Fact-based question decomposition in DeepQA

A. Kalyanpur
S. Patwardhan
B. K. Boguraev
A. Lally
J. Chu-Carroll

Factoid questions often contain more than one fact or assertion about their answers. Question-answering (QA) systems, however, typically do not use such fine-grained distinctions because of the need for deep understanding of the question in order to identify and separate the facts. We argue that decomposing complex factoid questions is beneficial to QA systems, because the more facts that support an answer candidate, the more likely it is to be the correct answer. We broadly categorize decomposable questions into two types: parallel and nested. Parallel decomposable questions contain subquestions that can be evaluated independent of each other. Nested questions require decompositions to be processed in sequence, with the answer to an "inner" subquestion plugged into an "outer" subquestion. In this paper, we present a novel question decomposition framework capable of handling both decomposition types, built on top of the base IBM Watson™ QA system for Jeopardy!™. The framework contains a suite of decomposition rules that use predominantly lexico-syntactic features to identify facts within complex questions. It also contains a question-rewriting component and a candidate re-ranker, which uses machine learning and heuristic selection strategies to generate a final ranked answer list, taking into account answer confidences from the base QA system. We apply our decomposition framework to the particularly challenging domain of Final Jeopardy! questions, which are found to be difficult even for qualified Jeopardy! players, and we show a statistically significant improvement in the performance of our baseline QA system.

Introduction

A "single-shot" approach to answering questions assumes that the information given in a question can be found in close proximity to the correct answer in some document. This is typically true of Text REtrieval Conference (TREC) questions, which tend to exploit a single fact about the answer, e.g., "What is the capital of France?" and "How did Bob Marley die?" Because most question-answering (QA) systems are evaluated on TREC-QA[1] data, they tend to reflect this assumption and thus conform to the single-shot tactic.

Our Jeopardy!** data set, on the other hand, contains many questions in which multiple facts related to the correct answer are given, as shown in the following example:

> (1) This company with origins dating back to 1876 became the first U.S. company to have 1 million stockholders in 1951.

The above question contains two facts, i.e., "a company with origins dating back to 1876" and "a company that became the first U.S. company to have 1 million stockholders in 1951". Although the single-shot approach can apply to such multifact questions, we believe that it is more effective to use the individual facts because each may be justified and answered by a different source of evidence. Our hypothesis is that the more independent facts support an answer candidate, the more likely it is to be the correct answer.

[1]TREC-QA task: available at http://trec.nist.gov/data/qa.html.

Digital Object Identifier: 10.1147/JRD.2012.2188934

References to "fact" above appeal to any entity-relationship expression, where the relation is an *n*-ary predicate. Note that this generalizes the definition of tertiary expressions [1], which can encode triples (e.g., "He served as governor") but not facts with relations involving more than two arguments (e.g., "For most of World War II, he served as governor of the Bahamas").

On the basis of this notion of fact, we focus on two types of decomposable questions:

- *Parallel decomposable* questions contain mutually independent facts about the correct answer. Example (1) is parallel decomposable because its subquestions corresponding to the two facts identified above can be evaluated independently of each other; example (2) below also belongs to this category.
- *Nested decomposable* questions contain an independent fact about an entity related to the correct answer and a separate fact that links that entity to the correct answer. Solving these questions involves decompositions to be processed in sequence, with the answer to an "inner" subquestion plugged into the "outer," such as in example (3), where the inner fact "A controversial 1979 war film" can be solved first and its answer ("Apocalypse Now") substituted in the outer.

(2) FOOD & DRINK HISTORY: Its original name meant "bitter water" and it was made palatable to Europeans after the Spaniards added sugar.
(3) WAR MOVIES: A controversial 1979 war film was based on a 1902 work by this author.

As a source of decomposable questions, we look at Final Jeopardy! questions. They are more complex than regular Jeopardy! questions and consequently more difficult to answer. Even qualified Jeopardy! players find Final Jeopardy! difficult: Analysis of historical (human) performance data shows that their accuracy over Final Jeopardy! questions is less than 50%. The complexity of these questions can be ascribed to a common characteristic they share; namely, the set of facts collectively describing the answer is such that a search query constructed from all of them, taken together without logical decomposition, is likely to get "flooded" by noise and not find meaningful answer-bearing passages. This is because Final Jeopardy! questions often describe widely diverse aspects of the answer, tending to require a solution strategy that must address these separately, as opposed to the more traditional "single-shot" approach. In many cases, this separation and subsequent composition of recursively derived subanswers is what we intend to solve by question decomposition. Given the complexity of Final Jeopardy! questions, demonstrating impact on the Final Jeopardy! question set is a challenge.

Complex factoid questions are not specific to Jeopardy! and can be found in other application domains such as medical, legal, and so forth. For instance, examples (4) and (5) illustrate parallel and nested decomposable questions outside of the Jeopardy! domain.

(4) Which 2011 tax form do I fill if I need to do itemized deductions and I have an IRA rollover from 2010?
(5) Which surgical procedure is required to deal with an aortic condition associated with bicuspid aortic valves?

In the remainder of this paper, we discuss strategies for solving these types of "complex" questions and present a novel decomposition approach developed for DeepQA. We demonstrate how it enhances the IBM Watson* system for answering Final Jeopardy! questions.

Question decomposition and synthesis framework

Regardless of whether parallel or nested, decomposable questions require orchestrated solving of subquestions identified therein, followed by appropriate use of multiple candidate answer lists to derive the correct answer, with increased confidence, as all evidence has been taken into consideration. Each question subpart (aligned with a fact) constitutes a new question that the system must answer before answering the original question. Consequently, we adopt a "meta" framework for answering decomposable questions, one in which an existing QA system is invoked to answer the subparts of the original question, as shown in **Figure 1**.

The key components of the framework are as follows:

a) *Decomposition recognizers*, which analyze the input question and identify decomposable parts using a set of predominantly lexico-syntactic cues.
b) *Question rewriters*, which rewrite the subquestions (facts) found by the recognizer, inserting key contextual information.
c) *Underlying QA system (configured without decomposition)*, which, given a factoid question, generates a ranked list of answer candidates, each with a *confidence* corresponding to the probability of the answer being correct. In our work, we use the Watson system, although in general, any QA system can be plugged into our meta-framework, as long as it satisfies two basic properties: 1) ability to solve factoid questions by providing answers with confidences that reflect correctness probability; and 2) separation of context (or theme or topic) information of the question from its main content by weighing the former less than the latter. In the Jeopardy! data, such information is contained in explicit question categories.

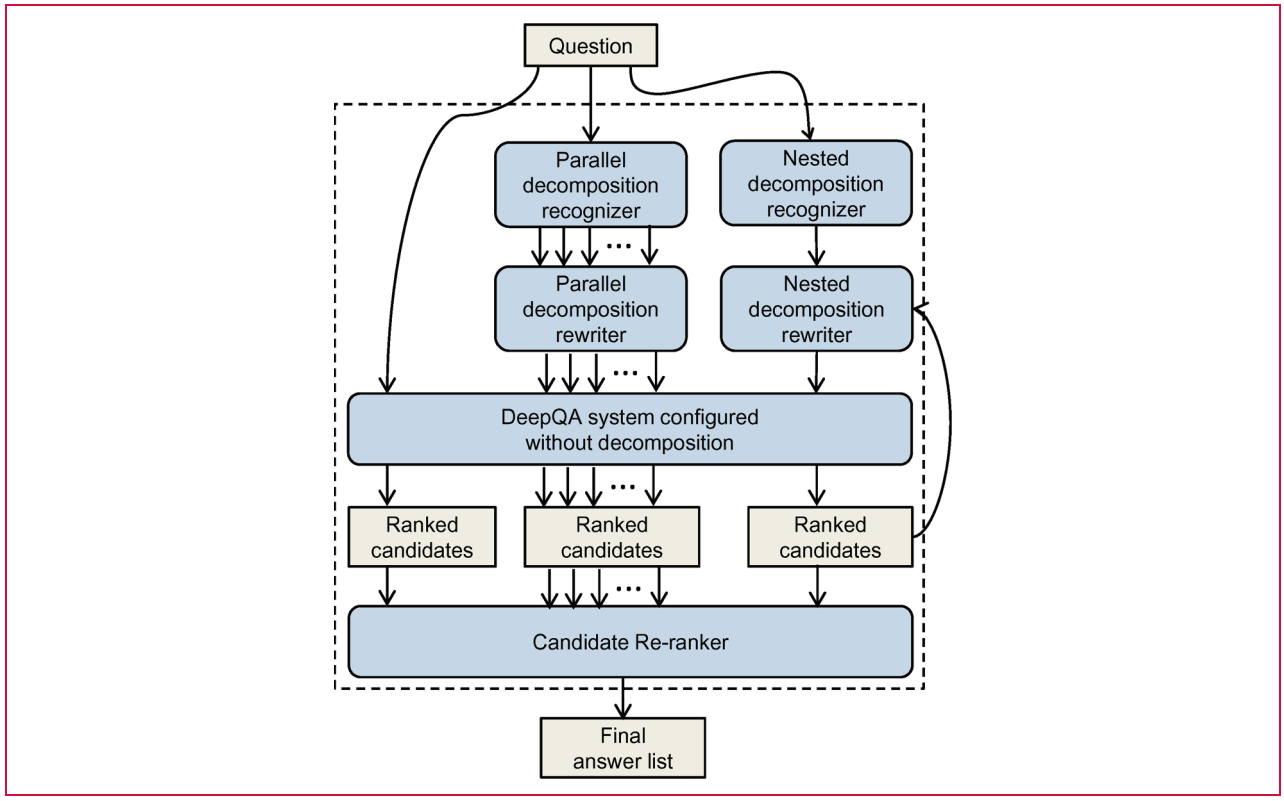

Figure 1

DeepQA system configured for question decomposition. (Used, with permission, from A. Kalyanpur, S. Patwardhan, B. Boguraev, A. Lally, and J. Chu-Carroll, "Fact-based question decomposition for candidate answer re-ranking," in the *20th ACM Conference on Information and Knowledge Management (CIKM) 2011* [poster], October 24–28, 2011, Glasgow, Scotland, UK. © 2011 ACM.)

d) *Candidate re-rankers*, which combine answers to the original question considered as a whole with those produced through decomposing the question to generate a final ranked answer list. Our combination strategies make use of candidate answer confidences and use either a machine learning-based approach for parallel decomposition or a heuristic selection strategy for nested decomposition to do the final ranking.

The parallel decomposition components produce multiple subquestions that are posed to the underlying QA system, whereas the nested decomposition components generate pairs of inner/outer questions that are submitted to the QA system in sequence. Note the feedback loop in the nested decomposition pipeline since the answer to the inner question needs to be substituted in the outer.

Solving parallel decomposable questions

In this section, we describe our algorithm to solve parallel decomposable questions. The algorithm has three main parts: 1) recognize the presence of independent subquestions using syntax-based patterns; 2) rewrite subquestions to insert context; and 3) synthesize answers from independent facts.

Recognizing independent subquestions

Identifying segments of a question that portray independent facts about the correct answer is not trivial. Independent facts about the correct answer can be "woven" into a single complex question in a variety of ways. For example, a modifier of the focus of the question[2] may describe a fact about the answer, or a fact may be embedded within a subordinate clause in the question or even in an attached prepositional phrase. On the basis of an analysis of complex decomposable questions, we have identified some syntactic cues that are reliable indicators for decomposition. We then developed a set of patterns for question decomposition that require fine-grained lexico-syntactic information and are expressed over the full predicate-argument structure (PAS) derived from a syntactic parse of the question [2]. We identify three major

[2]The *focus* is the text span in the question that acts as a placeholder for the answer, e.g., in the question "This Asian country got its independence in 1898", the focus is "This Asian country".

types of patterns, targeting parallel configurations as described below.

1. *Independent subtrees*—One strategy for identifying potentially independent subquestions within a question is to look for clauses that are likely to capture a separate piece of information about the answer from the rest of the question. Relative or subordinate clauses are examples of such potentially independent subtrees and are typically indicative of parallel decomposition, as shown in the example below:

   (6) FICTIONAL ANIMALS: The name of this character, introduced in 1894, comes from the Hindi for "bear".

   We use syntactic cues that indicate such clauses within the PAS of the question to identify these parallel decomposable questions. Labels of edges in the PAS that connect such subtrees to the focus are generally good indicators of a subquestion and are used to identify the subtree as a decomposable fact. In example (6), the patterns identify "this character, first introduced in 1894" as a decomposed subquestion. Another syntactic cue involves patterns using conjunctions as decomposition points, as shown in example (2) mentioned earlier.
2. *Composable units*—An alternate strategy for identifying subquestions is to "compose" the fact by combining elements from the question. In contrast to the previous type of patterns where we try to find a portion of the PAS that can be "broken off" as an independent subquestion, the "composable units" patterns take different parts of the PAS and combine them to form a subquestion. For example, from question (7),

   (7) THE CABINET: James Wilson of Iowa, who headed this Department for 16 years, served longer than any other cabinet officer,

   we derive a subquestion "James Wilson of Iowa headed this Department" by composing the elements in subject, verb, and object positions of the PAS, respectively.
3. *Segments with qualifiers*—We employ a separate group of patterns for cases where the modifier of the focus is a relative qualifier, such as *the first*, *only*, and *the westernmost*. In such cases, information from another clause is usually required to "complete" the relative qualifier. Without it, the statement or fact has incomplete information (e.g., "the third man") and needs a supporting clause (e.g., "the third man ... to climb Mt. Everest") for the decomposition to make sense as a fact about the correct answer. Because it occurs often enough in the Jeopardy! data set, we have a separate category for such patterns. To deal with these cases, we created patterns that combine the characteristics of composable unit patterns with the independent subtree patterns. We "compose" the relative qualifier, the focus (along with its modifiers), and the attached supporting clause subtree to instantiate this type of pattern.

Table 1 Parallel decomposition rule coverage.

Decomposition pattern category	*Questions fired* (total: 1,269)
Independent subtrees	724
Composable units	415
Segments with qualifiers	33
Total questions found parallel	598

We defined rules to capture each of the patterns and applied our parallel decomposition rules to a blind set of 1,269 Final Jeopardy! questions to get an estimate of how often these patterns occur in the data. Results are shown in **Table 1**. The table also shows the total distinct questions found parallel decomposable as multiple rules can fire on the same question.

We did not evaluate the precision and recall for the parallel decomposition rules for three reasons:

- Manually creating a standard of parallel decomposable questions with their subparts is laborious, particularly because it requires judging whether facts are mutually independent or not.
- Even with such a standard, it is difficult to measure our rules by meaningfully comparing or aligning their output with the decompositions in the standard.
- We use a machine learning model (described later) to suitably combine and weigh the decomposition rules on the basis of their performance on the training set, thus implicitly taking into account their precision and recall (e.g., precise rules get a higher weight in the model).

Rewriting subquestions

Decomposition rules described in the previous subsection are applied in sequence to a given question to determine whether it can be decomposed. For example, given the question:

(8) HISTORIC PEOPLE: The life story of this man who died in 1801 was chronicled in an A&E Biography DVD titled "Triumph and Treason".

The system decomposes the question into two subquestions:

Q1: This man who died in 1801.
Q2: The life story of this man was chronicled

in an A&E Biography DVD titled "Triumph and Treason".

Since this is a case of parallel decomposition, the goal is to solve the original question Q by solving for subquestions Q1 and Q2 independently and combining evidence for the same answer from the subquestions as reinforcement. In the case of nested decomposition, we would first solve the inner subquestion and substitute its answer into the outer subquestion to arrive at the final answer to the original question.

After identifying the subquestions, the naive approach would be to submit them as they are to the QA system. However, we note that there are several problems with this approach. The subquestions are often much shorter than the original question and, in many cases, no longer have a unique answer. Moreover, the complement of the subquestion in the original question may be a relevant contextual cue for identifying the correct answer. In the example above, Q1 suffers from this problem; it no longer has a unique answer, and in this case, the system does not produce the correct answer in the candidate answer list for Q1 because of the lack of contextual information.

To resolve this issue, we insert contextual information into the subquestions. We do this in two steps. First, given a subquestion Q_i, we obtain the set of all temporal expressions (times and dates) and named entities (identified by our named entity recognizer) in the original question text that are not present in Q_i. We then insert these keywords into the category or topic of the original question and use this modified category for the subquestion Q_i when querying the QA system.

The rationale for this approach is the following: When our QA system receives a category or question pair, it treats information in the category differently than that in the question. The category provides the theme (or context) for the question, with keywords in the category sometimes becoming query terms when searching the corpus. In addition, when evaluating supporting evidence for a candidate answer, several answer scorers produce separate match scores for the category and the question. As a result, the machine learning model used to determine the rank and confidence of candidate answers in the base system weighs information in the category differently (typically less) than that in the question. Our rewriting approach takes advantage of this differential weighting of information to ensure that the larger context of the original question is still taken into account when evaluating a subquestion, although with less weight. As a result, the following two contextualized subquestions are generated for our example:

Q1: HISTORY PEOPLE (A&E Biography DVD "Triumph and Treason"): This man who died in 1801.

Q2: HISTORY PEOPLE (1801): The life story of this man was chronicled in an A&E Biography DVD titled "Triumph and Treason".

Note that when inserting context keywords into the category, we add them in parentheses at the end. This is to ensure a clear separation between the original category and the bag-of-words context added to it, as most parsers typically treat parenthesized phrases as separate clauses. This also ensures that analytics, which may rely on information in the category—such as answer type detection—are not affected by this modification.

Synthesizing answers from multiple subquestions

In the case of parallel decomposition, once the system has decomposed a question into multiple subquestions, the underlying (base) QA system is deployed to produce a ranked answer list for each subquestion. These answer lists must be combined and ranked to come up with a final ranking to the original question.

One simple way to produce a final score for each candidate answer is to apply the independence assumption and take the product of the scores returned by the base QA system for each subquestion. A key shortcoming to this approach is that our subquestions are not truly independent because of the contextualization procedure adopted by our question rewriters. Furthermore, the subquestions are generated by decomposition rules that have varying precision and recall and, therefore, should not be equally weighed. Finally, since our decomposition rules tend to overgenerate, in some cases, the answer to the original question considered as a whole is our preferred answer. For these reasons, we use a machine learning model to combine information across original and subquestion answer scores using features to capture the above information.

The model is trained over the following features:

- A binary feature signaling whether the candidate was a top answer to the nondecomposed (original) question.
- Confidence for the candidate answer to the nondecomposed question.
- Number of subquestions that have the candidate answer in the top 10.
- Features corresponding to the patterns used in parallel decomposition, with each feature taking a numeric value equal to the confidence of the base QA system on a fact identified by the corresponding pattern.

In case the candidate answer is not returned in the answer list of the original question or any of the decomposed subquestions, the corresponding feature value is set to *missing*. In addition, if a particular rule produces more than one subquestion, we set the corresponding rule feature value

Table 2 Evaluating parallel decomposition.

QA system	*No. of correct answers (end-to-end accuracy)*	*Accuracy on decomposable questions*
Baseline	635 (50.05%)	339 (56.68%)
Parallel decomposition without question rewriting (using ML)	634 (49.96%)	338 (56.52%)
Parallel decomposition with question rewriting (using heuristic re-ranking)	638 (50.27%)	342 (57.19%)
Parallel decomposition with question rewriting (using ML re-ranking)	643 (50.66%)	347 (58.02%)

for the candidate answer to the sum of the confidences obtained for that answer across all the subquestions. For the machine learning algorithm, we use the Weka [3] implementation of logistic regression with instance weighting.

Evaluating parallel decomposition

Evaluation data

Our data set is a collection of Final Jeopardy! question–answer pairs obtained from the J! Archive website (http://www.j-archive.com/). This gives us ground truth for both training a system and evaluating its performance. We split our set of roughly 3,000 Final Jeopardy! questions into a training set of 1,138 questions, a development set of 517 questions, and a blind test set of 1,269 questions.

Experiments

The decomposition rules were defined and tuned during the development. The final re-ranking model was trained on the training set using the features described in the previous subsection. The model was trained using logistic regression with instance weighting, where positive instances are weighed four times that of negative instances to address the significant imbalance of instances in the two classes. The results of applying the parallel decomposition rules followed by the re-ranking model to the 1,269 blind test questions are shown in the last row of **Table 2**. The baseline is the performance of the underlying QA (Watson) system used in our meta-framework without the decomposition components or analysis.

We further evaluate the importance of the contextualization aspect of our question rewriting technique. For this purpose, we altered our algorithm to issue the subquestion text as is, using the original category, and retrained and evaluated the resultant model again on our test set. Results are in the second row of Table 2.

Finally, we also wanted to compare our machine learning-based re-ranking model with a simple heuristic strategy that re-ranks answers based on a new final score computed as the sum of the confidences for the answer across all subquestions, including the original answer confidence for the entire question. Results are in the third row of Table 2.

Discussion of results

Table 2 shows the end-to-end accuracy of the baseline system against different configurations of the parallel decomposition answering extension over two sets of questions, i.e., the entire blind test set (1,269 questions) and a subset of this identified as parallel decomposable by our patterns (598 out of 1,269 questions, i.e., 47%). Interestingly, the performance of the baseline QA system on the parallel decomposable subset was 56.6%, 6% higher than the performance on all test questions. This is because questions in the parallel decomposable class typically contain more than one fact or constraint that the answer must satisfy, and even without decomposition, the system can exploit this redundancy in some cases such as when one fact is strongly associated with the correct answer and there is evidence supporting this in the sources.

Our results also show that without rewriting the questions to include contextual information in the category for subquestions, the system performance did not improve much over the baseline. On the other hand, using rewriting to insert context, the parallel decomposition algorithm using a simple heuristic re-ranking approach started to show some gains over the baseline. Finally, the best result was obtained using a machine learning-based re-ranker with question rewriting, where the system gained 1.4% in accuracy (with 10 gains and 2 losses) on the decomposable question set, which translated to an end-to-end gain of 0.6%. This gain was statistically significant, with significance assessed for $p < .05$ using McNemar's test [4] with Yates' correction for continuity.

Solving nested decomposable questions

Currently, we use a single general-purpose detection rule to identify inner and outer subquestions. We solve the inner subquestion first to find a "missing link" answer and insert it into the outer question (based on its confidence) to obtain a new answer, which is compared with the original answer to determine the final ranking.

Detecting nested subquestions

Our rule for detecting inner subquestions looks for noun phrases with leading determiners in the parse of the question. For example, in the question below, noun phrases with leading determiners have been underlined.

> (9) TELEVISION: In *a 1983 movie* about *a kidnapping*, Daniel J. Travanti played *the man* who would later host this series.

Our motivation for this heuristic is that, typically, noun phrases of this type refer to some named entity, knowing which may help us find the answer to the entire question. In the above case, the phrase "1983 movie" refers to the movie "Adam", whereas the "the man" refers to John Walsh. We refer to such entities as "missing links", i.e., unseen entities strongly associated with both the correct answer and entities in the question. In [5], a technique is described to identify missing links on the basis of their semantic association with concepts in the question. In this paper, we adopt a different strategy for bridging across missing links on the basis of question decomposition.

Note that the rule described above to detect inner facts does not cover all types of nested questions. For example, it does not detect the nested fact (underlined) in the following question:

> (10) SCIENTISTS: *When Einstein won the Nobel Prize in Physics*, he was a naturalized citizen of this country.

Nevertheless, the current heuristic has fairly good coverage: In the 1,269 Final Jeopardy! questions that we tested, it fired on 727 questions with precision close to 90% (based on a manual evaluation of a random sample; unlike the parallel decomposition case, measuring precision of this rule is simpler by virtue of the algorithm, which seeks simple noun phrases with a specific shape). We are currently working on adding general-purpose nested decomposition patterns/rules (along the same lines as described for parallel decomposition) to capture inner facts such as in question (10).

Rewriting inner subquestions

Having found missing link noun phrases, the next step in our algorithm is to rewrite the original question, making each such noun phrase the new focus. For example, considering the noun phrase "a 1983 movie", we would rewrite the original question (9) as follows:

> Rewritten clue: In **this** 1983 movie about a kidnapping, Daniel J. Travanti played the man who would later host **a** series.

We follow this simple rewriting approach for all the nested subquestions and create a set of new questions, each of which explicitly asks about a missing link. The notion is to issue these questions to the entire QA system (treated as a black box) in order to find the missing links. This rewriting approach is different from that followed for parallel decomposition where the subquestions (facts) need to be independently verified. In the nested case, there exists strong dependence between the inner and outer subquestions (and issuing the missing link noun phrase by itself is not a meaningful inner question).

One key property of our underlying QA system that we exploit here is its ability to return reliable confidences for its answers. This allows us to be more flexible and recall oriented in our strategy that searches for phrases implying missing links by considering any noun phrase with a leading determiner, since if the phrase does not actually refer to some named entity, it is very unlikely that our QA system will come back with an answer with high confidence for the rewritten question.

Considering example (9) above and the problematic phrase "a kidnapping", which does not actually refer to a named entity, rewriting the question along the lines mentioned produces the following:

> Rewritten Clue: In a 1983 movie about **this** kidnapping, Daniel J. Travanti played the man who would later host **a** series.

When our QA system is asked this question, it returns "children" as its top answer with a confidence of 0.1, which makes sense given that the answer type "kidnapping" is not a meaningful one, and most of the type coercion answer scores [6] fail to recognize any candidate answer being of this type. Thus, we use the confidence of the system in its answer for the rewritten question to determine whether it has found a meaningful missing link or not.

Performance optimization

As example (9) illustrates, we may detect several nested facts from a single question, which translates to a set of new questions to ask the QA system. This may be an issue when playing Jeopardy! because of the extremely tight time constraints. To deal with this, we need some way to rank the new questions and issue only the top N questions in the time permitted.

Table 3 Evaluating nested decomposition.

QA system	*No. of correct answers (end-to-end accuracy)*	*Accuracy on decomposable questions*
Baseline	635 (50.05%)	345 (47.45%)
Nested decomposition with heuristic re-ranker	645 (50.82%)	355 (48.83%)

Note that when we detect a missing link noun phrase and transform the question to make this phrase the new focus, the lexical answer type (LAT) for this rewritten question is the headword of the noun phrase. In the example above, the LATs for each of the new questions are "movie", "man", and "kidnapping", respectively. Prior frequency information for LATs along with type instance coverage can be used to rank the new questions: LATs that are more frequently asked about (based on a history of prior questions) such as "movie" or "man" are probably more likely to refer to a missing link named entity than a LAT such as "kidnapping". Similarly, LATs such as "movie" and "man" appear more frequently as types (or classes) in ontologies and structured data sources such as DBpedia (http://dbpedia.org) than a LAT such as "kidnapping" and are thus more likely to form meaningful questions. Using this type-likelihood information, we are able to rank the missing link-detection questions and can issue as many questions as time permits.

Plugging nested-fact answers in the outer question

Having issued a nested subquestion to the QA system and obtained a missing link candidate answer, the next step is to substitute the missing link into the original question and query again for the correct answer. Inserting the missing link candidate in place of the noun phrase that implied it is relatively straightforward and is done using the parse of the sentence so that the new question is still well formed.

In the earlier example, since our QA system correctly returns the missing link "Adam" for the 1983 movie about kidnapping, the final question with the missing link inserted becomes

> Final Outer Clue (with inner-answer added): In **Adam**, Daniel J. Travanti played the man who would later host this series.

Posing this question to our QA system returns the correct answer "America's Most Wanted" in the top position with a confidence of 0.96 (quite high, as we have a 0–1 scale).

Heuristic re-ranking strategy

In general, we need a selection strategy to decide between the answer obtained through nested missing link analysis and the original answer obtained for the entire question. The factors we take into account when making this decision are system confidence for the missing link candidate (for the inner question), confidence of the new answer obtained after inserting a potential missing link into the original question, and confidence of the original top answer for the entire question.

We make this decision using a simple and intuitive heuristic selection strategy: We compute the final score for a candidate answer as the product of the confidence for the answer to the inner question and that for the answer to the outer question. The rationale is based on the notion of conditional probability, as the probability of producing the correct answer by inserting the missing link is dependent on the probability of the missing link itself being correct. Comparing this score against the original top answer confidence or score allows for selecting the answer with the higher score as the final answer. In cases where we detect multiple nested facts in the original question (as shown in the earlier example) and generate multiple missing link candidates, we follow the above approach for all the inner facts and select the final answer with the highest overall score.

Evaluating nested decomposition

Experiment

We ran a similar evaluation for nested decomposable questions, as done for parallel decomposition, using the same evaluation data of 1,269 Final Jeopardy! questions (as a blind test set) and compared the baseline (Watson) QA to the heuristic re-ranking approach described above. The single decomposition pattern (noun phrase with a leading determiner) fired on more than half of the entire blind test set (727 out of 1,269 questions, i.e., 57%). Results of the experiment are presented in **Table 3**, which shows end-to-end accuracy of the baseline over the test set and over the nested decomposable—according to our nested decomposable detection algorithm—subset of that. The table further compares the baseline with the nested decomposition answering extension described above.

Discussion of results

Here, the performance of the baseline QA system over the nested decomposable subset was slightly worse than the

overall performance (and much lower than the parallel decomposable cases). The likely explanation is that nested questions require solving for an inner fact first, the answer to which often provides necessary missing information to find the correct answer; this dependence on sequencing makes nested questions more difficult to solve than parallel decomposable questions, which contain multiple independent facts, and is in line with our hypothesis linking system performance with the ability to find answers to subquestions separately.

Our nested decomposition algorithm using the heuristic re-ranking approach was able to achieve a gain of 1.4% (with 13 gains and 3 losses, i.e., net of 10 gains) over the baseline on the decomposable question set, which translated to an end-to-end gain close to 0.8%. As in the case of parallel decomposition, these gains are statistically significant, with significance assessed for $p < .05$ using McNemar's test.

Joint impact of parallel and nested decomposition

The overall impact of using both parallel and nested decomposition was a 1.4% gain (23 gains and 5 losses) on end-to-end system accuracy, which was also statistically significant by McNemar's test (with a two-tailed p of .0013).

A key point to keep in mind when evaluating these results is that our baseline Watson system represents the state of the art in solving Jeopardy! questions. Furthermore, our experiment data consists of Final Jeopardy! questions, which are known to be more difficult than regular Jeopardy! questions. On the basis of player performance statistics obtained from J! Archive, we estimate qualified Jeopardy! players' accuracy to be 48% on Final Jeopardy!, whereas the baseline Watson has an accuracy value close to 51% on blind data. Hence, a gain of nearly 1.5% end to end on such questions can be considered a strong improvement.

Related work

In work on decomposition to date, a question is typically considered "complex"—and therefore decomposable—if it requires nonfactoid answers, e.g., multiple sentences or summaries of answers [7], connected paragraphs [8], explanations and/or justification of an answer [1], lists [9] or lists of sets [10], and so forth. The decomposition approaches typically defer to discourse and/or semantic properties of the question, additionally deploying complex processes such as lexical adjustment [7], question refocusing [1, 9], or deep temporal/spatial analysis [9, 11]. Largely, decomposition work has been driven by the needs of answering "beyond factoid" questions. In contrast, we focus here on decomposition applied to improving the quality of QA of a broad, i.e., open range, set of *factoid* questions.

In the literature, we find descriptions of strategies such as *local decomposition* and *meronymy decomposition* [9], semantic decomposition using *knowledge templates* [1], and *textual entailment* [7] to connect, through semantics and discourse, the original question with its numerous decompositions. Without implementation details, we have to assume that such notions of decomposition require access to fairly well circumscribed domain models, not an assumption we can make for an open-domain QA application. Indeed, this is why we constrain our decomposition algorithms to use only materials explicitly present in the question.

Some of the decomposition work to date does also look at factoid questions. The syntactic decomposition described in [1] is motivated by the observation that for any given question, there may be more than one "fact" (*ternary expression* or *relational triple*) pertaining to the answer; decomposition is particularly essential to the QA process when facts from the same question are associated with different knowledge resources so that different triples can be routed to appropriate resources. Direct comparison of syntactic decomposition rule sets is not possible, but it is likely that there are both overlaps and divergences between the rules deployed in [1] and the ones outlined in this paper; it is worth noting, however, that [1] makes no distinction between parallel and nested decomposition. The major difference in the two approaches is our use of a machine learning model to guide the final combination and ranking of candidate answers returned to possibly multiple decompositions into subquestions.

Another common trend in factoid question decomposition is, in particular, *temporal decomposition* [9–11], where the question is rephrased by plugging the value of a temporal expression in place of the expression itself. For us, this is a special case of nested decomposition. In any case, such systems do not provide a general solution for decomposition.

The closest similarity our fact-based decomposition has with an established approach is with the notion of asking additional questions in order to derive constraints on candidate answers, developed by [12]. However, additional questions are generated through knowledge of the domain. As we emphasized earlier, since we cannot appeal to a domain, we use the question context alone in generating queryable constraints.

In the Jeopardy! data, there are questions other than Final Jeopardy! that require and benefit from some kinds of decomposition. Most prominently, these are categorized as Definition questions and some kinds of Puzzle questions [13]. Decomposition in such kinds of special questions is more predictable than in regular factoids and thus is more systematically recognizable. At the same time, the special nature of the questions suggests that solution strategies need not follow the general approach we describe in

this paper. Such specialized decomposition strategies are described in [14].

Conclusion

In this paper, we have described a general-purpose question decomposition framework that identifies both independent and nested facts within complex questions. The framework can extend any QA system that provides answers with confidences for factoid questions and that considers the context or topic of the question separate from its main content. We demonstrated the effectiveness of this decomposition framework by using our state-of-the-art factoid QA system, Watson, and improving its end-to-end accuracy by nearly 1.5% on a blind test set of Final Jeopardy! questions and found the gains to be statistically significant. We expect this analysis to help in several real-world QA applications.

*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., in the United States, other countries, or both.

References

1. B. Katz, G. Borchardt, and S. Felshin, "Syntactic and semantic decomposition strategies for question answering from multiple sources," in *Proc. AAAI Workshop on Inference in Textual Question Answering*, 2005, pp. 35–41.
2. 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.
3. I. Witten and E. Frank, *Data Mining—Practical Machine Learning Tools and Techniques with Java Implementations*. San Francisco, CA: Morgan Kaufmann, 2000.
4. 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.
5. 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.
6. 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.
7. F. Lacatusu, A. Hickl, and S. Harabagiu, "The impact of question decomposition on the quality of answer summaries," in *Proc. 5th LREC*, Genoa, Italy, 2006. [Online]. Available: http://www.languagecomputer.com/papers/lacatusuEtAl2006lrec.pdf
8. R. Soricut and E. Brill, "Automatic question answering: Beyond the factoid," in *Main Proc. HLT-NAACL*, 2004, pp. 57–64. [Online]. Available: http://acl.ldc.upenn.edu/hlt-naacl2004/main/pdf/104_Paper.pdf
9. S. Hartrumpf, "Semantic decomposition for question answering," in *Proc. 18th ECAI*, G. Malik, C. D. Spyropoulos, N. Fakotakis, and N. Avouris, Eds., 2008, pp. 313–317.
10. C. J. Lin and R. R. Liu, "An analysis of multi-focus questions," in *Proc. SIGIR Workshop Focused Retrieval*, 2008, pp. 30–36.
11. E. Saquete, P. Martinez Barco, R. Munoz, and J. Vicedo, "Splitting complex temporal questions for question answering systems," in *42nd Annu. Meeting ACL*, 2004, p. 566.
12. J. Prager, J. Chu-Carroll, and K. Czuba, "Question answering by constraint satisfaction: QA-by-dossier with constraints," in *Proc. ACL*, 2004, pp. 574–581.
13. 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.
14. J. M. Prager, E. W. Brown, and J. Chu-Carroll, "Special questions and techniques," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 11, pp. 11:1–11:13, May/Jul. 2012.

Received July 18, 2011; accepted for publication December 15, 2011

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.

Siddharth Patwardhan *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (siddharth@us.ibm.com)*. Dr. Patwardhan is a Post-Doctoral Researcher in the Knowledge Structures Group at the T. J. Watson Research Center. He received a B.E. degree in computer engineering from the University of Pune in 2001, an M.S. degree in computer science from the University of Minnesota in 2003, and a Ph.D. degree in computer science from the University of Utah in 2010. He has been working at the IBM T. J. Watson Research Center since 2009, exploring research projects in natural-language processing and artificial intelligence. He is a member of the Algorithms Team working on the IBM Jeopardy! challenge and is an author or coauthor of more than 25 technical publications covering his work on information extraction, opinion/sentiment analysis, computational lexical semantics, and question answering. Dr. Patwardhan is a member of the Association for Computational Linguistics and a member of the Association for the Advancement of Artificial Intelligence.

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.

Adam Lally *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (alally@us.ibm.com).*
Mr. Lally is a Senior Technical Staff Member at the IBM T. J. Watson Research Center. He received a B.S. degree in computer science from Rensselaer Polytechnic Institute in 1998 and an M.S. degree in computer science from Columbia University in 2006. As a member of the IBM DeepQA Algorithms Team, he helped develop the Watson system architecture that gave the machine its speed. He also worked on the natural-language processing algorithms that enable Watson to understand questions and categories and gather and assess evidence in natural language. Before working on Watson, he was the lead software engineer for the Unstructured Information Management Architecture project, an open-source platform for creating, integrating, and deploying unstructured information management solutions.

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 a 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.