Semantic Parsing of Technical Support Questions

Abhirut Gupta, Anupama Ray, Gargi Dasgupta,
Gautam Singh, **Pooja Aggarwal** and **Prateeti Mohapatra**
IBM Research AI
{abhirutgupta, anupamar, gaargidasgupta,
gautamsi, aggarwal.pooja, pramoh01}@in.ibm.com

Abstract

Technical support problems are very complex. In contrast to regular web queries (that contain few keywords) or factoid questions (which are a few sentences), these problems usually include attributes like a detailed description of what is failing (symptom), steps taken in an effort to remediate the failure (activity), and sometimes a specific request or ask (intent). Automating support is the task of automatically providing answers to these problems given a corpus of solution documents. Traditional approaches to this task rely on information retrieval and are keyword based; looking for keyword overlap between the question and solution documents and ignoring these attributes. We present an approach for semantic parsing of technical questions that uses grammatical structure to extract these attributes as a baseline, and a CRF based model that can improve performance considerably in the presence of annotated data for training. We also demonstrate that combined with reasoning, these attributes help outperform retrieval baselines.

1 Introduction

Technical support services involve enterprises providing after-sales support to users of technology products. Traditionally, this entails employing a large number of human agents to manually diagnose, troubleshoot, and resolve customer problems. However, with ever increasing variety of products and the exponential growth of users, this model doesn't scale. Costs of hiring and training additional talent to keep up with this growth is a huge overhead. There is a clear incentive to automate this process, reducing manual effort and resolution times. Automating technical support refers to the task of automatically understanding user questions, performing diagnosis and troubleshooting, either remotely or in conversation with the user, and providing resolutions. Given the massive potential impact, it is no surprise that the problem draws attention and investment from many large corporations as well as startups (Flinders, 2015), and (Dhoolia et al., 2017).

Automating technical support has a very broad goal, and involves solving various sub-problems like document representation (Yang et al., 2017), identifying and extracting procedures, log analysis, and goal based dialoging (Bordes and Weston, 2016). Each of these problems has a plethora of daunting research challenges that remain largely unsolved. In this paper, we restrict our focus to the problem of understanding questions in technical support and retrieving relevant resolution documents from a corpus of support content. While a complete automation system would also tackle the challenging problem of identifying the troubleshooting/diagnostic procedure from these documents, we assume that a returned "relevant" document contains procedures required for resolving the problem.

As identified in (Gupta et al., 2017) technical support problems are complex, and involve various attributes like failing entities, a detailed description of the problem (symptom), steps taken in an effort to remediate the failure (activity), and sometimes a specific request or ask (intent). Consider two actual support problems in Figure 1 from the software and the hardware domains. The hardware query comprises a problem description with the entity "tape drive", mentioning the symptom "stuck" and an unsuccessful

Proceedings of the 27th International Conference on Computational Linguistics, pages 3251–3259
Santa Fe, New Mexico, USA, August 20-26, 2018.

Table 1: Average length of questions (in words) from QA datasets.

Dataset	Average length
Wiki QA	7.32
TREC 2007 QA	8.05
DB2 Prop	150.97
DB2 Open	105.87

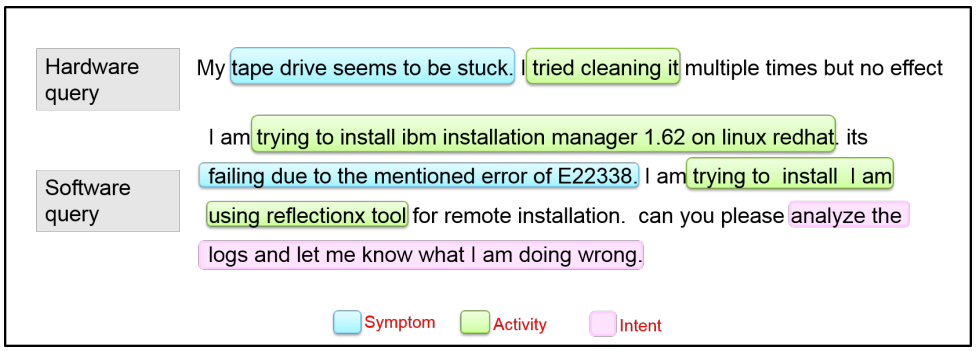

Figure 1: Illustration of Annotation of attributes on a Support ticket

attempt to mitigate the problem by "cleaning". The software question describes relevant details regarding the "failing installation". It also contains an explicit ask from the technician for "analysis of log files". State-of-the-art information retrieval based systems which depend on keywords do not perform well on such queries. For example, without an understanding of the hardware problem description and its attributes, the system might return pages which suggest cleaning the tape drive. For the software query, understanding the explicit ask of log analysis is key to a relevant response. This inherent complexity of questions in this domain differentiates it from traditional Question Answering or Information Retrieval problems. Table 1 compares the average length of questions in popular open QA datasets - WikiQA (Yang et al., 2015) and TREC 2007 QA (Dang et al., 2007), to two datasets in the domain of technical support that we curated. For effective retrieval of documents, it is essential that we extract these attributes (removing noise), and implement reasoning on top.

In this paper, we present a method for semantic parsing of support questions, to extract these attributes, which uses rules based on the grammatical structure of sentences. To the best of our knowledge, we are first to address the complexities of problems in technical support by identifying pertinent semantic attributes, extracting them, and proving their usefulness in retrieving relevant results.

In section 2 we present existing related work. In section 3, we present the details of our model for semantic parsing. The goal of extracting these attributes, however, is to improve quality of returned documents. To emphasize the importance of extracting these attributes, we show marked improvements in retrieval. The datasets used for experiments are discussed in 4 and evaluations are presented in section 5. We conclude the paper with a discussion of the learnings from evaluation and analysis, and directions for future research in section 6.

2 Related Work

The problem of answering natural language questions has received significant attention from the natural language processing research community. Dependency and simple phrase structure grammar models have been around for a while for understanding natural language text. The authors in (de Marneffe and Manning, 2008) parse grammatical relations from text for automated understanding. In Watson Jeopardy (McCord et al., 2012), authors use slot grammar parsing (McCord, 1990b) that decomposes the natural language text to slots, which is widely used for applications like question analysis (McCord et al., 2012), question decomposition, knowledge extraction for QA (Fan et al., 2012), relation extraction,

(Wang et al., 2012), candidate generation (Lally et al., 2012), and analysis and gathering of textual evidence (Murdock et al., 2012). Works focusing on understanding user intent use classifier models or dependency parsing schemes (Li and Roth, 2006; Metzler and Croft, 2005). Other recent works build neural models that represent a question as a continuous-valued vector (Chen et al., 2016). The key difference in these approaches and our problem setting is that they usually work on simpler, entity seeking questions. However questions in support are usually very complex, contain multiple attributes describing the situation that leads to a failure, attempts made to resolve the problem, and explicit asks. The answers to these questions are usually diagnostic or troubleshooting procedures contained in documents which need to be searched based on an understanding of the problem.

(Vtyurina and Clarke, 2016) tackle open domain complex questions by learning keywords from a large supervised dataset of questions and answers. Extraction of such keywords from questions certainly helps retrieval by removing noise. However, in technical support, the multiple types of attributes necessitate their separate extraction and reasoning. "Activity" words extracted from the question as keywords would hurt retrieval unless they are correctly identified, and their matching score is reduced. But, the same word if present as "Intent" would certainly help improve retrieval and their matching score would need to be boosted. Finally, (Yang et al., 2017) presents a system which improves the state of the art for technical support question answering. Their work is complementary to ours, as they address the problem of knowledge representation by constructing a knowledge graph from content and do not delve into the complexities of technical support problems.

To the best of our knowledge, ours is the first attempt that defines important finer attributes in technical support questions, proposes a model to extract them, and performs reasoning to retrieve relevant results.

3 Semantic Parsing And Reasoning

As explained earlier, we identify the following three attributes for technical support questions:

1. **Symptom** - A description of what's failing
2. **Activity** - Steps that have been tried to remediate the problem
3. **Intent** - A specific request or ask

Questions can contain multiple of each attributes. Because of this inherent complexity, it is imperative to parse these attributes for retrieving relevant results. Table 2 shows the frequency distribution of the different attributes in the training set.

Formally, the problem of semantic parsing is formulated as follows. Given a support question Q, which is comprised of a sequence of tokens - $Q = [q_1, q_2, .., q_n]$, we want to extract three sequences S, A and I which denote the symptoms, activities and intents (these sequences are not necessarily contiguous, allowing for multiple symptoms, activities, and intents in the same question).

We present a Semantic Parser to extract these attributes from questions using rules built on the grammatical structure of sentences. We also present a Conditional Random Field model, which, in the presence of large amount of supervision, outperforms the rule based Semantic Parser. However, such annotations are expensive to curate. Details of both the models are presented below.

3.1 Semantic Parser

There are several natural language parsers that allow tokenization and POS tagging. Some also provide a simple description of the grammatical relations in a sentence, but this is not sufficient for several applications. Slot Grammar system has a lexicalist character and treats different grammatical structures in a language independent manner through different rule types. IBM Watson uses two deep parsing components: English Slot Grammar (ESG) followed by Predicate Argument Structure (PAS) for linguistic analysis of text (McCord, 1990a).

ESG works on a sentence to produce a parse tree which has a deep structure (logical analysis) and a surface structure (grammatical analysis). PAS simplifies the ESG parse tree structure to the core semantic meaning. PAS forms a labeled directed graph which is more flexible and requires less knowledge than

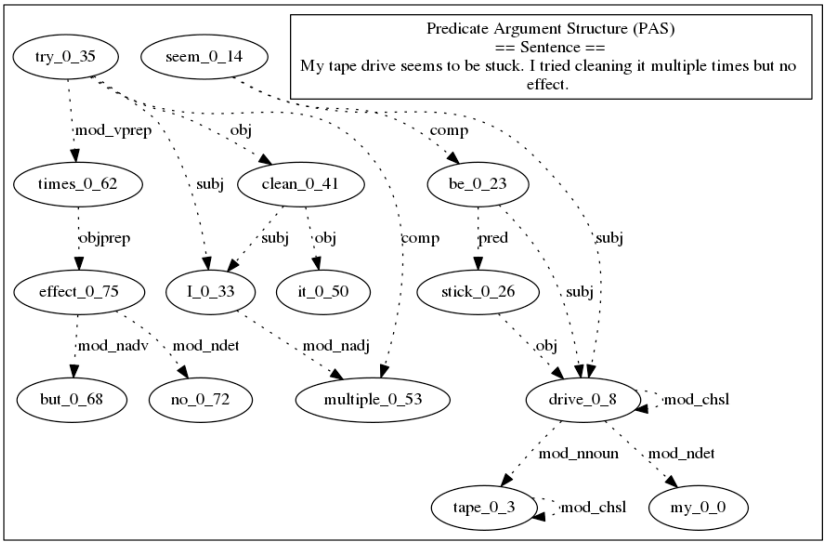

Figure 2: Sample PAS tree

```
pattern = there_is_np(@1) -> vp0[hasLemmaForm("be"),!hasParseFeature("q")]
{
        subj -> var0[hasLemmaForm("there")]
}
{
        pred -> vp1[hasLemmaForm("@1")]
}
```

Figure 3: Example of template that captures cases like “there is error”, “there was error”. “@1” would be replaced by a dictionary word such as *error message, symptom, blocker, expiration, or limitation*

ESG. PAS combines the two dimensions of ESG by omitting nodes with auxiliary verbs, closed class nodes introducing verb phrases, determiners (except for high semantic determiners), forms of *be* with no predicate and with adjective predicate. The negation annotator modifies the standard PAS by retaining some of these nodes, and we use the negation annotator to identify symptoms/problems like *does not work, cannot boot* from a question. Figure 2 shows an PAS tree structure for a sample sentence. While ESG retains syntactic information, PAS only represents semantics. Sentences with a similar meaning but different structures (like the active and passive form of the same sentence), would have varying ESG parses but the same PAS parse. Therefore, it is much more efficient to create patterns on the PAS tree than the ESG tree.

To obtain the attributes from user utterances, we create patterns to be matched to the input query. These patterns are essentially rules written on top of the slot grammar. Since manually specifying all possible rules can get cumbersome, templates are used to represent these patterns. Figure 3 presents a sample template for extracting symptoms like “there is/was an error”. These templates are combined with the dictionary words identified for each attribute (symptom, activity and intent) to form rules that are matched to the PAS tree for extraction. Text spanned by the ESG node, corresponding to the matched PAS node, is extracted and marked as the annotation output for the corresponding attribute. The base lexicon in ESG is augmented with support specific terminologies and Word-Net for wider coverage. Additionally, multi-word entries were added into chunk lexicons to obtain multi-word phrases as a single node in the PAS tree.

3.2 CRF based Parser

The task of extracting attributes from technical support problems can be formulated as a sequence labeling problem. This follows from the observation that these attributes manifest in a rough order -

symptoms, followed by attempts to mitigate, and finally explicit requests. Context words are also important in identifying spans of text for these attributes. CRFs are graphical models that can capture such dependencies among input observations (Lafferty et al., 2001). A CRF defines a conditional distribution $p(y|x)$ where *y* is the corresponding label sequence and *x* is the observed data sequence.The current label at time step t, y_t can be dependent on any of the previous *n* labels, and all the observations in an *n*-order CRF. In our case, the input sequence *x* corresponds to a series of words, while the label sequence *y* corresponds to the symptom, activity, intent or other assigned to the input sequence. The probabilistic model of a label sequence given some sequence of words is mediated in this model through a set of weighted functions f_i:

$$p(y|x) = \frac{exp(\sum_i \sum_t w_i f_i(y_{t-1}, y_t, x, t))}{Z(x)}$$

where the w_i are the weights assigned by the learning algorithm, and $Z(x)$ is a normalization factor over all label sequences.

For training, a manually annotated set of support questions is used. Spans are labeled with the B-I-O encoding, where each word in the query is annotated with one of the following classes (labels): the beginning of a symptom (B-S), inside of a symptom (I-S), the beginning of an activity (B-A), inside of an activity (I-A), the beginning of an intent (B-I), inside of an intent (I-I) or other (O).

To conduct our experiments, we used the linear-chain Stanford CRF implementation (Finkel et al., 2005) . We used current word, previous word, next word, current word character n-gram ($n \leq 6$), presence of word in left window (size = 4), presence of word in right window (size = 4), position of word in the sentence, and current POS tag as features computed using the Stanford NER toolkit.

3.3 Reasoning

We need to use the extracted attributes to perform reasoning on the retrieved results. For example, for the problem in Figure 1, we would not want to return back to the user, pages which detail a procedure for cleaning the tape drive. Similarly, when a user specifies a particular intent, we would ideally like to prioritize matches with this intent over matches with the symptom which might include other diagnostic or resolution procedures. For specifying these priorities, we create a module that weighs query fields in the retrieval engine based on the attribute type of the field. Query fields containing intents are given the highest weights, symptom containing query fields are given slightly lower weight, and fields with activity are given negative weights to suppress results matching this field. The actual values for weights are determined empirically.

4 Dataset and Experimental Setup

Training and evaluation of our parser models is done on a proprietary dataset of 1972 actual customer problems. These utterances have been manually annotated with the identified attributes by SMEs. 80% of this data is set aside for training and validating the CRF model. The rest of the data is used for evaluating both models.

Evaluation of retrieval results is done on two datasets which we call DB2 Prop, and DB2 Open, both of which contain questions and relevant answer links for DB2. The DB2 Prop dataset consists of proprietary user problems for which the correct answer url is provided by SMEs. The DB2 Open dataset is crawled from the developer works website [1]. It consists of user questions and urls extracted from "accepted" answers for those questions. We get manual annotations of attributes on 96 questions from DB2 Prop, and 48 questions from DB2 Open dataset from SMEs, and use these to evaluate retrieval performance.

For evaluation of retrieval results, we use elastic search as the retrieval index. The corpus is a crawl of publicly available technote pages [2] for the DB2 product, which are majority of the answers for both the DB2 Prop and DB2 Open datasets. From the downloaded pages, we extract and index the title and content of the body as document fields. The whole text of the problem is sent to this index (on both document

[1] https://developer.ibm.com/answers/topics/db2/

[2] https://www.ibm.com/mysupport/s/topic/0TO500000001fUNGAY/db2-linux-unix-and-windows, https://www.ibm.com/support/knowledgecenter/

Table 2: Distribution of questions with S, I, and A

Label	Count
Symptom	1773
Activity	121
Intent	510
Multiple Symptom	467
Multiple Activity	13
Multiple Intent	12

Table 3: Performance (%) Comparison for Symptoms: Strict Scoring

Method	Precision	Recall	F1
Semantic Parser	52.33	42.42	46.46
CRF	78.3	71.20	74.33

fields) as a baseline. The query containing the Semantic Parser extractions are sent with weights on query fields as described in section 3.3.

5 Evaluation

In this section we evaluate our system along two dimensions -

1. Precision, recall and F1 score for the extractions from the Semantic Parser and CRF
2. Relevance of the retrieved results between a whole query baseline and Semantic Parse based query

5.1 Evaluation of Attribute Extraction Models

Since annotations of these attributes were not readily available to us, we had Subject Matter Experts (SMEs) manually annotate 1972 questions from a proprietary ticketing system. The SMEs were asked to identify spans from questions which indicate the problem, efforts in resolving the problem, and explicit requests. 80% of these questions were used for training the CRF model, and the rest were used for evaluating both models. The distribution of symptom, activity and intent in our training dataset is shown in the Table 2. Longer more complex queries often had multiple symptoms, intent and activity (distributions shown in table 2).

As we can see, a large number of support questions have multiple symptoms. We designed two scoring schemes: strict scoring, where an extraction is marked correct only when all the values from ground truth are identified; and partial scoring, where the extraction is marked correct if any one of the values from ground truth is identified. The partial scoring scheme would not harshly evaluate models when most, but not all values of an attribute are extracted. Tables 3 and 4 summarize the performance of all the models. We present results on the strict scoring scheme only for symptoms, since there are very few questions with multiple activities and intents. For an attribute which has just one value in the ground truth, strict and partial scoring are the same. In Table 3, we note that CRF does the best in extracting multiple symptoms. The main reason is that CRF uses feature as well as contextual modeling of the words in the sentence. It inherently being a sequential labeling algorithm, accounts for its 78% precision in identifying all symptoms.

Note that CRF is a supervised learning model and needs manually annotated training data. The rule based Semantic Parser does not have this limitation and instead uses linguistic pattern matching on dependency trees. This renders it more generic and can be applied to problems from other products with minimal intervention. With a typical big service provider supporting close to 7-8k products, this is a useful property. We observe that the CRF outperforms the rule based Semantic Parser on Symptoms for which we have an abundance of training examples. For the other two attributes, the Semantic Parser is slightly better than the CRF.

Table 4: Performance (%) Comparison (Symptom, Activity and Intent): Partial Scoring

Model	Symptom			Activity			Intent		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Semantic Parser	67.81	58.84	62.17	71.33	37.2	48.64	88.44	59.29	70.63
CRF	88.64	81.60	84.35	70.0	33.33	44.85	93.74	48.67	64.07

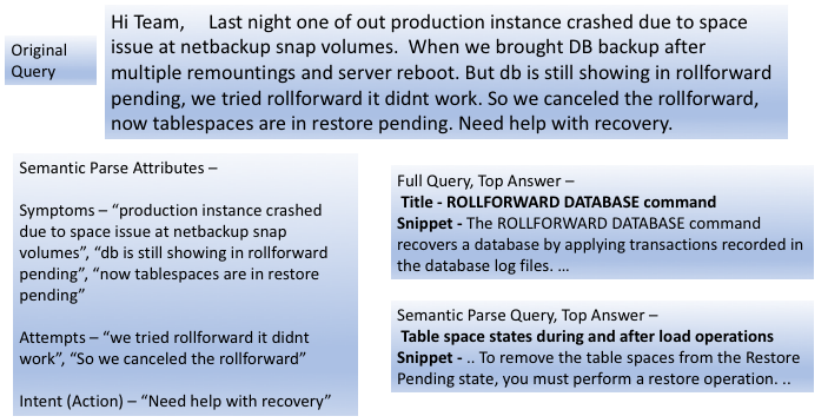

Figure 4: Question with extractions and retrieved results

5.2 Evaluation of retrieved results

As explained in the setup, evaluation of retrieved results is performed using elastic search. Figure 4 gives an example of the top retrieved results using the original query and the Semantic Parse attributes. The baseline query consists of the whole question, and extractions from the Semantic Parser are weighed according to the attribute-type in the reasoner. The highest weight is given to intent extractions, and a negative weight is assigned to attempt extractions. We see that because of repetitions of the token "rollforward" in the original query, the top result returned is about the "ROLLFORWARD DATABASE command". When we give a negative weight to the attempt Semantic Parse extraction, and increase the weight of the symptom extraction (although less than that of the intent extraction), the result about "Table space states" bubbles to the top.

The retrieval results (Precision@k) on the two datasets are presented in Table 5. Precision@k measures the fraction of questions with the correct answer in the top k results.

$$Precision@k = \frac{n_k * 100}{N} \tag{1}$$

where n_k is the number of questions with the correct answer in top k results, and N is the total number of questions.

We see a slight improvement in P@1 on the DB2 Prop dataset, and a 42.9% improvement in P@5. However, on the DB2 Open dataset, we see very slight improvement on the P@1 and no improvement in P@5. On further analysis we find that there are two reasons for this. Firstly, we find that for a large number of the queries, the answer marked as the ground truth is only partially related to the solution, and sometimes it is completely different. Since the accepted answer was assumed to be correct and there is no manual verification of the correctness in this dataset (unlike in DB2 Prop), we find that the confidence on ground truth in DB2 Open is low. Another reason we find is that in DB2 Open, people often attach code snippets and stack traces in the question, which can easily be understood by humans but are difficult for the models to annotate. Identifying and extracting out (and in the future, understanding) these code snippets and stack traces, would improve the accuracy on this dataset further.

Table 5: Retrieval Performance (P@k)

Model	DB2 Prop			DB2 Open		
	P@1	P@3	P@5	P@1	P@3	P@5
Whole Query	7.29	14.58	28.13	25.0	33.33	37.5
Semantic Parse Query	9.38	19.79	40.63	27.08	31.25	37.5

6 Discussion and Future Work

Through our experiments, evaluation and error analysis, we stumble across a number of problems that can be addressed to improve understanding of technical support questions, and increase retrieval accuracy. Some of the interesting problems are detailed below -

- **Sub-division of Activity** Through careful analysis of activity extractions, we find that it can further be split into two classes that help better understanding of the question. The first class are actions or steps that were taken before the problem occurred. For example, in the snippet - *We tried to upgrade the DB2 AESE ON RHEL from 10.5.5 to 11.1 and in upgrade check got orphan rows*, "trying to upgrade DB2" is not a step taken to remediate the problem, but the action that resulted in the error. The second class are actions that are taken to remediate the problem (the activity extraction in figure 4). It is clear that we would not want to exclude documents/procedures containing tokens from the first class from appearing in the results. The challenge with creating such a distinction, we find, is that examples from both these classes manifest in very similar linguistic patterns. Correct distinction between these two classes would probably require us to use position features from the symptom extraction.

- **Identify Stack Traces, Core Dumps, and Code Snippets** As identified in section 5.2, stack traces, core dumps, and code snippets, confuse the Semantic Parser. Returning these poor extractions hurts retrieval performance. Further, if we could truly understand these sections in the way humans do, it would certainly increase efficiency of retrieval.

- **Noisy Intent Extractions** Extraction of patterns like "please help" or "please suggest resolution" as intent, creates noise for retrieval. Such extractions could be filtered if they do not contain entity or action keywords from the domain.

- **Semantic Parsing performance** - We see that the CRF models outperforms the Semantic Parser in the presence of large amounts of training data by leveraging contextual features. However, for cases where training data is not available, it fails to reach comparable performance with the rule based parser. The Semantic Parser also has the advantage of being easily generalizable across products as it works with the semantic parse of sentences. An ideal model would be able to combine these two strengths, being generalizable and being able to leverage contextual features.

References

[Bordes and Weston2016] Antoine Bordes and Jason Weston. 2016. Learning end-to-end goal-oriented dialog. *CoRR*, abs/1605.07683.

[Chen et al.2016] Long Chen, Joemon M. Jose, Haitao Yu, Fajie Yuan, and Dell Zhang. 2016. A semantic graph based topic model for question retrieval in community question answering. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 287–296.

[Dang et al.2007] Hoa Trang Dang, Diane Kelly, and Jimmy J Lin. 2007. Overview of the trec 2007 question answering track. In *Trec*, volume 7, page 63.

[de Marneffe and Manning2008] Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The stanford typed dependencies representation. In *Coling 2008: Proceedings of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8.

[Dhoolia et al.2017] Pankaj Dhoolia, P Chugh, P Costa, Neelamadhav Gantayat, M Gupta, N Kambhatla, R Kumar, S Mani, P Mitra, C Rogerson, et al. 2017. A cognitive system for business and technical support: A case study. *IBM Journal of Research and Development*, 61(1):7–74.

[Fan et al.2012] J. Fan, A. Kalyanpur, D. C. Gondek, and D. A. Ferrucci. 2012. Automatic knowledge extraction from documents. *IBM Journal of Research and Development*, 56(3.4):5:1–5:10.

[Finkel et al.2005] J. R. Finkel, T. Grenager, and C. D. Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Association for Computational Linguistics*.

[Flinders2015] Karl Flinders. 2015. Amelia, the ipsoft robot. *Computer Weekly*.

[Gupta et al.2017] Abhirut Gupta, Arjun Akula, Gargi Dasgupta, Pooja Aggarwal, and Prateeti Mohapatra. 2017. Desire: Deep semantic understanding and retrieval for technical support services. In Khalil Drira, Hongbing Wang, Qi Yu, Yan Wang, Yuhong Yan, François Charoy, Jan Mendling, Mohamed Mohamed, Zhongjie Wang, and Sami Bhiri, editors, *Service-Oriented Computing – ICSOC 2016 Workshops*, pages 207–210, Cham. Springer International Publishing.

[Lafferty et al.2001] J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *International Conference on Maching Learning*.

[Lally et al.2012] Adam Lally, John M. Prager, Michael C. McCord, Branimir Boguraev, Siddharth Patwardhan, James Fan, Paul Fodor, and Jennifer Chu-Carroll. 2012. Question analysis: How watson reads a clue. *IBM Journal of Research and Development*, 56(3):2.

[Li and Roth2006] Xin Li and Dan Roth. 2006. Learning question classifiers: the role of semantic information. *Natural Language Engineering*, 12(3):229–249.

[McCord et al.2012] Michael C. McCord, J. William Murdock, and Branimir Boguraev. 2012. Deep parsing in watson. *IBM Journal of Research and Development*, 56(3):3.

[McCord1990a] Michael C. McCord. 1990a. Slot grammar. In Rudi Studer, editor, *Natural Language and Logic*, pages 118–145, Berlin, Heidelberg. Springer Berlin Heidelberg.

[McCord1990b] Michael C. McCord. 1990b. Slot grammar: A system for simpler construction of practical natural language grammars. In *Proceedings of the International Symposium on Natural Language and Logic*, pages 118–145.

[Metzler and Croft2005] D. Metzler and B. W. Croft. 2005. Analysis of statistical question classification for fact-based questions. *Information Retrieval*, 8(3):481–504.

[Murdock et al.2012] J. William Murdock, James Fan, Adam Lally, Hideki Shima, and Branimir Boguraev. 2012. Textual evidence gathering and analysis. 56:8.

[Vtyurina and Clarke2016] Alexandra Vtyurina and Charles LA Clarke. 2016. Complex questions:let me google it for you. In *Proceedings of the second Web QA Workshop*.

[Wang et al.2012] Chang Wang, Aditya Kalyanpur, James Fan, Branimir Boguraev, and David Gondek. 2012. Relation extraction and scoring in deepqa. *IBM Journal of Research and Development*, 56(3):9.

[Yang et al.2015] Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2013–2018.

[Yang et al.2017] Shuo Yang, Lei Zou, Zhongyuan Wang, Jun Yan, and Ji-Rong Wen. 2017. Efficiently answering technical questions a knowledge graph approach.