Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Functional Partitioning of Ontologies for Natural Language Query Completion in Question Answering Systems

Jaydeep Sen, Ashish Mittal, Diptikalyan Saha, Karthik Sankaranarayanan
IBM Research AI
jaydesen@in.ibm.com, arakeshk@in.ibm.com, diptsaha@in.ibm.com, kartsank@in.ibm.com

Abstract

Query completion systems are well studied in the context of information retrieval systems that handle keyword queries. However, Natural Language Interface to Databases (NLIDB) systems that focus on syntactically correct and semantically complete queries to obtain high precision answers require a fundamentally different approach to the query completion problem as opposed to IR systems. To the best of our knowledge, we are first to focus on the problem of query completion for NLIDB systems. In particular, we introduce a novel concept of functional partitioning of an ontology and then design algorithms to intelligently use the components obtained from functional partitioning to extend a state-of-the-art NLIDB system to produce accurate and semantically meaningful query completions in the absence of query logs. We test the proposed query completion framework on multiple benchmark datasets and demonstrate the efficacy of our technique empirically.

1 Introduction

The omnipresence of mobile devices coupled with recent advances in natural language processing capabilities has resulted in a growing number of natural language interfaces [Popescu *et al.*, 2003; Li and Jagadish, ; Saha *et al.*, 2016] enabling naive users to interact with the data without the need of knowing technical query languages such as SQL or even the exact schema of the data.

Query completion systems seek to provide intelligent suggestions to complete partial queries as a user is typing her query. Typical users can view these query suggestions while typing on the fly and can either select one of these suggestions or simply keep typing. Such a feature has several practical benefits. Firstly, the user does not need to type the rest of the query if she finds her intended query in one of the options. Secondly, such auto-completions provide confidence to the user that the system will be able to answer those complete queries since these suggestions are generated by the system itself. This becomes even more important for NLIDB systems because they are domain specific applications and often the end user may not be completely aware of the domain schema to write complete queries.

While the problem of query completion has been studied very well in the context of IR systems working with keyword queries[Baeza-Yates *et al.*, 2004; Cao *et al.*, 2008; Bhatia *et al.*, 2011], for NLIDB systems the aim is quite different. Going beyond just keyword queries, here query completion requires generating semantically correct and complete queries which must be answerable in that domain. To the best of our knowledge, for the first time we are designing and evaluating a query completion framework in the context of NLIDB systems. Most often NLIDB systems are deployed in BYOD (Bring Your Own Data) settings to enable users to search their own data. Therefore, here we consider the problem of query completion for NLIDB systems without assuming the availability of any query logs.

In this paper, we present an ontology-driven approach for query completion where we build our system on top of a state-of-the-art domain-specific NLIDB system called ATHENA [Saha *et al.*, 2016]. The contribution of this paper is as follows:

- We present a novel ontology-based query completion algorithm for NLIDB systems.
- We propose a novel ontology partitioning technique called functional partitioning which can be employed to find semantically close and most relevant queries.
- We are the first to measure the performance of (any) query completion on NLIDB systems with relevant metrics and subsequently demonstrate the efficacy of our technique on 6 varied benchmarks.

2 Background and Motivation

In this section we first present a background on ontology and ontology driven interpretation building. Next, we also describe the common challenges associated with query suggestion on NLIDB systems as a motivation for our present work.

2.1 Notations

We follow the W3C OWL standard definition for ontologies [W3C, 2009] where an ontology defines a set of representational primitives with which to model a domain of knowledge. We denote the set of classes or concepts as C, the set of

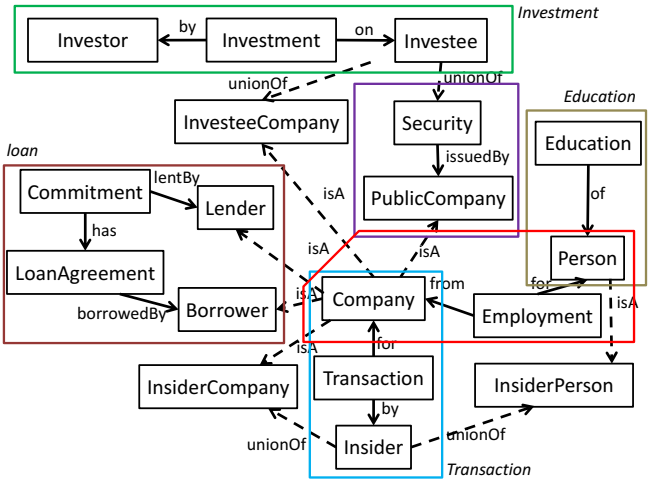

Figure 1: Partial Finance Ontology with Functional Partitioning

properties as P, and the set of relationships as R. A relationship (or a relation, in short) defines a association between a pair of classes $r_k = (c_i, c_j) \in R$. A semantic graph $G = (V, E)$ is a directed graph constructed from an ontology where $V = C$ and $E = R$ i.e. edges exist between concepts taking part in a relation. One such graph (along with components as discussed in Section 3.1) is shown in Figure 1 corresponding to a fragment of financial domain ontology commonly used in the industry. As can be seen in Figure 1, the relations are of three types, namely *membership*, *inheritance*, and *functional*. For membership and inheritance, the edge direction is from parent concept to child (or member) concept. Examples include *Company* $\rightarrow$ *Lender* or *Insider* $\rightarrow$ *InsiderCompany* etc.. For a functional relation, the direction of edges is from concept with cardinality n to a concept with cardinality 1. For example, *Commitment* $\rightarrow$ *Lender* implies one lender can perform multiple commitments and thus the edge direction is from `Commitment` to `Lender`. For relationships like *employment* which is an $m : n$ relationship between `Person` and `Company`, the semantic graph contains an intermediate concept generating two $1 : n$ type relationship as *Employment* $\rightarrow$ *Company* and *Employment* $\rightarrow$ *Person*.

2.2 Ontology-driven NLQ Interpretation

We take a state-of-the-art NLIDB system ATHENA[Saha *et al.*, 2016] as a reference to discuss the general algorithm for ontology-driven natural language query (NLQ) interpretation which forms the basis of many existing approaches [Popescu *et al.*, 2003; Li and Jagadish,].

The algorithm first produces *evidence* that one or more ontology elements (concept, relation, property) have been referenced in the input NLQ. Formally, an *evidence* $v_i : t_i \mapsto E_i$ maps a token t_i (a word in the NLQ text) to a set of ontology elements $E_i \subseteq \{C \cup R \cup P\}$ called *candidates*. In general a token can match multiple elements in the ontology. For example, the token "year" is mapped to properties `Transaction.purchase_year` and `LoanAgreement.period_of_report_year` amongst others.

Next a set of *selected sets* is computed from the *Evidence Set V*. A *selected set* (*SS*) is formally defined as $SS = \{(t_i \mapsto e_i) \mid \forall (t_i \mapsto E_i) \in V, \exists e_i \in E_i\}$, and formed by iterating over all evidences $(t_i \mapsto E_i)$ in V and collecting a **single** ontology element (e_i) from each evidence's candidates (E_i). Thus *selected set* (*SS*) generated from an *Evidence Set V* stands for a specific mapping from a token to a single ontology element. For each *selected set* typical approaches [Tata and Lohman, 2008; Saha *et al.*, 2016] use Steiner Tree based algorithms to generate the most compact subgraph connecting all the elements in the *selected set* in the ontology graph and thus producing a single interpretation per *selected set*. Along with ontology-based interpretation, any NLIDB system such as ATHENA also employs annotators to tag tokens for clauses such as SELECT, WHERE, GROUP BY, ORDER BY etc.. The interpretation and annotations together produce a unique target SQL query.

2.3 Challenges in Query Completion on a NLIDB system

Given a partial query, a possible way to suggest completions is to build the semantic graph corresponding to the ontology element matches from the partial query and then extend the semantic graph by including more ontology elements. The major challenge here is to select which elements to include to expand the partial query semantic graph. This challenge is illustrated using the example with Figure 2.

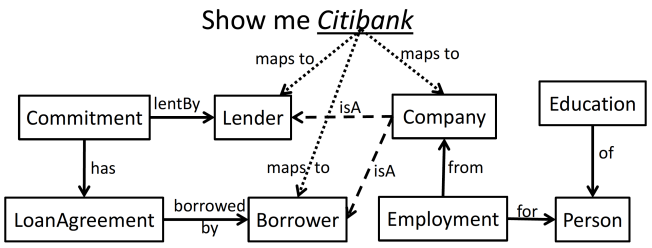

Figure 2: Query suggestion from Partially Typed User Query

For the partial query "Show me Citibank" (see Figure 2), the token "Citibank" maps to `Company.Name` and its many other child concepts' `Name` property. Using the map from "Citibank" to `Lender.name`, concepts like `LoanAgreement`, `LoanCommitment` can be reached which can be used to extend the partial query subgraph for a possible query completion *Show me Citibank's loans to Caterpillar*. Similarly, it is also possible to generate queries like *Show me Citibank employees degrees* by exploring the path `Company- Compensation- Person- Education`. However, this is more complex query involving a sub-query to know Citibank's Employees first before knowing their degrees. Even though both the queries involve exactly 4 concepts and 3 edges, they differ in their inherent complexity. This explains why just a simple graph traversal to include new concepts within a certain number of edges is not good enough for identifying complexity levels of suggested queries and motivates the use of partitioning the ontology to identify exploration boundaries dynamically.

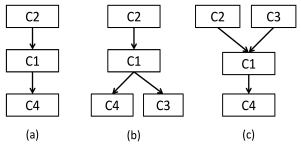

Figure 3: Examples for conditional Merge rule

Algorithm 1: Finding Functional Partition

Input: Semantic Graph
Output: $\mathcal{P} = \{C_i, \ldots, C_n\}$
Initialize $\mathcal{P} = R_F$
// Computation of Base components
foreach $c_i \in ConceptSet$ **do**
  // apply merge rule Merge $(c_i, c_j), (c_i, c_k)$ to C_i if $\{(c_i, c_j), (c_i, c_k)\} \subseteq R_F$
while(some change in $\mathcal{P}$)
// apply conditional merge rule **foreach** $c_i \in ConceptSet$ **do**
  Add $\{(c_i, c_j)\}$ to C_n, if $\forall c_k, (c_k, c_i) \in C_n$.
// Computation of derived components
foreach $C \in P$ **do**
  $C' = C$
  while *no change in* C' **do**
    foreach $(p,c) \in R_I, (c_k, p) \in C'$ **do**
      $C' = C' \cup \{(p,c)\}$
    foreach $(u,m) \in R_M, (c_k, m) \in C'$ **do**
      $C' = C' \cup \{(u,m)\}$
  $P = P \cup \{C'\}$

3 Proposed System and Algorithms

Our proposed system is based on a novel concept of functional partitioning of ontology introduced in this paper. In this section, first we focus on formally defining functional partitioning and then we show how to use it in doing query completions for NLIDB systems.

3.1 Definition of Functional Partitioning

We start with defining the basics over which the notion of functional partitioning is formally defined and then we present the algorithm for computing such a partition.

Definition 3.1 *Uniquely Identifies(ui): The Relation "uniquely identifies (ui)" is defined on an ordered pair of concept C1 and C2, where ui(C1,C2) is true iff a unique instance of C2 can be inferred from any instance of C1.*

This is typically the case if C1 and C2 is having a $n:1$ relationship, where an instance of C1 can uniquely identify the corresponding related instance of C2. Therefore, following the convention from Section 2.1, $ui(C1, C2)$ is equivalent to $(C1, C2) \in R_F$.

Corollary 3.0.1 *Relation ui is transitive.*

This follows from the definition.

Corollary 3.0.2 *Transitive closure of ui produces a partition in R_F where the components may not be disjoint.*

This can be seen in Figure 3(c) where the transitive closure from concept C2 is {C2,C1,C4} and the transitive closure from concept C3 is {C3,C1,C4}, which are not disjoint. Next we formally define functional partitioning with specific rules.

A functional partition P divides the functional relations R_F of an ontology into disjoint sets, called components, subject to the following criteria:

- Merge rule: All outgoing functional relationship of a concept belong to the same component. Formally, $\{(c_i, c_k), (c_i, c_j)\} \subseteq C$, $\{(c_i, c_k), (c_i, c_j)\} \in R_F$.
- Conditional merge rule: If all incoming functional relationship of a concept belong to the same component then its outgoing functional relationships are also in the same component. Formally, $(c_i, c_j) \in C$, if $\{c_k | (c_k, c_i) \in R_F\} = \{c_l | (c_l, c_i) \in C\}$.
- Minimality: Component size should be minimal. In other words, maximal number of components should be formed based on the above rules.

As seen in Figure 1, functional partitioning produces different components for each of the 6 domain functionalities viz. "Loans", "Employment", "Investments", "Transactions", "Security" and "Education', while each has its own disjoint set of functional relations.

Merge rule: As mentioned earlier, a component is formed by taking related functionalities together. The decision of including the relations (c_i, c_j) and (c_i, c_k) into a single component follows from the fact that an instance of c_i can uniquely identify instances of both c_j and c_k, so they must be related. The two functional relationships between `Commitment`, `LoanAgreement`, `Lender` in Figure 1 is such an example.

Conditional merge rule: Conditional merge rule considers incoming functional relationships to see which of the components can be merged. It states if the incoming functional relationships are all in the same component C then the outgoing functional relationships should also be in C. There can be different possible instances of this rule as shown in Figure 3. In the first case (a), there is only single incoming edge and therefore (c_2, c_1) and (c_1, c_4) are in the same component. In our running example, this is seen for relationships between `Commitment`, `LoanAgreement`, and `Borrower`. The second case in Figure 3(b) also makes the incoming relationship component same as the outgoing relationship's component. The third case (c) considers the general case where there are multiple incoming edges. If both the incoming relationships are not in the same component the outgoing relationship's component cannot be merged with either of the incoming relationship's component.

Consider the relationships between `Employment`, `Person`, and `Education`. Both the relationships cannot be put to the same component by either of the above rules. And therefore, because of the minimality constraint, they are kept in the different component. The intuition again follows the uniqueness criteria. In this case, an instance of *Person* can be related to multiple instances of *Employment* and *Education*. Thus one instance of *Employment* can not uniquely identify the instances of *Education* or vice versa.

Note that the merge rule and conditional merge rule work on the functional relationship. The parent (or union) concept are not part of same components. This is intuitive too as each child concept is involved in a specialized functionality that applies exclusively to itself and not with its parent. Our algorithm first identifies a set of components, called base components, following the rules presented above on the semantic graph. Then for each base component, a derived component

Algorithm 2: Generating Suggestions using Functional Partitioning

Input: Partial Query pQ, Ontology O, Annotations $A : t_i \mapsto OQL_{clause}$, Partition $\mathcal{P} = \mathcal{C}_1, \ldots, \mathcal{C}_n$
Output: $Qs = \{Qs_1, Qs_2, ..., Qs_n\}$

1 //Initialize the set of suggestions $QSs = \{\}$
2 //generate evidence set form Partial Query
3 Evidence Set $V = \{(t_i \mapsto E_i) \mid t_i \in pQ, E_i \in O\}$
4 //generate selected sets from the tokens in partially typed query
$SS = \{(t_i \mapsto e_i) \mid \forall (t_i \mapsto E_i) \in V, \exists e_i \in E_i\}$
5 Component [] $targetComponents = \{\}$
6 // compute the set of target base components for level I completion of pQ
7 **foreach** *selected set* $ss \in SS$ **do**
8 **if** $\exists t_i$ *such that Annotation* $(t_i) = Query_{select}$ **then**
9 **foreach** *Component* $C_i \in P$ **do**
10 **if** $e_i \in C_i$ **then**
11 $targetComponents = targetComponents + C_i$
12 **else**
13 $targetComponents = \{(C_i \in P) \mid \forall e_i \in ss, \exists C_i \; s.t. \; e_i \in C_i\}$
14 **foreach** *Component* $C_i \in targetComponents$ **do**
15 **foreach** *Concept* $C \in C_i$ **do**
16 **foreach** *Property* $p \in C$ **do**
17 **if** *p is of semanticType Year* **then**
18 ss = ss + p
19 $Annotation(p) \leftarrow \{Query.groupBy, Query.where\}$
20 **if** *p is namedProperty* **then**
21 ss = ss + p
22 $Annotation(p) \leftarrow \{Query.groupBy, Query.where\}$
23 **if** *p is measureProperty* **then**
24 ss = ss + p
25 $Annotation(p) \leftarrow \{Query.select\}$
26 Default:
27 ss = ss + p
28 $Annotation(p) \leftarrow \{Query.select\}$
29 InterpretationTree $ITree_{ss} \leftarrow$ Generate Interpretation(ss)
30 **foreach** *AnnotaionMap* a_m *of* $e_i \in ss$ **do**
31 oql $\leftarrow$ Generate OQL using $ITree_{ss}$ and annotation map a_m
32 query $\leftarrow$ Generate a natural language query using a_m on ITree
33 $Qs = Qs + query$
34 return Qs

is formed by additionally including inheritance and membership relations from parent to child.

The algorithm for finding a functional partition is presented in Algorithm 1. Initially, each functional relationship can be considered as a single component. The algorithm is essentially a fixed point algorithm which iteratively applies the rules till there is no change found in the partition.Note that the merge rule can be applied only once for each concept as its repeated application will not change the partition. That optimization is performed in the first step of the algorithm in Lines 3-4. As the application of conditional merge rule merges two partitions it has to applied repeatedly until there is no change in the partition (Lines 5-7). Finally, each base component, identified by the last step, is extended to include the is-a or membership edges to include the child concepts. The process ends when no more is-a or membership edges can be added (Lines 9-16).

The time complexity of the algorithm is $O(|R_F|^2)$. As already described, functional partitioning purely depends on the ontology structure and so is not affected by poor database design.

Algorithm 3: Finding set of related components for generating multi level query completions

Input: Partial Query pQ, Ontology O, Annotations $A : t_i \mapsto Query_{clause}$, Partition $\mathcal{P} = \mathcal{C}_1, \ldots, \mathcal{C}_n$
Output: $QS_1, QS_2, ..., QS_n$

1 Component [] *targetComponents* = Get target base components from Algorithm 2
2 Component[] *expandedTargetComponents* = {}
3 **foreach** *Component* $c \in targetComponents$ **do**
4 Component[] derivedComponents = $\{DC \in P, DC \text{ is a dervied component} \mid DC \cap c \neq \phi\}$
5 **foreach** *Derived Component* $dc \in DC$ **do**
6 **foreach** *Base Component* $bc \in P$ **do**
7 **if** $dc \cap bc \neq \phi \parallel c \cap bc \neq \phi$ **then**
8 *expandedTargetComponents* = *expandedTargetComponents* + $\{bc\}$
9 **foreach** *Interpretation Tree Itree built across components in expandedTargetComponents* **do**
10 Q = Build Query from ITree
11 Assign level of Q $\leftarrow$ number of components involved in Itree
12 Qs = Qs + {Q}

3.2 Query Completion

In this section, we first propose the criteria of a good query completion system on NLIDB that can address the challenges mentioned in Section 2.3. Then we formally design functional partition based algorithms to meet the criterion. Below we capture the criteria of desired query completions.

Multi Level Query Completions:

We formally define the complexity *level* of a query as the number of additional components needed to complete the query beyond the components needed to capture the partial query. By this definition, for the partial query "*show me Citibank*", both the query completions "*Show me Citibank Loans*" and "*Show me Citibank Employees*" are in *level I* involving only a single component of Loans and Employment respectively. Whereas "*Show me Citibank employees degrees*" is in *level II* spanning two components of `Employment`, `Education`. Moreover, the complexity level of a query completion is defined over the partial query entered. So the same query completion "*Show me Citibank employees degrees*" will actually be in *level I* for the partial query "*Show me Citibank employees*". Thus for a given partial query, the idea of Multi-level query completion is to divide the complete set of possible query completions into different levels as per their complexity and then provide query completions only till a certain level to address the challenge of generating only a precise and relevant set of query completions among combinatorially huge number of possible completions.

Algorithm 2 presents the approach that uses functional partitioning to make *level I* query completions for a partial query. Line 3- 13 analyzes the *evidence sets* (as described in Sec 2.2) from the partial query to identify the components referred by the partial query. Note that, if the partial query already includes an evidence with select annotation, that evidence alone is used to find the right component for the partial query(Line 11). The identified components are used to find ontology elements to extend the semantic graph obtained from the partial query. The domain semantics captured in the ontology are used to assign most likely annotations for

	Partition Characteristics				
Ontology	**#Base**	**#derived**	**Time(in ms)**	**#Unique**	**#Across**
FIN	48	72	32.3	68(91%)	7(9%)
MAS	11	11	5.7	71(36%)	125(64%)
IMDB	10	18	4.5	101(79%)	27(21%)
Yelp	3	3	2.3	115(88%)	16(12%)
Software	38	50	38.1	50(94%)	4(6%)
Institution	37	49	19.2	49(92%)	4(8%)

Table 1: Summary statistics

the extended elements (Line 17- 28). For example, a property having semantic type "year" can be used in queries as *Where* conditions e.g. in 2011 or they can also be used in *GroupBy* clause e.g. by year. The interpretation tree obtained from the extended semantic graph together with the assigned annotations produces a unique query interpretation(Line 31). Finally, Line 32 translates each interpretation to produce natural language query completions.

While Algorithm 2 presents an approach to suggest only *level I* queries, Algorithm 3 extends it further to suggest query completions spanning across multiple partitions and thus belonging to *Level II* and so on. The key part here is to use derived components (Line 4-Line 8) to find possible common join concepts between two disjoint base components. So queries like "show me investments in Companies having loans from Citibank" can now be produced by combining Loans and Investments via Company as the join concept.

4 Experiments

In this section, we experiment with 6 publicly available ontologies to test the efficacy of functional partitioning based approach for query completion. The chosen ontologies are (1)FIN (2)MAS (3)YELP (4)IMDB (5)Software (6)Institution. Among the chosen ontologies FIN, MAS, IMDB, YELP have already been widely used in previous NLIDB papers such as NALIR [Li *et al.*, 2005], ATHENA [Saha *et al.*, 2016], SQLizer [Yaghmazadeh *et al.*, 2017]. In addition to that, we chose two other publicly available ontologies Institution [ins, 2014] and Software [sof, 2012]. The set is so chosen that it includes complex ontologies like FIN, Software, Institution with a much richer set of relationships including multiple isA and union and also relatively simple ontologies like IMDB, Yelp or MAS. For each of the ontologies, we employ a set of query workload answerable over the ontology. For existing datasets like FIN, MAS, IMDB, Yelp we used the same publicly available query workload as was used in their respective publications [Saha *et al.*, 2016; Li *et al.*, 2005; Yaghmazadeh *et al.*, 2017], whereas, for Software and Institution we asked a group of users familiar with ontology to create answerable queries for that domain. Table 1 tabulates the time taken for computing partitions and also other important statistics we obtain while applying functional partitioning for each of the ontologies and their query workloads.

As seen in Table1 FIN being the largest ontology with 75 concepts produces 48 base components and 72 derived components, followed by Software and Institution which are the next two biggest ontologies. For all the ontologies except MAS, most of the queries are within a unique single partition, which implies that they can be suggested as *Level I* queries itself. The only exception MAS has a number of queries on join tables between multiple components, resulting in comparatively poor coverage with single components.

4.1 Experimental Setup

We generate partial queries from each of the queries in the query workloads across ontology by considering prefixes at different positions. The prefix positions are varied in three levels viz P1: Prefix query with only the first evidence keyword, Pn: Prefix query with all the evidence keywords except the last one and Pmid: Prefix query with more keywords than P1 but less than Pn. For each partial query, we treat the complete queries as the target query that the auto-completion should be able to suggest.

To evaluate functional partitioning based query completion (hereafter called as QC1), we compare it with a baseline approach (hereafter called QC2) that is motivated by the existing works on ontology based query expansion [Wu *et al.*, 2011] in IR domain where the common idea is to use ontology edges for extending the semantic graph. Because QC2 is unaware of any partition boundaries, it uses a distance threshold to choose candidate ontology elements for extending the semantic graph. For QC1, as already defined, the number of components needed for the complete query defines the level for query completion. To have a fair comparison with QC1, the level for QC2 is defined as follows: if the distance between farthest nodes in a query completion q_c from QC2 is less than the average component size for that ontology, q_c is considered as the level I query for QC2, distance less than twice the average component size is *level II* and so on.

Evaluation Results

We first define the most relevant metrics to evaluate auto-completion in NLIDB system and then compare QC1 and QC2 with respect to each of them. The common setup for all the defined metrics has a partial query q_p and a query completion system like QC1 or QC2 suggests a set of query completions with that partial query as prefix i.e. $QS_c = \{q_c \mid q_c$ has q_p as prefix$\}$. The metrics are all defined on QS_c.

Recall: Given a partial query q_p, if QS_c contains the actual query intended by the user, the query completion system generating QS_c has a recall value 1 for that partial query q_p, otherwise 0. The recall value of the benchmark dataset can be computed as the average recall value over the set of benchmark partial queries created for that domain.

The recall values presented in Figure 4 are computed considering P_1 as the prefix query. The task of suggesting query completions from P1 is most challenging because P1 with only the first evidence keyword has the least available information about the intended query. As seen in Figure 4, for all the ontologies Level I recall is higher for QC1 than QC2 and also most of the queries can be recalled in Level I queries itself. Although the recall value is seen to increase from level I to level III, it is equally important to compute the total number of suggestions that the system had to make to the user in order to produce the correct one. So next we compute the Average Suggestion Rank for both QC1, QC2.

Average Suggestion Rank: Given a partial query q_p and the actual intended query from the user being q_i, suggestion

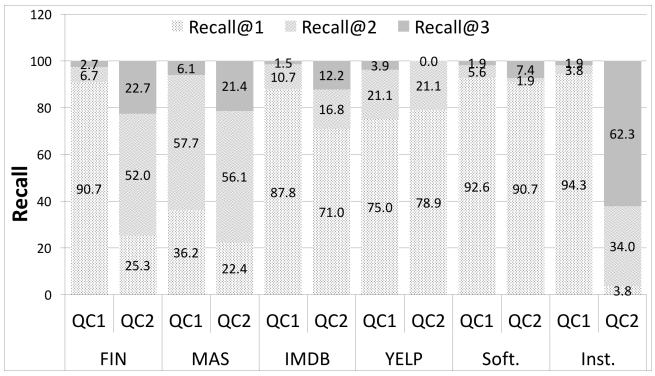

Figure 4: Recall Values

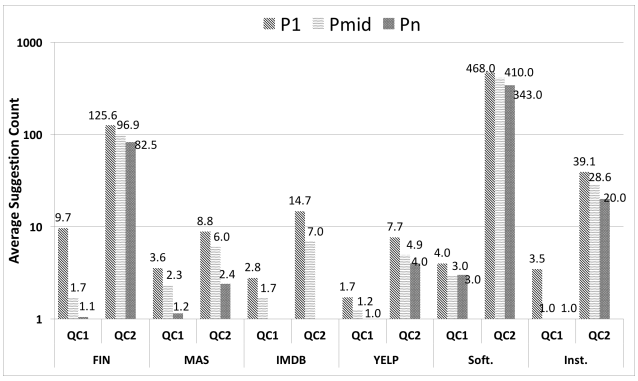

Figure 6: Average Suggestion Count across Multiple Prefix

rank for q_i is the position of q_i in the set of suggested query completions QS_c. Average suggestion rank for a data set of queries is defined as the average taken over suggestion rank values found for all possible partial queries generated from queries in the dataset.

Average Suggestion Rank is key to differentiate between two query completion systems both having a good recall but only one of them making more relevant and precise suggestions, while the other may suggest all possible queries.

As seen in Figure 5 QC1 always has a significantly lesser number of suggestion count than QC2 for all datasets and recall levels. However, the average suggestion count at recall level 2 for QC1 is around 14, much higher than the recall level 1 average which was 4. For a general system with arbitrarily large ontology, the difference can even increase more. Thus it is only fair for a query completion system to work only with recall level 1.

As the recall level depends also on the partial query(Section 3.2), we study the effect of a partial query in generating query completion suggestions in Figure 6. There we tabulate the Average Suggestion Rank values for recall level I by varying the prefix from P_1 to P_n. An important observation from Figure 6 is that as the user keeps typing more keywords, QC1 becomes more precise and suggests lesser number completions to the user.

Usability Score: This metric is aimed at measuring the user perception in terms of how much a query completion mentions the complete intent explicitly and unambiguously. The usability score for a query completion q_c from a partial query q_p is defined as either 1 or 0 depending on if q_c explicitly mentions values for all the filters applicable to that specific intent of the query. For example, considering the partial query q_p as "show me Citibank", a query completion q_c^1 = "show me Citibank's loans to caterpillar in last 5 years" can have usability score as 1 but not query completions like q_c^2 = "show me Citibank's loans" or q_c^3 = "show me Citibank's loans in last 5 years". Because this metric is more subjected to user perception, the scores to compare QC1 and QC2 are collected by doing a user survey, while a set of users scored each of the suggested query completion as 0 or 1 depending on their perception of complete and unambiguous suggestion of queries.

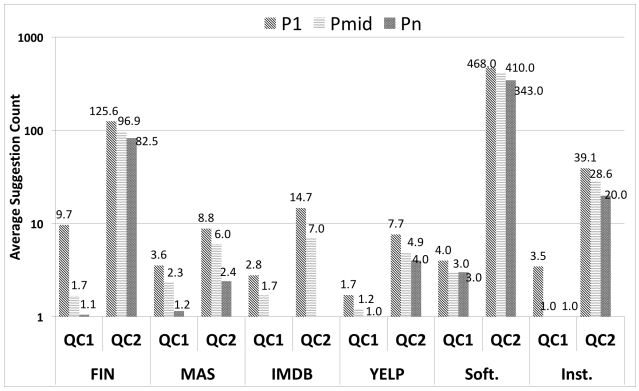

Figure 5: Average Suggestion Count across multiple Recall Levels

Table 2 shows Usability Score is also higher for QC1 than QC2. QC1 being aware of component boundaries can provide complete suggestions for each component.

Both QC1 and QC2 uses an interpretation over the ontology graph to suggest a query completion. Therefore, every suggestion has some associated meaning in domain ontology and so a valid query. Thus precision although is an important metric for query completion in IR systems, is not a differentiator between QC1 and QC2.

The results demonstrate that our query completion system can suggest precise query completions and in most cases can retrieve the intended query by the user. Also the number of provided suggestions are reasonably low to make the system practically usable.

5 Related Work

In this section, we discuss related works mainly along two categories (i) Query suggestions (ii) Ontology Partitioning

	Usability Scores	
Ontology	$\|QC1\|$	$\|QC2\|$
FIN	0.91	0.83
MAS	0.64	0.36
IMDB	0.88	0.67
Yelp	0.79	0.32
Software	0.98	0.94
Institution	0.92	0.42

Table 2: Usability Score the experimental ontologies

Query suggestion While query auto-completion and query suggestion is a well-studied problem in information retrieval, most of the existing works tend to fall into two categories. They either rely on a history of query logs to generate these suggestions [Baeza-Yates *et al.*, 2004; Cao *et al.*, 2008] or they attempt to build a model of queries by analyzing a corpus in the domain of interest [Bhatia *et al.*, 2011; Meij *et al.*, 2009]. Some recent work tries to do ontology based query suggestion. While [Song, 2015] suggests manually building a grammar from the ontology to guide the query suggestion process, [Franconi *et al.*, 2010] use drill-down like intelligent UI features to explore all possible queries, disregarding the ranking problem altogether.

Ontology Partitioning In the existing literature, ontology partitioning is utilized to achieve the goal of modularization and help with maintenance, validation, publication, and processing of large ontologies. Thus are mostly focused on analyzing structural similarities to infer possible partitions. Most prominent works try to partition the ontology graph by analyzing the connection densities[Michel and Stuckenschmidt, 2004; Schlicht and Stuckenschmidt, 2007]. Other approaches considering structural similarity, linguistic features include [Ahmed *et al.*, 2015; Zhang *et al.*, 2011; Etminani *et al.*, 2010].

6 Summary and Future Work

In this paper, we present a novel ontology partitioning approach called functional partitioning and use it to build a query completion framework on top of a state-of-the-art ontology based NLIDB system ATHENA. We experimentally demonstrate that functional partitioning leads to precise and accurate query completions for NLIDB systems even in the absence of query logs.

There are two separate directions of future work - one involving the functional partitioning based mechanism to obtain follow up query recommendations and other involving utilization of query logs (when available) to further improve on the query completion system proposed in this work.

References

[Ahmed *et al.*, 2015] Soraya Setti Ahmed, Mimoun Malki, and Sidi Mohamed Benslimane. Ontology partitioning: Clustering based approach. In *I.J. ITCS*, pages 1–11, 2015.

[Baeza-Yates *et al.*, 2004] Ricardo Baeza-Yates, Carlos Hurtado, and Marcelo Mendoza. Query recommendation using query logs in search engines. EDBT'04, pages 588–596, Berlin, Heidelberg, 2004. Springer-Verlag.

[Bhatia *et al.*, 2011] Sumit Bhatia, Debapriyo Majumdar, and Prasenjit Mitra. Query suggestions in the absence of query logs. SIGIR '11, pages 795–804, New York, NY, USA, 2011. ACM.

[Cao *et al.*, 2008] Huanhuan Cao, Daxin Jiang, Jian Pei, Qi He, Zhen Liao, Enhong Chen, and Hang Li. Context-aware query suggestion by mining click-through and session data. In *KDD*, KDD '08, pages 875–883, New York, NY, USA, 2008. ACM.

[Etminani *et al.*, 2010] K. Etminani, A. Rezaeian Delui, and M. Naghibzadeh. Overlapped ontology partitioning based on semantic similarity measures. In *5th International Symposium on Telecommunications (IST), 2010*, pages 1013–1018, 2010.

[Franconi *et al.*, 2010] Enrico Franconi, Paolo Guagliardo, and Marco Trevisan. Quelo: A nl-based intelligent query interface. 622, 01 2010.

[ins, 2014] InstituteOntology. `http://www.isibang.ac.in/~bisu/ontology/instOntology.owl`, 2014.

[Li and Jagadish,] Fei Li and H. V. Jagadish. Constructing an Interactive Natural Language Interface for Relational Databases. *VLDB,2014*.

[Li *et al.*, 2005] Yunyao Li, Huahai Yang, and H. V. Jagadish. NaLIX: An Interactive Natural Language Interface for Querying XML. In *ACM SIGMOD*, 2005.

[Meij *et al.*, 2009] Edgar Meij, Marc Bron, Laura Hollink, Bouke Huurnink, and Maarten Rijke. Learning semantic query suggestions. ISWC '09, pages 424–440, 2009.

[Michel and Stuckenschmidt, 2004] Klein Michel and Heiner Stuckenschmidt. Structure-based partitioning of large concept hierarchies. ISWC, 2004.

[Popescu *et al.*, 2003] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a Theory of Natural Language Interfaces to Databases. In *IUI*, 2003.

[Saha *et al.*, 2016] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R. Mittal, and Fatma Özcan. Athena: An ontology-driven system for natural language querying over relational data stores. *Proc. VLDB Endow.*, 9(12):1209–1220, August 2016.

[Schlicht and Stuckenschmidt, 2007] Anne Schlicht and Heiner Stuckenschmidt. Criteria-based partitioning of large ontologies. In *International Conference on Knowledge Capture*, K-CAP '07, 2007.

[sof, 2012] SoftwareOntology. `http://se-on.org/`, 2012.

[Song, 2015] Dezhao Song. Tr discover: A natural language question answering system for interlinked datasets. In *International Semantic Web Conference*, 2015.

[Tata and Lohman, 2008] Sandeep Tata and Guy M. Lohman. SQAK: Doing More with Keywords. In *ACM SIGMOD*, 2008.

[W3C, 2009] W3C. `http://www.w3.org/TR/owl-guide/`, 2009.

[Wu *et al.*, 2011] Jiewen Wu, Ihab F. Ilyas, and Grant E. Weddell. A study of ontology-based query expansion. 2011.

[Yaghmazadeh *et al.*, 2017] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017.

[Zhang *et al.*, 2011] Liang Zhang, Kun Liu, Xue Qin, and Shengqun Tang. Extracting module from owl-dl ontology. In *ICSEM*, volume 1, pages 176–179, Oct 2011.