Morpheus: A Deep Web Question Answering System

Christian Grant, Clint P. George, Joir-dan Gumbs, Joseph N. Wilson, Peter J. Dobbins
University of Florida, Dept. of Computer Science
Gainesville, Florida, USA
{cgrant, cgeorge, jgumbs, jnw, pjd} @cise.ufl.edu

ABSTRACT
When users search the deep web, the essence of their search is often found in a previously answered query. The Morpheus question answering system reuses prior searches to answer similar user queries. Queries are represented in a semi-structured format that contains query terms and referenced classes within a specific ontology. Morpheus answers questions by using methods from prior successful searches. The system ranks stored methods based on a similarity quasi-metric defined on assigned classes of queries. Similarity depends on the class heterarchy in an ontology and its associated text corpora. Morpheus revisits the prior search pathways of the stored searches to construct possible answers. Realm-based ontologies are created using Wikipedia pages, associated categories, and the synset heterarchy of WordNet. This paper describes the entire process with emphasis on the matching of user queries to stored answering methods.

1. INTRODUCTION
When traveling through a jungle to a destination, it is easy to get lost. The first person to journey somewhere may make a number of mistakes when trying to find the best path to their destination. Those who come later find it easier to reach the destination if a well-marked trail has been created. Olsen and Malizia describe this idea as exploiting trails [8]. Rather than treating a user’s discovery experience as a unique entity, one can exploit the fact that a similar search may have already been performed. In one study, almost 40 percent of web queries were repetitions of previous queries [12]. Thus, reuse of prior searches is one way to optimize the search process. Morpheus is a question answering system motivated by reuse of prior web search pathways to yield an answer to a user query. Morpheus follows path finders to their destinations and not only marks the trail, but also provides a taxi service to take followers to similar destinations.

Morpheus focuses on the deep (or hidden) web to answer questions because of the large stores of quality information provided by the databases that support it [7]. Web forms act as an interface to this information. Morpheus employs user exploration through these web forms to learn the types of data each deep web location provides.

There are two distinct Morpheus user roles. A path finder enters queries in the Morpheus web interface and searches for an answer to the query using an instrumented web browser. This web tracking tool stores the query and necessary information to revisit the pathways to the page where the path finder found the answer. A path follower uses the Morpheus system much like a regular search engine with a natural language interface. The path follower enters a question in a text box and receives a guided path to the answer. The system exploits previously found paths to provide an answer.

Morpheus represents a user question as a semi-structured query (SSQ). It assumes the query terms belong to classes of a consistent realm-based ontology, that is, one having a singly rooted heterarchy whose subclass/superclass relations have meaningful semantic interpretations. When a path follower enters a query, Morpheus ranks SSQs in the store based on class similarity. Suppose a path follower asks: A 1997 Toyota Camry V6 needs what size tires? In this query the classes associated with terms, e.g. Manufacturer with Toyota, help us identify similar queries.

This paper discusses related question answering and ontology generation systems in section 2. Section 3 explains the Morpheus system and its implementation. In Section 4 we describe the current results of our approach. Finally, we conclude with future goals for the system.

2. RELATED WORK

2.1 Question Answering Systems
The earliest question answering systems such as BASEBALL [6] and Lunar [13] had closed domains and closed corpora, that is, they support a finite amount of questions on corpora containing a fixed set of documents. Morpheus
uses the web as its dynamic, open corpus and examines deep web sources to answer questions. This process is federated question answering.

Several other QA systems that use the web as a resource have been developed. Example systems include START\(^1\) and Swingly\(^2\). These systems use web pages from searches with web crawlers or search engines to find answers. Morpheus differs in that it seeks out relevant deep web sources, and instead of using a web search engine, it uses only the pages referenced in a previously answered question.

### 2.2 Ontology generators

The DBpedia\(^3\) project is a community of contributors extracting semantic information from Wikipedia and making this information available on the Web. Wikipedia semantics includes disambiguation pages, geo-coordinates, categorization information, images, info-box templates, links to external web pages, and redirects to pages in Wiki markup form [3]. DBpedia does not define any new relations between the final web pages, and redirects to pages in Wiki markup form [3].

YAGO is a semi-automatically constructed ontology obtained from the Wikipedia pages, info-boxes, categories, and WordNet\(^4\) synsets heterarchy [11]. YAGO uses the Wikipedia page titles as its ontology individuals and categories as its ontology classes. YAGO uses only the nouns from WordNet and ignores the WordNet verbs and adjectives. YAGO discovers connections between WordNet synsets and Wikipedia categories, parsing the category names and matching the parsed category components with the WordNet synsets. Each Wikipedia category not having a WordNet match is ignored in the YAGO ontology. The ontology’s heterarchy is built using the hypernym and hyponym relations of the WordNet synsets.

We use YAGO’s principles to construct ontologies that provide similarity measures for answering questions within the same domain. Thus far, these ontologies can be used to classify terms, however their classes do not always appropriately categorize query parameters. It is necessary to provide an appropriate level of class granularity. Section 3 discusses our approach for identifying classes and their instances from deep web forms and documents.

### 3. SYSTEM ARCHITECTURE

This section presents ontology and corpora, query processing, ranking queries, and query executing.

#### 3.1 Using Ontology and Corpora

Morpheus uses an ontology that contains classes of a particular realm of interest. Each leaf node in the ontology is associated with a corpus of words belonging to a class. For example, we have constructed a vehicular ontology containing classes relevant to the vehicular realm. This ontology provides a structure for reference in the following sections.

Morpheus references the DBpedia categories, Wikipedia pages, and the WordNet synset heterarchy to find class-relevant web pages. First a realm is mapped to a DBpedia category [3]. Using the DBpedia ontology properties broader and narrower, a Markov Blanket [10] is created covering all neighboring categories.

To build a corpus for each of the leaf nodes in the ontology, we extract terms from the Wikipedia pages associated with the DBpedia categories found in its blanket. From this term corpus, we can find the likelihood of a term belonging to a particular class. This assists in classifying terms in a path follower query. The likelihood is determined by the probability of a class given a term using Bayes Rule (Eq. 1), since we can easily obtain the term-class and term-corpus probabilities as relative frequencies.

\[
P(\text{class}|\text{term}) = \frac{P(\text{term}|\text{class})P(\text{class})}{P(\text{term})} \tag{1}
\]

In addition, we employ Latent Dirichlet Allocation (LDA) to identify latent topics of the documents in a corpus [4]. LDA is Bayesian model that represents a document in the corpus by distributions over topics, and a topic itself is a distribution over all terms in the corpus. For example, the latent topics reflect the thematic structure of Wikipedia pages. Thus, LDA discovers relevant topic proportions in a document using posterior inference [4]. Given a text document, we tag related documents by matching their similarity over the estimated topic proportions, assisting in ontology and corpora construction. We use LDA as a dimensionality reduction tool. LDA’s topic mixture are represented as feature vectors for each document. We are evaluating support vector machines as a classifier over the documents-topic proportions. Due to its fully generative semantics, this usage of LDA could address drawbacks of frequency based approaches (e.g. TF-IDF, LSI, and pLSI) such as dimensionality and failure to find the discriminative set of words for a document.

#### 3.2 Recording

The Query Resolution Recorder (QRR) is an instrumented web browser that records the interactions of a path finder answering a question. The path finder also uses the tool to identify ontological classes associated with search terms. Morpheus stores the query, its terms, and its classes as an SSQ. Table 1 is an example showing the SSQ model of the query: A 1997 Toyota Camry V6 needs what size tires? The SSQ in Table 1 is said to be qualified because the classes associated with its terms have been identified. Using the QRR, the path finder is also able to identify where answers can be found within traversed pages.

![Table 1: Example SSQ model](image)

The Query Resolution Method (QRM) is a data structure that models the question answering process. A QRM represents a generalized executable realization of the search process that the path finder followed. The QRM is able to reconstruct the page search path followed by the path finder. Each QRM contains a realm from our ontology, an SSQ, and information to support the query answering process. For each dynamic page, the QRM contains a list of inputs and reference outputs from the URL.

When a path follower submits a query the Morpheus search process parses and tags queries in order to record important

---

\(^1\)http://start.csail.mit.edu
\(^2\)http://swingly.com
\(^3\)http://dbpedia.org
\(^4\)http://wordnet.princeton.edu
terms. The system assigns the most probable realm given the terms in the query as calculated from realm-specific corpora. Once the realm is assigned, an ontology search is performed to assign classes to the terms. An SSQ is constructed and the system attempts to match this new SSQ to existing QRMs. Rather than matching exact query terms, the system matches input and output classes, because a QRM can potentially answer many similar queries.

### 3.3 Ranking

To answer a user’s query, a candidate SSQ, Morpheus finds similar qualified SSQs that are associated with QRMs in the Morpheus data store. To determine SSQ similarity, we consider the SSQ’s realm, input terms, output terms, and their assigned classes.

The class divergence of two classes within the ontology characterizes their dissimilarity. This solution is motivated by the concept of multiple dispatch in CLOS and Dylan programming for generic function type matches [2].

We consider the class match as a type match and we use class divergence to calculate the relevance between the candidate SSQ and a qualified SSQ. Each qualified SSQ will have input terms, output terms, associated classes, and one realm from the QRM. For the candidate SSQ, the relevant classes for terms are determined from the natural language processing engine and corpora. The calculation of a candidate classes for terms are determined from the natural language processing engine and corpora. The calculation of a realm from the QRM. For the candidate SSQ, the relevant classes for terms are determined from the natural language processing engine and corpora. The calculation of a realm from the QRM is performed using the terms found within the query and any probabilities found with \( p(\text{realm}|\text{term}) \). We match QRMs that belong to the same realm of the candidate SSQ. The relevance of a qualified SSQ to the candidate SSQ is determined by aggregating the divergence measure of input term classes associated with each SSQ. In addition, we order QRMs in the data store by decreasing relevance. The order provides a ranking for the results to the user. The following describes class divergence in detail.

We define class divergence (Eq. 2), a quasi-metric, between a source class and a target class using the topological structure of the classes in an ontology. We write \( S \prec T \) for the reflexive transitive closure of the superclass relation. Let \( d(P, Q) \) represent the hop distance in the directed ontology inheritance graph from \( P \) to \( Q \). The class divergence between two source and target class ranges from zero (for identical classes) to one (for type incompatible classes). Let \( S \), \( T \), \( C \) be the source class, target class, and \( C \) be a least common ancestor class of \( S \) and \( T \) i.e., one that minimizes \( d(S, C) + d(T, C) \). The class divergence between \( S \) and \( T \) is defined by:

\[
\begin{align*}
d_{\text{cd}}(S, T) &= 0 & \text{if } S \text{ Uri} \equiv T \text{ Uri} \\
d(S, T)/(3h) & \quad S \prec T \\
1 & \quad T \prec S \\
(d(S, \text{root}) + d(S, C))/(3h) & \quad \text{otherwise}
\end{align*}
\]

where \( h \) is the longest path in the ontology class hierarchy.

Note, if \( S \prec T \) and \( S \not\prec Q \) then \( d(S, T) < d(S, Q) \), that is, the divergence of a source class to a target ancestor class is smaller than the divergence of a source class to any class that is not an ancestor. This is an important property in determining the compatibility of classes for answering queries. If an SSQ answers queries concerning an ancestor class, it is more relevant than an SSQ that answers queries from any non-ancestral class.

Suppose we want to find the class divergence between \( \text{Bus} \) and \( \text{Sedan} \) from the ontology shown in Figure 1. \( \text{LandVehicle} \) is their least common ancestor because \( \text{Sedan} \) is a subclass of \( \text{Automobile} \), which is a subclass of \( \text{LandVehicle} \), and \( \text{Bus} \) is a subclass of \( \text{LandVehicle} \).

The longest path from \( \text{Bus} \) and \( \text{sedan} \) to the tree root is four (\( h = 4 \)). By formula 2, \( d(\text{Bus}, \text{Sedan}) = (d(\text{Bus}, \text{Root}) + d(\text{Bus}, \text{LandVehicle}) + d(\text{Sedan}, \text{LandVehicle}))/3(4) \) thus \((3 + 1 + 2)/(3 * 4) = 6/12\).

### 3.4 Executing

Once we have ranked the QRMs for a given user query, we can produce answers by re-visiting the pathways stored in the QRMs. The Morpheus Query Executor (QE) evaluates a script of the query resolving process. It simulates a human clicking buttons to follow links, submit forms, and highlight data, forming a textual answer. The QE assumes that because of the auto generated nature of deep web pages, the location of answers are the same irrespective of page changes. It uses the relative XPath location to the answer node on HTML pages as described in [1].

### 4. RESULTS

First, we built an ontology for the vehicular realm exploiting the Wikipedia pages, DBpedia categories, and WordNet synsets. For each of the classes in the ontology we built corpora from the corresponding pWikipedia pages. Figure 1 shows a subsection of this ontology.

In Table 2 we show the data output by the Morpheus parse of the query. It extracts the \( \text{wh-} \)term that classifies the sentence as a question, identifies the answer class, and locates descriptive phrases to produce the answer. Finally, the engine produces n-grams from phrases in the descriptive information sections.

Using the data in Table 2 we determine relevant classes in non-increasing order of relevance. Table 3 shows the eight best term classes and their probabilities for automotive queries.
what size tires for a Toyota Camry V6?

Table 2: The output of NLP engine

| Term                  | Class      | P(Class|Term) |
|-----------------------|------------|---------|
| 1997 Toyota           | Engines    | 0.72    |
| Toyota Camry          | Sedans     | 0.74    |
| Toyota Camry V6       | Coupes     | 0.74    |
| Camry                 | Sedans     | 0.74    |
| Camry V6              | Coupes     | 0.74    |
| V6                    | Sedans     | 0.74    |

Table 3: Term classes and probabilities

<table>
<thead>
<tr>
<th>Query</th>
<th>Tagged Classes</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 1997 Toyota Camry V6 needs what size tires?</td>
<td>Sedan, Automobile, Engine, Manufacturer</td>
<td>0.91</td>
</tr>
<tr>
<td>What is the tire size for a 1998 Sienna XLE Van?</td>
<td>Van, Manufacturer</td>
<td>0.72</td>
</tr>
<tr>
<td>Where can I buy an engine for a Toyota Camry V6?</td>
<td>Sedan, Automobile Engine, Manufacturers</td>
<td>0.74</td>
</tr>
</tbody>
</table>

Table 4: Highest ranked queries

We found the best classes for the terms in the candidate SSQ. We calculated the class divergence between these classes and the qualified SSQ classes in the QRM store. QRMs are ranked based upon the relevance score and the class divergence measure. Table 4 shows the thAnswers are produced by the QRE, a python back end.ree highest ranked queries. Finally, we execute the best QRMs and display the results to the user.

5. CONCLUSION

In this paper, we propose a novel question answering system that uses the deep web and previously answered user queries to answer similar questions. The system uses a path finder to annotate answer paths so path followers can discover answers to similar questions. Each (question, answer path) pair is assigned a realm, and new questions are matched to existing (question, answer path) pairs. The classification of new question terms into classes is based on term frequency distributions in our realm specific corpora of web documents. These terms are the input to existing answer paths and we re-execute these paths with the new input to produce answers.

Our solution is composed a web front end where users can ask questions. The QRR was developed as a Firefox plugin and an associated C# application. Our similarity measures were coded using Java and open source libraries. Answers are produced by the QRE, a python back end. The data is stored in a PostresSQL database.

Topic modeling provides a promising approach to identifying pages relevant to a class in a more automated manner. We believe our web form entry annotation methods and form label extraction [7] can yield promising results. Combining this with the method of Elmeleegy et al. [5] may remove the user from the answer path generation process.

Additionally, we are investigating methods of merging QRMs to answer compound questions. This will allow us to chain QRMs using the principles of transform composition [9].

6. ACKNOWLEDGMENTS

We thank Guillermo Cordero, Benjamin Landers, Patrick Meyer, Christopher Shields, and Terence Tai for their assistance building this system. This material is based upon work supported by the National Science Foundation under Grant No. 0712799 and a Graduate Research Fellowship to Christian Grant.

7. REFERENCES