Natural Language Query Refinement for Problem Resolution from Crowd-Sourced Semi-Structured Data

Rashmi Gangadharaiah and Balakrishnan Narayanaswamy
IBM Research,
India Research Lab
{rashgang,murali.balakrishnan}@in.ibm.com

Abstract

We study the problem of natural language query generation for decision support systems (DSS) in the problem resolution domain. In this domain, a user has a task he is unable to accomplish (eg. *bluetooth headphone not playing music*), which we capture using language structures. We show how important units that define a problem can robustly and automatically be extracted from large noisy online forum data, with no labeled data or query logs. We also show how these units can be selected to reduce the number of interactions and how they can be used to generate natural language interactions for query refinement.

1 Introduction

Decision Support Systems (DSSs) that help decision makers extract useful knowledge from large amounts of data have found widespread application in areas ranging from clinical and medical diagnosis (Musen et al., 2006) to banking and credit verification (Palma-dos Reis et al., 1999). IBM's Watson Deep Question Answering system (Ferrucci et al., 2010), can be applied to DSSs to diagnose and recommend treatments for lung cancer and to help manage health insurance decisions and claims[1]. Motivated by the rapid explosion of contact centres, we focus on the application of DSSs to assist technical contact center agents.

Contact center DSSs should be designed to assist an agent in the *problem resolution* domain. This domain is characterized by a user calling in to a contact center with the problem of being unable to perform an action with their product (e.g. *I am unable to connect to youtube*). Currently contact center DSSs are essentially search engines for technical manuals. However, this has two shortcomings : (i) in most cutting edge consumer technology, like software and smart devices, the range of possible applications and use cases makes it impossible to list all of them in the manuals- limiting their usefulness under the heavy tailed nature of customer problems, (ii) contact centers are known to suffer from high churn due to pressures and difficulties of the jobs, particularly the need for rapid resolution, making ease of use essential since users of these DSSs are somewhere between experts (in using the system) and novices (in the actual technology customers need help with).

With the birth and growth of the Web 2.0 and in particular, large and active online product forums, such as, Yahoo! Answers [2], Ubuntu Forums [3] and Apple Support Communities [4], there is the hope that other technology savvy users will find and resolve large number of problems within days of the release of a product. However, these forums are *noisy*, i.e. they contain many throw-away comments and erroneous solutions. The first important question we address in this paper is, how can we mine relevant information from online forums and, in essence, *crowdsource* the creation of contact center DSSs? In particular, we show how many problems faced by consumers can be captured by *actions on attributes* (e.g. *bluetooth headphone not playing music*).

In order to address the second shortcoming, we study the problem of automatic *interactive* query refinement in DSSs. When DSSs are used by non-computer scientists, natural language understanding and interaction problems take center-stage (Alter, 1977). Since both customers and agents are not experts in a technical area, mis-understandings are common. As agents are evaluated based on the number of problems resolved, it is often the case

[1]http://www.ihealthbeat.org/articles/2013/2/11/ibm-offering-two-new-decision-support-tools-based-on-watson.aspx

[2]http://answers.yahoo.com/
[3]http://ubuntuforums.org/
[4]https://discussions.apple.com/

that queries entered by an agent are underspecified. In response to such a query, a search engine may return a large number of documents. For complicated technical queries, the time taken by an agent to read the long list of returned information and possibly reformulate the query could be significant. The second question we address in this paper is how can a DSS make *natural language* suggestions that assist the agent in acquiring additional information from a caller to resolve her problem in the shortest amount of time? Finally, for rapid prototyping and deployment, we develop a system and architecture that *does not use any form of labeled data or query logs*, a big differentiator from prior work. Query Logs are not always available for enterprise systems that are not on the web and/or have a smaller user base (Bhatia et al., 2011). When software and hardware change rapidly over time, it is infeasible to quickly collect large query logs. Also, logs may not always be accessible due to privacy and legal constraints.

To the best of our knowledge, this paper presents the first interactive system (and detailed evaluation thereof) for natural language problem resolution in the absence of manually labeled logs or pre-determined dialog state sequences. Concretely, our primary contributions are:

• **Problem Representation and Unit Extraction:** We define and automatically extract units that best represent a problem. We show how to do this robustly from noisy forum threads, allowing us to use abundant online user generated content.

• **Unit selection for Interaction:** We propose and evaluate a complete interactive system for users to quickly find a resolution based on semi-structured online forum data. Follow up questions are generated automatically from the retrieved results to minimize the number of interactions.

• **Natural Language Question Generation:** We demonstrate that, in a dialog system it is possible and useful to automatically generate fluent interactions based on the units we define *using appropriate templates*. We use these to create follow up questions to the user, which have much needed context, and show that this improves precision.

2 Proposed System

In online forums, people facing issues with their product (thread initiators) post their problems and other users write subsequent posts discussing solutions. These threads form a rich data source that could contain problems similar to what a user who calls in to the contact center faces, and can be used to find an appropriate solution. Our system (Figure 1) has two phases. In the offline phase, the system extracts units that describe the problem being discussed. In the online phase, the interaction engine selects the most appropriate units that best divide the space of search results obtained, to minimize the number of interactions. The system then generates follow up interactions by inserting the units into appropriate unit type dependent templates. The answers to these follow up questions are used to improve the search results.

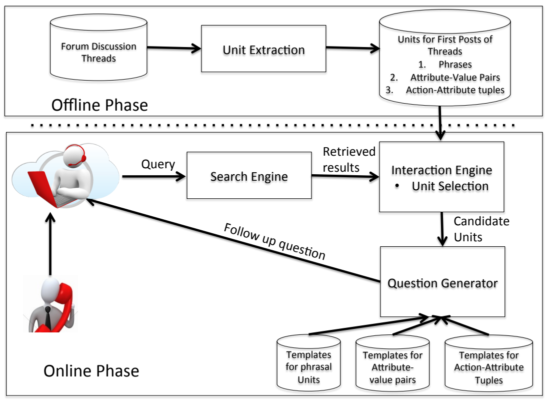

Figure 1: System Description

2.1 Representational Units

It is important to select representational units that capture the signature, or the most important characteristics of the information that users search for. This signature should be sufficient to find relevant results. In order to understand the right units for the problem resolution domain, we conducted the following user study. Five annotators analyzed the first posts from 50 threads from the Apple Discussion Forum, and were asked to mark the most relevant short segments of the post that best described the problem (an example in Table 1).

I cannot hear the notifications on my bluetooth now. it's at normal volume when i send a message, if i receive an email or text volume is very low yet I have all volumes up all the way. Is there a new bluetooth volume i have to turn up? or did another update screw the bluetooth . Was working just great before i updated to ios - 4 . please help me.

Table 1: Relevant short segments for a forum post.

Based on the user study, the first kind of units we considered were **phrases**, which are consecutive words that occur very frequently in the thread.

Phrases as query suggestions have been shown to improve user experience when compared to just showing terms (Kelly et al., 2009) since longer phrases tend to provide more context information. One shortcoming of these contiguous phrasal units is that they are sensitive to typography, i.e. small changes in phrasing (e.g. *ios - 4* and *ios 4*) lead to different phrases and the occurrence counts are divided among these variations. This causes difficulties both in the problem representation as well as in the search for problem resolution which are exacerbated by the noisy, casual syntax in forums. Motivated by Probst et al. (2007), we extract **attribute-value pairs**. These units provide both robustness as well as more configurational context to the problem. Another observation from the segments marked was that many of them involved a user wanting to perform an action (*I cannot hear notifications on bluetooth*) or the problems caused by a user's action (*working great before I updated*). We capture them using **action-attribute tuples** (details in Section 3.1.1).

Thread initiators describe the same problem in different ways leading to multiple threads discussing the same problem. Ideally, we want the representation of the problem to be the same for all these threads to build robust statistics. Consider the following examples, *sync server has failed*, *sync server failed*, *sync server had failed*, *sync server has been failing*. While the phrasing is different, we see that their dependency parse trees (Figure 2) show a common relation between the verb or action, *fail*, and the attribute *sync server* (the base form of the verbs are obtained from their corresponding lemmas with TreeTagger (Schmid, 1994)). Motivated by this, we use dependency parse trees for extracting action-attribute tuples.

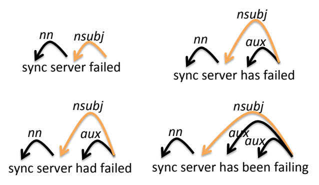

Figure 2: Dependency Parse Trees for various forms of the same problem.

3 Detailed System Description

We now give a detailed description, showing how the three types of units can be robustly extracted and used from noisy online forums.

3.1 Offline Component

In the offline phase we first extract candidate units that describe a problem (and its solution) from the forum threads. We then filter this description, using the thread itself, to retain the important units.

3.1.1 Candidates for Problem Description

Sentences are tagged and parsed using the Stanford dependency parser (de Marneffe et al., 2006). The following units are then obtained from the first post of the discussion thread.

Phrasal units: defined to be Noun Phrases satisfying the regular expression, *(Adjective)* (Number/Noun/Proper Noun)*+, (eg., *interactive web sites, osx widgets, 2007 outlook calendar*). These are extracted from the discussion thread along with their frequencies of occurrence.

Attribute-Value pairs: The dependency relations *amod* (adjectival modifier of a noun phrase) and *num* (numeric modifier of a noun) in the parsed output are used for this purpose. In the case of *amod*, the attribute is the noun that the adjective modifies and its value is the adjective. For example, with *amod(signal,strong)*, the attribute is *signal* and its value is *strong*. In the case of *num*, the attribute is the noun and its value is the numeric modifier. For example, with *num(digits,10)*, the attribute is *digits* and its value is *10*. As mentioned in Section 2.1, these pairs capture more context of the problem being discussed. Additional attribute-value pairs are extracted by expanding the attributes with the adjacent nouns and adjectives that occur with it. For the example in Figure 3, the attribute *signal* is modified to *cell phone signal* and added to the list of attribute-value pairs along with their frequencies of occurrence.

Action-Attribute tuples: The dependency relations used for these units are given in Table 3 with examples. Many of these units help describe the user's problem while others provide contextual information behind the problem being discussed. These units are described with 4-tuples (Arg_1-*verb*-Arg_2-Arg_3), three of which are the arguments of the verb or attributes of an action. The relations given in the first column of Table 3 form fillers for the attributes of the action. The example in Figure 3 gives the tuple, *I-entered-dns-null*, where, *I* is the subject, *entered* is the action performed, *dns* is the object. If the verb has a *prt* relation, the particle is appended to the verb. For example, *turned* has a *prt* relation in *prt(turned,off)*,

First Post	I cannot hear the notifications on my bluetooth now. it's at normal volume when i send a message but if i receive an email or text the volume is very low yet I have all volumes up all the way. Is there a new bluetooth volume i have to turn up with ios - 4? or is it that another update screwed with the bluetooth again. Was working just great before i updated ios - 4. please help me. thanks!
Phrases	volume, bluetooth volume, bluetooth, notifications, update, ios, normal volume, work, text, way, email, message, new bluetooth volume,
Phrases + Att-Val pairs	volume, bluetooth volume, bluetooth, notifications, update, ios, normal volume, work, text, way, email, message, new bluetooth volume, ios 4
Phrases + Att-Val pairs + Act-Att tuples	volume, bluetooth volume, bluetooth, notifications, update, ios, normal volume, work, text, way, email, message, new bluetooth volume, ios 4, low volume, I hear notifications on bluetooth, update screwed bluetooth, I send message, I receive email, it is at normal volume, working great before updated, I missed emails, *I volumes way.

Table 2: Problem representations for a forum post.

hence, the verb is now modified to *turned off*. Since *entered* in this example takes only a subject and an object as arguments, the third argument is *null*. Consider another example, *I removed the wep password in the router settings*, the tuple is now *I-removed-password-in the settings*. The last row in Table 3 gives an example of the usage of the *xcomp* relation. As done with Attribute-Value pairs, the attributes in these units are also expanded with the adjacent nouns and adjectives and added to the list of Action-Attribute tuples along with their frequencies of occurrence in the entire thread.

3.1.2 Scoring and Filtering

Since the problem is defined by the thread initiator in the first post of the thread, units in the first post are scored and ranked based on tf-idf (Manning et al., 2008). We treat each thread as a document and the top 50 highest scoring candidates form the problem description for the thread. Units are extracted from the rest of the thread in order to obtain frequency statistics for the units in the first post. Pronouns, prepositions and determiners are dropped from the units while obtaining the counts. In addition, verbs in the action-attribute tuples are converted to their base form using the lemma information obtained from TreeTagger (Schmid, 1994), to obtain counts. This makes the scores robust to small variations in the units.

Examples of extracted units are shown in Table 2. We see that errors in the parse (*volumes* was tagged as a verb) cause erroneous units (**I volumes up*). For this reason, we use frequency statistics from the rest of the discussion thread, to determine if a unit is valid or not. The tf-idf based scheme also removes commonly used phrases such as, *please help me*, *thank you*, etc.

3.2 Online Phase

The system searches for a set of initial documents based on the user's initial query. Next, the follow-up candidate units are selected (Section 3.2.1) from the units extracted in the offline phase for the

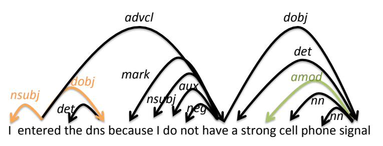
advcl
dobj
det
mark
amod
nsubj
dobj
nsubj
aux
nn
det
neg
nn
I entered the dns because I do not have a strong cell phone signal

Figure 3: Dependency parse: *I entered the dns because I do not have a strong cell phone signal.*

retrieved documents and natural language interactions are further generated by filling the templates (Section 3.2.2) with the selected units.

3.2.1 Selection of candidate units for Question Generation

Interactions should be selected to (i) understand the user's requirements better by making the query more specific and reduce the number of results returned by the search engine, and (ii) reduce the number of interactions required. We use information gain to find the best unit that reduces the search space, motivated by its near optimality (Golovin et al., 2010). If S is the set of all retrieved documents, $S_1 \subseteq S$ containing $unit_i$ and S_2 is a subset of S that do not contain $unit_i$, the gain from branching (or interacting) on $unit_i$ is,

$$Gain(S, unit_i) = E(S) - \frac{|S_1|}{|S|}E(S_1) - \frac{|S_2|}{|S|}E(S_2) \quad (1)$$

$$E(S) = \sum_{k=1,\ldots|S|} -p(doc_k)log_2 p(doc_k) \quad (2)$$

Where, each document is assigned a probability based on its rank in the search results:

$$p(doc_j) = \frac{\frac{1}{rank(doc_j)}}{\sum_{k=1,\ldots|S|}\frac{1}{rank(doc_k)}} \quad (3)$$

The unit that gives the highest information gain forms the candidate for question generation. Information gain has been widely used in Decision Trees (Mitchell, 1997), where the nodes represent attributes and the edges indicate values, and is known to result in short trees. In our case, the nodes represent the follow up questions and the edges indicate whether the user's answer is *yes* or

Relation	Example	Attributes(Arg_1/Arg_2/Arg_3)	Action(Verb)
nsubj	nsubj(entered,I)	I (Arg_1)	entered
dobj	dobj(entered,dns)	dns (Arg_2)	entered
iobj	iobj(give, address)	address (Arg_3)	address
pobj	prep(connect,to), pobj(to,wifi)	wifi(Arg_2)	connect
prep_(to,into, etc)	prep_in(removed,settings)	in the settings (Arg_3 if Arg_2 not present, else Arg_2)	removed
xcomp	xcomp(prompt, connect), prep_to(connect,wifi), nsubj (prompt, password)	password(Arg_1), to wifi (Arg_2)	prompt to connect

Table 3: Dependency relations used to extract Action-Attribute tuples

no. The goal in decision trees is to quickly exhaust the space of examples with fewer steps, resulting in shorter trees. The goal in this paper is to traverse the space of results obtained with the initial query to reach the most relevant document with the fewest interactions. Since the dialog problem can be easily mapped to decision trees, the choice of information gain allows the user to arrive at the most relevant document with the smallest number of interactions in the online phase.

3.2.2 Question Generation

Questions are generated based on the type, number and tense information present in the units. The list of templates used for question generation is given in Table 4. Once a candidate unit is selected, a template is chosen based on its type. Phrasal units have a single template. If an attribute has two values with very similar information gains, the template for Attribute-Value pairs accommodates the different values. For example, if the pairs, *Outlook:2003* and *Outlook:2007* have very similar gains, the question would then be *Is your outlook: Option*$_1$*:2003 Option*$_2$*:2007 ?* and the user has the option to click on the one that is relevant to his query. For Action-Attribute tuples, the templates are chosen based on the the person, number and tense information from the verbs (Table 4). *null* in the table (for example, *null-send-emails-null*) indicates that a particular argument does not exist or was not found and hence the argument will not be added to the appropriate template. Certain templates require converting the verb to a different form (e.g., *VBD* to *VBN*). This mapping is stored as a dictionary obtained by running the TreeTagger on the entire dataset and various forms are automatically obtained by linking them to the lemmas of the verbs (for example, *give(VB/lemma) gave(VBD) given(VBN) gives(VBZ)*).

4 Results and Discussion

To evaluate our system, we built and simulated a contact center DSS for iPhone problem resolution.

4.1 Description of Dataset

We crawled threads created during the period 2007-2011 from the Apple Discussion Forum resulting in about $147,000$ discussion threads. In order to create a test data set, threads were clustered treating each discussion thread as a data point using a tf-idf representation. The thread nearest the centroid from the 60 largest clusters were marked as the ‘most common’ problems.

Underspecified Query 1: “cannot sync calendar”
Forms 6 specific queries
1. because iphone disconnected
2. because the sync server failed to sync
3. because the sync session could not be started
4. because the phone freezes
5. error occurred while mingling data
6. error occurred while merging data

Table 5: Specific and under-specified queries

To generate specific and under-specified queries on this data set, in our experiments, we use the first post as a proxy for the problem description. An annotator created one short query (underspecified) from the first post of each of the 60 selected threads. These queries were given to the Lemur search engine (Strohman et al., 2005) to retrieve the 50 most similar threads from an index built on the entire set of $147,000$ threads. He manually analyzed the first posts of the retrieved threads to create contexts, resulting in 217 specific queries. To understand this process we give an example from our data creation in Table 5. Starting from an under-specified query *cannot sync calendar*, the annotator found 6 specific queries. Two other annotators, were given these specific queries along with the search engine’s results from the corresponding under-specified query. They were asked to choose the most relevant results for the specific queries. The intersection of the choices of the annotators formed our ‘gold standard’.

4.2 User based analysis of the problem representation and unit extraction

We conducted two user studies to determine the (subjective) value of our problem representation,

Unit's Type	**Template (with examples)**
Phrases	Is your query related to *[unit] ?* eg: Is your query related to *osx widgets ?*
Attribute-Value pairs (if single value)	Is your *[attribute] [value] ?* eg: Is your *wifi signal strong ?*
Attribute-Value pairs (if multiple values)	Is your *[attribute]*: Option$_1$: *[value$_1$]* ... Option$_n$:*[value$_n$]* ? eg: Is your *outlook calendar*: Option$_1$:*2003* Option$_2$:*2007* ?
Action-Attribute tuples (Verb/Action is VB: base form) ARG$_1$ is empty / ARG$_1$ is a pronoun	Does the *[ARG$_1$(sg)] [VERB]* the *[ARG$_2$] [ARG$_3$]* ? Do the *[ARG$_1$(pl)] [VERB]* the*[ARG$_2$] [ARG$_3$]* ? Do you want to *[VERB]* the*[ARG$_2$] [ARG$_3$]* ? eg: Does the *wifi network prompt* the *password* ? from the tuple: wifi network-prompt-password-null
Action-Attribute (Verb/Action is VBP: non-3rd person, singular, present) ARG$_1$ is empty / ARG$_1$ is a pronoun	*[ARG$_1$] [VERB] [ARG$_2$] [ARG$_3$]* ? Do you want to *[VERB]* the *[ARG$_2$] [ARG$_3$]* ? eg: Do you want to *send* the *emails* ? from the tuple: null-send-emails-null
Action-Attribute (Verb/Action is VBN: past participle) ARG$_1$ is empty / ARG$_1$ is a pronoun ARG$_2$ and ARG$_3$ are empty ARG$_2$ and ARG$_3$ are empty	 Have you *[VERB] [ARG$_2$] [ARG$_3$]* ? Has the *[ARG$_1$(sg)]* been *[VERB]* ? Have the *[ARG$_1$(pl)]* been *[VERB]* ? Has the *[ARG$_1$$(sg)$] [VERB]* the *[ARG$_2$] [ARG$_3$]* ? Have the *[ARG$_1$$(pl)$] [VERB]* the *[ARG$_2$] [ARG$_3$]* ? eg:Has the *update caused* the *phone to crash* ? from the tuple: update-caused-phone-to crash
Action-Attribute (Verb/Action is VBZ: 3rd person, singular, present) ARG$_1$ is empty	Does the *[ARG$_1$] [VERB$_{VB}$]* the *[ARG$_2$] [ARG$_3$]* ? Does the phone *[VERB$_{VB}$]* the *[ARG$_2$] [ARG$_3$]* ? eg: Does the *iphone use idol support* from the tuple: iphone-uses-idol support-null
Action-Attribute (Verb/Action is VBD: past tense) ARG$_1$ is empty ARG$_1$ is empty ARG$_1$ is a pronoun	Has the *[ARG$_1$$(sg)$] [VERB$_{VBN}$] [ARG$_2$] [ARG$_3$]* ? Have the *[ARG$_2$$(pl)$] [ARG$_3$]* been *[VERB$_{VBN}$]* ? Is the *[ARG$_2$$(sg)$] [ARG$_3$] [VERB$_{VBN}$]* ? Have you *[VERB$_{VBN}$] [ARG$_2$] [ARG$_3$]* ? Have the *[ARG$_1$$(pl)$] [VERB$_{VBN}$] [ARG$_2$] [ARG$_3$]* ? eg: Has the *iphone found several networks* ? from the tuple: iphone-found-several networks-null
Action-Attribute (Verb/Action is VBG: gerund/present participle) ARG$_1$ is empty	Is the *[ARG$_1$$(sg)$]* *[VERB]* the *[ARG$_2$] [ARG$_3$]* ? Are the *[ARG$_1$$(pl)$] [VERB]* the *[ARG$_2$] [ARG$_3$]* ? Is the phone *[VERB]* the *[ARG$_2$] [ARG$_3$]* ? eg: Is the *site delivering* the *flash version* ? from the tuple: site-delivering-flash version-null

Table 4: Templates for the follow up Question Generation .

focusing on action-attribute tuples. In the first user study, 5 users were given the first post of 20 threads with three problem representations for the first post, (1) phrasal units only, (2) phrasal units and attribute-value (Att-Val) pairs and (3) phrasal units, attribute-value pairs and action-attribute (Act-Att) tuples. They were asked to indicate which representation best represented the problem. All users preferred the third representation on all the first posts. An example first post and units are in Table 2.

In the second study, the same 5 users were asked to indicate how many units in Representation 3 were not relevant to the problem discussed in the first post, for a subset of 10 threads. We defined 'not relevant' as noisy components which do not aid in the problem representation e.g. *oh boy* and *thanks!* (see Table 2). All users marked 2 examples (*sort* and *way*) as not relevant, out of 110 units that the algorithm generated for these threads.

These two user studies taken together show that the combined set of units, is able to capture the problem description well and that our algorithm is able to filter out noise in the thread to create a robust and useful representation of the problem. The results in Section 4.3 (Tables 6 and 7), show the value of our problem representation, in a complete end-to-end system, with objective metrics.

4.3 Unit selection for interaction

We evaluate a complete system with both user (or agent) and search engine in the loop. We focus on measuring the value of the interactions by an analysis of which results 'rise to the top'. The experiment was conducted as follows. Annotators were given a specific query and its underspecified query

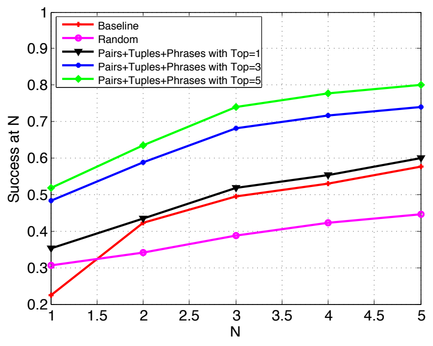

Figure 4: Success at N.

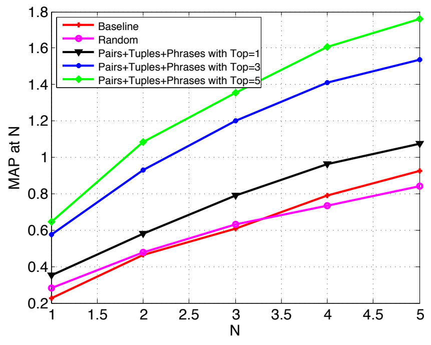

Figure 5: MAP at N.

(as created in Section 4.1) along with the results obtained when the underspecified query was input to the search engine. They were presented with the $Top = 1, 3$ or 5 scoring follow up questions. E.g., for the underspecified query in Table 5 and specific query 2, the generated question was (see Table 4), *Has the sync server failed to sync?*. The user then selected the most appropriate follow up question, reducing the number of results. We then measured the relevance of the reduced result, with respect to the gold standard (see Section 4.1) for that specific query, using metrics commonly used in Information Retrieval - MRR, Mean Average Precision (MAP) and Success at rank N (Baeza-Yates et al., 1999). We restrict $N = 5$ (small) since the rapid resolution time required of contact center agents does not allow them to look at many results.

In Figures 4, 5 and Table 6, we compare our system against a baseline system, which is the set of results obtained with the underspecified query, and a system where 5 interaction units are selected at random from the initial search results. Note that, as the number of follow up questions presented increases, the scores will improve since it is more likely that the 'correct' choice is presented. However, there is a trade-off here since the agent has to again peruse more questions, which increases time spent, and so we limit this value to 5 as well. In terms of all three measures, our system is able

Unit's Type	MRR
Baseline	0.3997
Random	0.4021
Phrases (with Top=5)	0.6548
Phrases, Pairs (with Top=5)	0.6745
Phrases, Pairs, Tuples (with Top=5)	0.7362

Table 6: MRR for different unit types.

to give a substantial improvement in performance. E.g., one intelligently chosen interaction performs better than 5 randomly chosen ones. These results show the value of the units we select and the choice of information gain as a metric. To measure the importance of each unit type, we analyzed the selected follow up questions ($Top = 5$) for each underspecified query. Table 7 lists the fraction of queries whose origin was a specific unit type.

Unit's Type	Preference
Phrases	51%
Attribute-Value Pairs	12%
Action-Attribute Tuples	37%

Table 7: Fraction of follow up questions selected that originated from a specific unit type

4.4 Evaluating Templates for Question Generation

Finally, an annotator was given 100 generated follow up questions from the previous experiment and asked to label them as understandable or not. The annotator marked 13% as not understandable. Examples were, *does the phone connect*, *has the touchscreen stopped*, *does the message connect* (which were due to errors in parsing) and *do you want to leave the car* (due to a filtering error).

5 Related Work

Our work is related to three somewhat distinct areas of research, dialog systems, question answering (QA) systems and interactive search. Unlike most QA systems, we continue a sequence of interactions where the system and the user are active participants. The primary contribution of this work is a combined DSS, search, natural language dialog and query refinement system built automatically from semi-structured forum data. No prior work on interactive systems deals with problem resolution from large scale, noisy online forums.

Many speech dialog systems exist today for tasks including, obtaining train information (RAILTEL) (Bennacef et al., 1996), airline information (Rudnicky et al., 2000) and weather information (Zue et al., 2000). These systems perform simple database retrieval tasks, where, the keywords and their possible values are known apriori.

In general document retrieval tasks, when a user's query is under-specified and a large number of documents are retrieved, interactive search engines have been designed to assist the user in narrowing down the search results (Bruza et al., 2000). Much research has concentrated on query reformulation or query suggestions tasks. Suggestions are often limited to terms or phrases either extracted from query logs (Guo et al., 2008; Baeza-Yates et al., 2004) or from the documents obtained in the initial search results (Kelly et al., 2009). Bhogal et al. (2007) require rich ontologies for query expansion, which may be difficult and expensive to obtain for new domains. Leung et al. (2008) identify related queries from the web snippets of search results. Cucerzan and White (2007) use users' post-query navigation patterns along with the query logs to provide query suggestions. Mei et al. (2008) rank query suggestions using the click-through (query-url) graph. Boldi et al. (2009) provide query suggestions based on short random walk on the query flow graph. The main drawback behind these approaches is the dependence on query logs and labeled data to train query selection classifiers. We show how certain units are robust representations of documents in the problem resolution domain which can automatically be extracted from semi-structured data.

Feuer et al. (2007) use a proximity search-based system that suggests sub and super phrases. Cutting et al. (1993; Hearst and Pedersen (1996; Kelly et al. (2009) cluster retrieved documents and make suggestions based on the centroids of the clusters. Kraft and Zien (2004) and Bhatia et al. (2011) use n-grams extracted from the text corpus to suggest query refinement. Although these techniques do not rely on query logs for providing suggestions, the suggestions are limited to *contiguous* phrases. They also do not generate follow up questions, but instead provide a list of suggestions and require the user to select one among them or use them manually to reformulate the initial queries.

Automatically framing natural language questions as follow up questions to the user is still a challenging task since, (1) Diriye et al. (2009) and Kelly et al. (2009) showed that interactive query expansion terms are poorly used, and tend to lack information meaningful to the user, thus emphasizing the need for larger context to best capture the actual query/problem intent (2) finding a few question/suggestions that would narrow the search results will lead to fewer interactions as opposed to displaying the single best result (3) particularly for non-technical users, interactions and clarifications need to be fluent enough for the user to understand and continue his interaction with the system (Alter, 1977). In this paper, we show how to extract important representative contextual units (which do not necessarily contain contiguous words) and use these to generate contextual interactions.

Sajjad et al. (2012) consider a data set where objects belong to a known category, with textual descriptions of objects and categories collected from human labelers, using which n-gram based attributes of objects are defined. Subsets of these attributes are filtered, again using labeled data. Kotov and Zhai (2010) frame questions with the help of handmade templates for the problem of factoid search from a subset of Wikipedia. However, they do not select queries with the goal of minimizing the number of interactions. To extend these approaches to problem-resolution finding, (as opposed to factoids or item descriptions) simple most common noun phrases (as used in Sajjad et al. (2012) and Kotov and Zhai (2010)) are insufficient, since they do not capture the problem or intent of the user. As motivated in Section 1, this requires a better representation of candidate phrases. Our paper also suggests an approach that does not need any human labelled or annotated data. Suggestions are selected using units such that the problem intent is well captured and also ensure that fewer interactions take place between the user and the system. Follow-up questions are framed using templates designed for these units, allowing us to move beyond simple terms and phrases.

6 Conclusion and Future Work

This paper proposed an interactive system for natural language problem resolution in the absence of manually labelled logs or pre-determined dialog state sequences. As future work, we would like to use additional information such as, the trustworthiness of the posters, quality of solutions in the threads, etc., while scoring the documents.

References

Steven Alter. 1977. Why is man-computer interaction important for decision support systems? *Interfaces*, 7(2):109–115.

Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. 1999. *Modern information retrieval*. ACM press.

Ricardo Baeza-Yates, Carlos Hurtado, and Marcelo Mendoza. 2004. Query recommendation using query logs in search engines. In *EDBT*.

S. Bennacef, L. Devillers, S. Rosset, and L. Lamel. 1996. Dialog in the RAILTEL telephone-based system. In *ICSLP*.

Sumit Bhatia, Debapriyo Majumdar, and Prasenjit Mitra. 2011. Query suggestions in the absence of query logs. In *SIGIR*.

J. Bhogal, A. Macfarlane, and P. Smith. 2007. A review of ontology based query expansion. *Inf. Process. Manage.*

Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, and Sebastiano Vigna. 2009. Query suggestions using query-flow graphs. WSCD.

Peter Bruza, Robert McArthur, and Simon Dennis. 2000. Interactive internet search: keyword, directory and query reformulation mechanisms compared. In *SIGIR*.

Silviu Cucerzan and Ryen W. White. 2007. Query suggestion based on user landing pages. In *SIGIR*.

Douglass R. Cutting, David R. Karger, and Jan O. Pedersen. 1993. Constant interaction-time scatter/gather browsing of very large document collections. In *SIGIR*.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. *LREC*.

Abdigani Diriye, Ann Blandford, and Anastasios Tombros. 2009. A polyrepresentational approach to interactive query expansion. In *ACM/IEEE-CS*.

David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. 2010. Building Watson: An overview of the DeepQA project. *AI Magazine*, 31(3).

Alan Feuer, Stefan Savev, and Javed A Aslam. 2007. Evaluation of phrasal query suggestions. In *CIKM*.

Daniel Golovin, Andreas Krause, and Debajyoti Ray. 2010. Near-optimal bayesian active learning with noisy observations. In *NIPS*.

Jiafeng Guo, Gu Xu, Hang Li, and Xueqi Cheng. 2008. A unified and discriminative model for query refinement. In *SIGIR*.

Marti A. Hearst and Jan O. Pedersen. 1996. Reexamining the cluster hypothesis: scatter/gather on retrieval results. In *SIGIR*.

Diane Kelly, Karl Gyllstrom, and Earl W. Bailey. 2009. A comparison of query and term suggestion features for interactive searching. In *SIGIR*.

Alexander Kotov and ChengXiang Zhai. 2010. Towards natural question guided search. In *WWW*.

Reiner Kraft and Jason Zien. 2004. Mining anchor text for query refinement. In *WWW*.

Kenneth Wai-Ting Leung, Wilfred Ng, and Dik Lun Lee. 2008. Personalized concept-based clustering of search engine queries. *IEEE Trans. on Knowl. and Data Eng.*

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval*. Cambridge University Press.

Qiaozhu Mei, Dengyong Zhou, and Kenneth Church. 2008. Query suggestion using hitting time. In *CIKM*.

Tom M Mitchell. 1997. Machine learning. *Burr Ridge, IL: McGraw Hill*.

Mark A Musen, Yuval Shahar, and Edward H Shortliffe. 2006. Clinical decision-support systems. *Biomedical informatics*.

António Palma-dos Reis, Fatemeh Zahedi, et al. 1999. Designing personalized intelligent financial decision support systems. *Decision Support Systems*, 26(1).

Katharina Probst, Rayid Ghani, Marko Krema, Andy Fano, and Yan Liu. 2007. Extracting and using attribute-value pairs from product descriptions on the web. *From Web to Social Web: Discovering and Deploying User and Content Profiles*.

Alexander I. Rudnicky, Christina L. Bennett, Alan W. Black, Ananlada Chotimongkol, Kevin A. Lenzo, Alice Oh, and Rita Singh. 2000. Task and domain specific modelling in the carnegie mellon communicator system. In *INTERSPEECH*.

Hassan Sajjad, Patrick Pantel, and Micheal Gamon. 2012. Underspecified query refinement via natural language question generation. In *COLING*.

Helmut Schmid. 1994. Probabilistic part-of-speech tagging using decision trees. In *NEMLP*.

Trevor Strohman, Donald Metzler, Howard Turtle, and W Bruce Croft. 2005. Indri: A language model-based search engine for complex queries. In *ICIA*.

Victor Zue, Stephanie Seneff, James Glass, Joseph Polifroni, Christine Pao, Timothy J. Hazen, and Lee Hetherington. 2000. Jupiter: A telephone-based conversational interface for weather information. *IEEE Trans. on Speech and Audio Processing*.