition [11] and Watson's own decomposition strategies for solving Final Jeopardy! questions [4]. However, Katz et al. [11] focused on what we refer to as nested decomposition questions [12], where a question is decomposed into two questions that must be solved serially. For example, "What is the population of the capital of Japan?" can be answered by first finding out the capital of Japan and then answering "What is the population of Tokyo?" For Watson's puzzle questions, the SubQuestionSpans are solved in parallel, with the final answers synthesized from precandidates from each subquestion.

The problem of determining the meaning of a word in context, narrowly construed, is the domain of Lexical Semantics [13] or, more broadly, is the concern of much of natural-language processing. Working with alternative meanings (i.e., those *not* consistent with the surface reading or current context) has been studied much less in a computational linguistics setting, although it is the mechanism through which puns operate. Kiddon and Brun [14] describe a system that identifies one specific kind of double entendre based on unigram features. A more general analysis of the problem of humor recognition is given in Mihalcea and Strapparava [5]. On the generation side, Ritchie et al. [15] describe a system that constructs simple pun-based jokes based on schemas that rely on similar attributes (synonymy, homophony) to those we use to detect puns.

In Watson's treatment of Multiple Choice questions, possible candidate answers as specified in the category or question are identified and evaluated using Watson's standard suite of answer scorers. The answer-scoring process causes additional supporting passages for the candidate to be retrieved, and the degrees to which these passages support the answer are used as features to rank the candidate [16]. This process is analogous to work on answer validation [17, 18] in verifying which one of the three possible choices is most likely the answer.

Conclusions

Although Special Questions represent a small fraction of all Jeopardy! questions, they often occur as entire categories and represent one sixth of an entire round; therefore, they must be addressed to obtain the best possible gameplay performance. Special Questions differ substantially from Standard Jeopardy! Questions and other question types that are usually explored in the area of question answering; hence, they require specially developed algorithms. We presented the most commonly occurring Jeopardy! Special Question types and the solutions that we implemented to address these questions. We also described several general techniques used to solve these questions. These included question decomposition and answer synthesis applied to Puzzle and selected Math questions; Constraints and Puns found in Standard Questions; and Learning from Revealed Answers, as applied to category-level question classes and lexical constraints. We presented evaluations of Special Question recognition and answering, as well as demonstrated the impact of lexical constraint processing and learning from revealed answers.

*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., Trustees of Princeton University, Hasbro, Inc., or Wikimedia Foundation in the United States, other countries, or both.

References

1. 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.
2. 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.
3. J. Chu-Carroll, E. W. Brown, A. Lally, and J. W. Murdock, "Identifying implicit relationships," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 12, pp. 12:1–12:10, May/Jul. 2012.
4. A. Kalyanpur, S. Patwardhan, B. K. Boguraev, A. Lally, and J. Chu-Carroll, "Fact-based question decomposition in DeepQA," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 13, pp. 13:1–13:11, May/Jul. 2012.
5. R. Mihalcea and C. Strapparava, "Learning to laugh (automatically): Computational models for humor recognition," *Comput. Intell.*, vol. 22, no. 2, pp. 126–142, May 2006.
6. 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.
7. G. Miller, "WordNet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995.
8. 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.
9. 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.
10. 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., Serv. Agents World Wide Web*, vol. 7, no. 3, pp. 154–165, Sep. 2009.
11. B. Katz, G. Borchardt, and S. Felshin, "Syntactic and semantic decomposition strategies for question answering from multiple sources," in *Proc. AAAI Workshop Inference Textual Question Answering*, 2005, pp. 35–41.
12. A. Kalyanpur, S. Patwardhan, B. Boguraev, A. Lally, and J. Chu-Carroll, "Fact-based question decomposition for candidate answer re-ranking," in *Proc. 20th ACM Conf. Inform. Knowl. Manage.*, 2011, pp. 2045–2048.
13. D. A. Cruse, *Lexical Semantics*. Cambridge, U.K.: Cambridge Univ. Press, 1986.
14. C. Kiddon and Y. Brun, "That's what she said: Double entendre identification," in *Proc. 49th Annu. Meeting ACL-HLT*, 2011, pp. 89–94.
15. G. Ritchie, R. Manurung, H. Pain, A. Waller, R. Black, and D. O'Mara, "A practical application of computational humour," in *Proc. 4th Int. Joint Conf. Comput. Creativity*, A. Cardoso and G. A. Wiggins, Eds., 2007, pp. 91–98.
16. 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.
17. J. Ko, L. Si, E. Nyberg, and T. Mitamura, "Probabilistic models for answer-ranking in multilingual question-answering," *ACM Trans. Inf. Syst.*, vol. 28, no. 3, p. 16, Jun. 2010.
18. B. Magnini, M. Negri, R. Prevete, and H. Tanev, "Is it the right answer? Exploiting web redundancy for answer validation," in *Proc. ACL 40th Anniv. Meeting*, 2002, pp. 425–432.

Received August 2, 2011; accepted for publication January 17, 2012

John M. Prager *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (jprager@us.ibm.com).* Dr. Prager has been working in technical fields related directly or indirectly to question answering for most of his professional career. Most recently, while at the T. J. Watson Research Center, he has been part of the Watson project, building a system that plays the *Jeopardy!* quiz-show game. He has been involved in both the algorithms area, in particular working on puns and other wordplay, and the strategy area. Previously, he led IBM's successful entries in Text REtrieval Conference Question-Answering (TREC-QA) tasks, an annual evaluation at the National Institute of Standards and Technology (NIST). Prior to that, he worked in various areas of search, including language identification, web search, and categorization. He has contributed components to the IBM Intelligent Miner for Text product. For a while in the early 1990s, he ran the search service on www.ibm.com. While at the IBM Cambridge Scientific Center, Cambridge, MA, he was the project leader of the Real-time Explanation and Suggestion project, which would provide users with help by taking natural-language questions and processing them with an inference engine tied to a large repository of facts and rules about network-wide resources. He has degrees in mathematics and computer science from the University of Cambridge and in artificial intelligence from the University of Massachusetts; his publications include conference and journal papers, nine patents, and a book on Alan Turing.

Eric W. Brown *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (ewb@us.ibm.com).* Dr. Brown is a Research Staff Member in the Semantic Analysis and Integration Department at the T. J. Watson Research Center. He received the B.S. degree in computer science from the University of Vermont, Burlington, in 1989 and M.S. and Ph.D. degrees in computer science from the University of Massachusetts, Amherst, in 1992 and 1996, respectively. He subsequently joined IBM, where he has worked on information retrieval, text analysis, and question answering. Since 2007, he has been a technical lead on the Watson project. Dr. Brown is a member of the Association for Computing Machinery and Sigma Xi.

Jennifer Chu-Carroll *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (jencc@us.ibm.com).* Dr. Chu-Carroll is a Research Staff Member in the Semantic Analysis and Integration Department at the T. J. Watson Research Center. She received the Ph.D. degree in computer science from the University of Delaware in 1996. Prior to joining IBM in 2001, she spent 5 years as a Member of Technical Staff at Lucent Technologies Bell Laboratories. Dr. Chu-Carroll's research interests are in the area of natural-language processing, more specifically in question-answering and dialogue systems. Dr. Chu-Carroll serves on numerous technical committees, including as program committee co-chair of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT) 2006 and as general chair of NAACL HLT 2012.