Query Driven Hypothesis Generation for Answering Queries over NLP Graphs

Chris Welty[1], Ken Barker[1], Lora Aroyo[2], Shilpa Arora[3]

[1] IBM Watson Research Center
cawelty@gmail.com, kjbarker@us.ibm.com
[2] VU University Amsterdam
lora.aroyo@vu.edu
[3] Carnegie Mellon University
shilpaa@cs.cmu.edu

Abstract. It has become common to use RDF to store the results of Natural Language Processing (NLP) as a graph of the entities mentioned in the text with the relationships mentioned in the text as links between them. These NLP graphs can be measured with Precision and Recall against a ground truth graph representing what the documents actually say. When asking conjunctive queries on NLP graphs, the Recall of the query is expected to be roughly the product of the Recall of the relations in each conjunct. Since Recall is typically less than one, conjunctive query Recall on NLP graphs degrades geometrically with the number of conjuncts. We present an approach to address this Recall problem by hypothesizing links in the graph that would improve query Recall, and then attempting to find more evidence to support them. Using this approach, we confirm that in the context of answering queries over NLP graphs, we can use lower confidence results from NLP components if they complete a query result.

1 Introduction

Semantic web technologies like RDF and OWL are increasingly becoming desired tools for Natural Language Processing (NLP), for example to store the results of information extraction from documents. As discussed in [3], it is quite natural to store NLP results as a graph, and required elements of an NLP stack to produce RDF graphs are named-entity recognition, relation extraction, coreference resolution, and knowledge integration. Traditionally, the components of an NLP stack simply feed the knowledge integration layer as the last step of a pipeline; we have for several years been exploring the use of semantic technology to influence the NLP stack as well. In this paper, we discuss the particular problem of evaluating SPARQL queries over RDF graphs that store the results of NLP, we call these *NLP graphs*.

The simple step from special-purpose formats and storage of NLP results to using RDF immediately introduces the advantage of a well developed query language (SPARQL), and this itself quickly exposes a problem in the combination: as more complex queries are asked of the results, Recall becomes the dominating factor in performance. In particular, for queries that are a conjunction of terms, the overall Recall can be severely degraded by term Recall. In general we expect conjunctive query Recall to

be roughly the product of the Recall of the relations in the conjuncts, in the worst case, query Recall can be zero even for non-zero term Recall if the variable bindings over individual terms do not overlap. For example, consider the conjunctive query to find Jordanian citizens who are members of Hezbollah:

```
SELECT ?p
WHERE {
   ?p mric:citizenOf geo:Jordan.
    mric:Hezbollah mric:hasMember ?p .
}
```

Even if the Recall of each term (`citizenOf`, `hasMember`) is non-zero, their bindings for the variable `?p` must overlap in order for the query Recall to be non-zero. Roughly speaking, we can estimate conjunctive query Recall as the product of the Recall of the relations and the probability that triples share arguments. An NLP graph can be analyzed to produce such estimates, and our empirical analysis supports this model, however our purpose is not to model it accurately, but to provide an explanation for our observations that conjuncive query Recall on NLP graphs is unacceptably low, and to provide a solution.

We describe a system that uses solutions to subsets of a conjunctive SPARQL query as candidate solutions to the full query, and then attempts to confirm the candidate solutions using various kinds of inference, external resources, and secondary extraction results. Previously we showed the hypothesis generation and validation technique is a promising way of incorporating background knowledge and reasoning in NLP graphs [1]. In this paper we extend those results by considering evidence from the secondary hypotheses produced by the components in the NLP stack that are traditionally discarded, and show they can improve Recall while also significantly improving F-measure.

2 Background

Our task is drawn from the evaluation scenarios used for DARPA's Machine Reading program (MRP)[4]. We were given a target ontology in OWL of types and binary relations, a corpus of $10,000$ documents taken from the Gigaword corpus, and a set of 79 documents manually annotated with mentions of the target relations and their argument types. The goal was to find mentions of the types and relations from the ontology in the corpus, and extract them into an RDF Graph. The MRP evaluation scenario consisted of 50 queries in SPARQL that were expected to be answered over this NLP Graph. The query results were evaluated manually by an evaluation team. Each query was supposed to have at least one correct answer in the corpus, some queries had over $1,000$ correct answers.

In order to provide more statistical significance in evaluating our system, we made a few modifications to this evaluation scenario (see section 2.3). In section 2.1 we present our basic NLP pipeline, and in section 2.2 we present our query-answering solution.

[4] http://www.darpa.mil/Our_Work/I2O/Programs/Machine_Reading.aspx

2.1 NLP Stack

The NLP Stack contains tokenizers, parsers, coreference resolution and entity disambiguation and matching modules, and entity and relation extractors trained on the ontology and text in the domain. Its main component is the RelEx relation extractor, an SVM learner with a string-subsequence kernel defined based on the context of each entity mention pair in a sentence [2]. An entity mention pair forms an instance that is *positive* if a given relation exists between them, and *negative* otherwise; there is no representational difference between explicitly negated relations and not stating the relation (e.g. "Joe is not married to Sue" and "Joe is employed by Sue" are both negative examples of the *spouseOf* relation). The output score from the SVM is converted to probabilities/confidences for each known relation that the sentence expresses it between the pair of mentions.

The NLP Stack is run over the target corpus producing an 'extraction graph', in which graph nodes are *entities* (clusters of mentions produced by coreference) and graph edges are either type statements between entities and classes in the ontology, or relations detected between mentions of these entities in the corpus.

The NLP stack produces two extraction graphs, the *primary graph* which contains the single best type, relation, and coreference results for each mention or mention pair, and the *secondary graph*, which contains all possibilities considered by the NLP stack. For example, in the case of relation detection, the system produces, for each pair of mentions in a sentence, a probability that each known relation (and 'no relation') holds, and produces only the highest probability relation in the primary graph. In many cases, the losing probabilities are still quite high, as they are computed independently, and may represent quite useful extraction results. We keep these secondary results for potential exploitation when needed, as discussed later (Section 4).

The graphs produced are special kinds of RDF graphs, which we refer to as *NLP Graphs*. A particular triple in the graph is not an expression of truth, rather it is understood to be a representation of what an NLP component, or a human annotator, read in a document. Attached to each triple is a confidence supplied by system components, and provenance information indicating where the triple was believed to have been stated in natural language. Again, the confidence is not that the triple is true, but reflects the confidence that the text states the triple.

2.2 Query Answering

The queries are a conjunction of terms from the ontology with some relation arguments as variables and some bound. For example, consider the query asking to find all terrorist groups that were agents of bombings in Lebanon on October 23, 1983:

```
SELECT ?t
WHERE {
   ?t rdf:type mric:TerroristOrganization .
   ?b rdf:type mric:Bombing .
   ?b mric:mediatingAgent ?t .
   ?b mric:eventLocation mric:Lebanon .
```

```
   ?b mric:eventDate "1983-10-23" .
}
```

The query answering system must find all bindings for the variable `?t` that satisfy the conjunctive query and also report where in the target corpus the answers are found. In practice, this means finding spans of text ('provenance') expressing the relations in the query.

2.3 Evaluation Queries and Ground Truth

The original MRP evaluation required extensive manual effort to perform because of one basic problem: when RDF graphs are used to store the results of NLP, there is no way to reliably produce identifiers on the nodes in the graph so that they will match the identifiers on the nodes in a ground truth. To address this problem, system results included provenance information in a special format that evaluators would use to find the mention (i.e. document and span) of an entity in a graph. Using the provenance, evaluators semi-automatically mapped the entity IDs from system results to entity IDs in the ground truth. This process proved expensive and error-prone, and was difficult to reproduce, consequently making it difficult to test systems adequately before the evaluation. Due to the cost, only 50 queries were used in the original MRP evaluation, which did not give us a statistically significant way to validate system performance overall, let alone to test parts of the system in isolation.

In order to generate more meaningful experiments over more queries, we had to eliminate the manual entity ID mapping step. To do this, we generated an NLP graph from manually annotated data, which we called a gold-standard graph. From the gold-standard graph, we automatically generated SPARQL queries (this process is described below), ran those queries against the graph and produced query results which we called the gold-standard results. We then measured performance of system results against these gold standard results. It should be clear that since the gold-standard and system NLP graphs both use the same set of node IDs, comparing system query results to gold standard query results is trivial, the only difference between the two graphs (system and gold standard) is the topology of the graph, not the nodes.

The manual annotation took place on a corpus of 169 documents selected from the gigaword corpus, which were completely annotated with types, relations, coreference, and entity names (when they occurred in the text). The corpus was split into two sets of 60 documents for train and devtest, and a 49 document final (blind) test. The split was done in a way that balanced the relative frequencies of the relations in each set (see Table 1). The training set was used to train the relation extraction (RelEx) component on gold standard mentions, their types, and coreference. Gold standard mentions, types, and coreference were also used when applying RelEx to the train, devtest and test sets. Obviously this increases the F-measure of the RelEx output, but our experiments used that output as a baseline. Again, the purpose of this approach was to give us entity IDs from the gold standard in our system output.

The original MRP evaluation queries had at least one answer in the corpus. Our evaluation SPARQL queries were generated from the gold standard NLP graph for each

document set by extracting random connected subsets of the graph containing $2-8$ domain relations (not including `rdf:type`), adding type statements for each node, and then replacing nodes that had no proper names in text with *select* variables. Since the original MRP evaluation queries were not randomly generated, but hand-written ostensibly based on the percieved needs of the domain (national intelligence), we analyzed them and found they tended to be grounded in known (i.e., named) entities, such as "perpetrators of bombings in *Lebanon*" or "spouses of people who attended school in *NY*". To achieve a similar effect in our synthesized evaluation queries, we ensured each query included the id of an entity that was mentioned by name (as opposed to purely nominal or pronominal) at least once in the corpus. We generated 475 test queries for the devtest set and 492 for test. These queries were then run over the same gold standard graph they were generated from – since they had variables the results would be different than what we started with – and the results became our gold standard results for query evaluation.

Although imperfect, our evaluation setup allowed us to run experiments repeatedly over hundreds of queries with no manual intervention. Our choices sacrifice graph size as well as giving up system typing and coreference in the output, which are sources of NLP errors that our hypothesis generation and validation component are capable of correcting.

3 Query-Driven Hypothesis Generation

The primary obstacle for answering conjunctive queries over NLP Graphs is Recall, which degrades geometrically with the number of query terms. To address this problem, we devised a system of hypothesis generation that focuses attention on parts of an NLP Graph that *almost* match a query, identifying statements that if proven would generate new query solutions.

3.1 Generating Hypotheses

The system we developed relaxes queries by removing query terms, then re-optomizes the query and finds solutions to the remaining set of terms, using those solutions to ground the relaxed queries for further analysis. In a sense we are looking for missing links in a graph that, if added, would result in a new query solution. The variable bindings for the relaxed query results give us a place to start. Each solution to a query Q is a single binding over the set of variables V shared among N query terms. We relax Q by removing a query term, leaving a relaxed query Q' having some subset of variables V' shared among the remaining $N-1$ terms. The solutions to Q' provide bindings over V'. We use those bindings to ground all of the original N query terms in Q. These terms of Q grounded using the bindings from solutions to Q' are hypotheses to be tested. Of course, $N-1$ of the hypotheses are already satisfied by the extractions, but since extractions are themselves noisy, we subject them to hypothesis testing, too.

Each set of N query terms is a hypothesis set: if the hypotheses in a hypothesis set can be proven, the hypothesis set constitutes a new solution to Q. If no solutions to subqueries of size $N-1$ are found, we consider subqueries of size $N-2$, and so on

down to subqueries of size $N/2$. (In practice we rarely go lower than $N-2$: the number of hypotheses begins to explode, and they are rarely related enough to be useful). Each relaxed subquery is dynamically optimized based on the number of term solutions and shared variables.

For example, consider again the query from section 2.2 asking to find all terrorist groups that were agents of bombings in Lebanon on October 23, 1983. The system would consider five relaxations at level one, corresponding to the subqueries of length four resulting from ablating each term in turn:

1. find all bombings in Lebanon on 1983-10-23 with agents (hypothesize that the agents are terrorist organizations)
2. find all events in Lebanon on 1983-10-23 by terrorist orgs (hypothesize that the events are bombings)
3. find all bombings in Lebanon on 1983-10-23 (all known terrorist orgs are hypothetical agents)
4. find all bombings by terrorist orgs on 1983-10-23 (hypothesize that the bombings were in Lebanon)
5. find all bombings by terrorist organizations in Lebanon (hypothesize that the bombings were on 1983-10-23)

If no solutions for any of the five subqueries are found, the system will attempt to generate hypotheses for the twenty subqueries of length three, and so on.

One special case is when subquery solutions do not bind a variable. This occurs for subqueries of length $N-k$ for any variables that appear k (or fewer) times in the original query. For example, if the above query produced no solutions to subqueries of length $N-1$, the system would consider subqueries of length $N-2$. At this level, we would produce a subquery in which `?t` does not appear. Solutions to the subquery would provide candidate bindings for `?b`, which would generate `mediatingAgent` hypotheses in which the first argument is bound, but the second is variable. Hypotheses with variable arguments risk generating a large number of query answers improving Recall but degrading Precision.

Finally, we note that generating hypotheses from relaxed subqueries is appropriate for queries that are *almost answerable*. Hypotheses from missing terms will likely only be useful when most of the terms in the query are not missing. Our system does not, however, blindly accept hypotheses. They are only accepted if there is evidence for them and confidence in the evidence above certain thresholds (see 4.3). Nevertheless, this approach is biased toward generating more answers to queries and will likely perform poorly when doing so is not appropriate (for example, queries for which the corpus does not contain the answer).

4 Hypothesis Validation

Each hypothesis set from the hypothesis generator contains hypotheses in the form of RDF statements, which, if added to the primary extraction NLP graph, would provide a new answer to the original query. These hypotheses are not accepted simply because they would provide new answers, rather they are targets for further, deeper processing

that for a variety of reasons was not performed as part of primary NLP stack processing. We call this deeper processing of hypothesis *hypothesis validation*. Only validated hypotheses are added to the query result sets.

4.1 Architecture

All hypotheses are passed to a stack of hypothesis checkers. Each hypothesis checker reports its confidence that the hypothesis holds and, when possible, gives a pointer to a span of text in the target corpus that supports the hypothesis (the provenance). In the current architecture, hypothesis checkers may also choose whether to allow a hypothesis to flow through to the next checker. A significant property of the architecture is that an individual hypothesis checker can be any arbitrary module capable of reporting support for a hypothesis with confidence and provenance, including external IR systems, IE systems, knowledge bases, reasoners or even independent question answering systems.

One of the prime motivations for this architecture is to circumscribe complex or problematic computational tasks, such as formal reasoning or choosing between multiple low-confidence extractions. When isolated to proving, supporting, or refuting individual RDF statements, these tasks can be made more tractable by providing a goal. For example, if we ran a tableau-based reasoner over the primary extraction NLP graph, the reasoner would report inconsistency (see for example [3]) and fail to do anything useful. However, when constrained to only the part of a graph that is connected to a hypothesis, the reasoner may be used effectively. We did not use a tableau reasoner in the experiments described here, it remains future work to put that together with the individual hypothesis checkers discussed below, but the point is the same: complex computational tasks are made more tractable by using hypotheses as goals.

4.2 Hypothesis Checkers

We have tried many hypothesis checking components. For the experiments described here, we used only three:

Primary Extraction Graph. The original primary extraction NLP graph is itself just one of the hypothesis checkers in our stack. By default, this means that the hypotheses corresponding to the $N - 1$ terms not ablated are guaranteed to be supported, with the original extraction confidence and provenance. By including these terms as hypotheses and rechecking them, our system could adjust their confidence using other hypothesis checkers and even reject them if their confidence is low. Other extraction graphs could also be plugged in as checkers (extraction graphs over other corpora, or even a new extraction graph over the same corpus with different NLP stack configurations). For these experiments, however, it was only possible for a hypothesis supported by the primary graph to increase in confidence through support from the knowledge base.

Secondary Extraction Graph. Many existing NLP components consider multiple interpretations of the text during processing and score or rank these interpretations. When used in a pipeline, it is very common to simply take the top interpretation and pass that to the next stage. The individual components are typically tuned to a performance tradeoff that maximizes F-measure. In the case of answering conjunctive queries

over NLP graphs, we needed to explore options that would emphasize Recall, as discussed above. The secondary extraction graph is an RDF NLP Graph generated from *all the interpretations considered by the NLP Stack.* In practice this can include multiple mentions, types on those mentions, multiple entities, multiple types on the entities, and multiple relations between them, and can obviously be quite large. In these experiments we explored only the interpretations considered by the RelEx component, which generates a probability of every known relation holding between every pair of mentions within a sentence. Our secondary graphs are pruned at a particular confidence threshold (or rank), and in our experiments we explored different confidence thresholds. Clearly the secondary graph can have very high Recall, but extremely low Precision: for RelEx the Recall with no threshold should approach 1 and the Precision should approach $1/|R|$ ($|R|$ is the number of known relations), since for each mention pair a probability is generated for each relation and no more than one will be correct in the ground truth. The central hypothesis of this paper is that the secondary graph is a useful source of information when it is used in conjunction with the primary graph to answer a query.

Knowledge Base. A third hypothesis checker is a knowledge base allowing taxonomic inference as well as more complex rules. For our evaluation, the knowledge base contained rules derived directly from the supplied domain ontology. It also contained a small number of general, domain-independent rules (such as family relationships). Rules derived directly from the ontology include:

- simple superclass-subclass rules ($Bombing(?x) \rightarrow Attack(?x)$)
- simple relation-subrelation rules ($hasSon(?x, ?y) \rightarrow hasChild(?x, ?y)$)
- simple relation inverse rules ($hasChild(?x, ?y) \leftrightarrow hasParent(?y, ?x)$)

We also derived about 40 more complex rules automatically from the ontology, based on specialization of the domain or range of subrelations, e.g.
($hasSubGroup(?x, ?y) \& HumanOrganization(?x) \rightarrow hasSubOrganization(?x, ?y)$).

In other experiments [1] we used the knowledge base as a hypothesis checker to confirm, for example, the hypothesis $eventLocation(?b, Lebanon)$ when the extraction graph contains $eventLocation(?b, Beirut)$. In the experiments described here, however, we turned this off as it requires an entity matching step (that maps text strings to knowledge-based IDs in the background geographic knowledge base [5]), which would have had to be run on both the ground truth and the extraction graphs. This step has its own kinds of errors, which we wanted to remove from these experiments.

4.3 Accepting Support for Hypotheses

Each hypothesis checker provides its measure of confidence that a hypothesis holds, making it possible to learn thresholds for accepting or rejecting hypothesis support. This was one of the main reasons for our experimental design: the 50 manually scored queries provided by the program did not give us a satisfactory space to tune the threshold parameters. In the experiments described below, we explored this space over hundreds of queries.

With a secondary graph to support hypotheses, we need two thresholds: one for the primary graph and one for the secondary.

Accepted hypotheses are added to the extraction graph and become part of query results. For hypotheses generated from subqueries of length $N - 1$, each accepted hypothesis is an RDF triple that results in a new answer to a query. For hypotheses generated from subqueries of length $N - 2$ or shorter, hypothesis additions may combine to produce new query answers.

Extending the knowledge in the extraction graph with supported hypotheses may also produce answers to other queries that had no solutions, or produce more answers to queries that did. This does not mean an increase in Recall of course, as the hypotheses, even if supported, might be incorrect. For the current experiments we used fully bound hypothesis sets as complete solutions to queries, and not as additions to the primary extraction graph.

5 Experiments

The main purpose of the paper is to present the hypothesis generation and validation framework for NLP Graphs. The central hypothesis is that within this framework, there is value in the secondary extraction graph for conjunctive query answering. More precisely, it is our hypothesis that the probabily of a secondary graph statement being correct increases significantly when that statement generates a new result to a conjunctive query over the primary graph.

5.1 Experimental Setup

As discussed in 2.3, a manually annotated corpus of 169 documents was split into train, development, and test sets of 60, 60, and 49 documents, split in a way to balance the distribution of the domain relations (see below, however, for notes on this balance). In total there are 48 relation types. Table 1 lists the number of instances of these relations in the train, development and test sets. Since documents contain multiple relations, it was not possible to balance the relation distribution perfectly.

The RelEx component was trained using gold standard mentions and types, and was applied to the test and devtest documents also using gold mentions and types. The NLP Graphs were generated from the RelEx output as well as gold standard coreference information, the latter forming gold standard entities that were represented as NLP Graph nodes, with the selected relations representing links between them.

Relation	**Train**	**Dev**	**Test**	**Relation**	**Train**	**Dev**	**Test**
affectedBy	317	342	204	hasMember-HumanAgent	37	45	46
agentOf	812	847	596	hasMember-Person	29	35	19
attendedSchool	2	3	5	hasSibling	16	9	4
awardedBy	1	3	1	hasSpouse	8	7	15
awardedTo	1	3	1	hasSub-Organization	8	6	4
basedIn	122	93	51	instrumentOf	4	2	1

before	14	21	10	isLedBy	139	236	118
building-Physically-Destroyed	21	7	9	killingHuman-Agent	26	17	13
capitalOf	7	4	5	locatedAt	396	410	264
clientOf	23	14	14	mediating-Instrument-WeaponClass	15	12	4
colleague	29	23	16	near	31	22	8
diedAt	8	2	8	overlaps	4	0	3
diedOf	67	34	38	ownerOf	82	49	62
diedOn	14	8	11	participantIn	39	71	36
employedBy	201	216	170	partOf	188	236	132
eventDate	196	175	143	partOfMany	289	230	142
eventLocation	82	42	63	personKilled	2	7	1
founderOf	12	7	2	personGroup-Killed	0	1	0
hasAgeInYears	24	24	24	populationOf	4	1	2
hasBirthDate	1	2	0	quantityOf	204	176	114
hasBirthPlace	12	6	3	rateOf	1	2	2
hasChild	28	27	31	relative	21	13	9
hasCitizenship	9	10	7	residesIn	67	37	26
hasDisease	18	28	18	spokespersonFor	17	27	4
				timeOf	0	1	1

Table 1: Distribution of relations in train and test sets.

The NLP graphs can be generated by selecting relations from the NLP output in two dimensions:

- **Top vs. All:** Whether to take the top scoring relation between any mention pair (top) or all relations (all) between the pair.
- **Thresh:** A confidence threshold (strictly) above which relations are selected.

Clearly the same dimensions could have applied to the selection of type triples for the NLP graph, but the type information in these experiments is from the ground truth, so all type information from the NLP output was selected.

We refer to the graphs by document set (dev or test), $top/all@threshold$. Thus $devTop@.2$ is the NLP Graph from the development set using top relations above a .2 confidence, and $testAll@0$ is the NLP graph from the test set using all relations above 0 confidence.

For development and test sets, we generated 3 NLP graphs for use as primary extraction graphs, in all cases using top, and selecting relations at thresholds 0, .1, and .2. We also generated a single NLP graph for use as secondary for each set, using the $all@0$ setting.

Precision and Recall are determined by comparing system query results to the ground truth query results. The confidence of a query result set is the minimum confidence of the statements in the result, which performed better in experiments than using the product. Therefore for experiments with validation in a secondary graph the confidence thresholds below the primary graph threshold indicate the confidence of the secondary graph statements added to the result.

We generate a primary graph at different confidence levels when a secondary graph is being used for validation, because it is too complex to vary the thresholds for both graphs. For each primary graph threshold, we show the performance of different secondary graph thresholds, focussing at the space below the primary threshold (as at the primary graph threshold the secondary and the primary graph are relatively similar, i.e. it is rare to have more than one relation above threshold between two mentions).

We chose .2 as it was the threshold producing the max F1 score on the devset for RelEx when measured at the mention level (as opposed to the entity level in the NLP Graph), which was .28. We chose .1 as it was the threshold we had guessed at in the program evaluation, before having any data to back it up.

After performing many tests and experiments tuning the system on the development set, we ran a single experiment on the test set.

5.2 Results and Analysis

We show the results of the six experiments (Fig. 1 - 4), each one comparing a baseline, using the primary graph only, to using the primary graph with the secondary graph for hypothesis validation (Fig. 1, 2 and 3). The experiments are named after the primary graph with a "+s" indicating validation in a secondary graph.

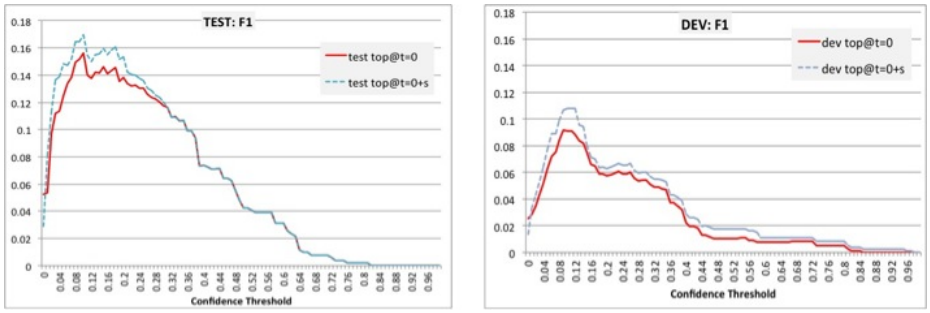

Fig. 1. F-Measure for the 0-threshold primary graph (with and without secondary graph for hypothesis validation) on test and dev sets

The Recall using the secondary graph for validation is always higher than without it, with a corresponding drop in Precision. F-measure improves across the confidence curve, with dramatic improvement in the interesting range below the primary graph threshold. Of particular interest is that the max F1 confidence point on the $top@.1 + s$

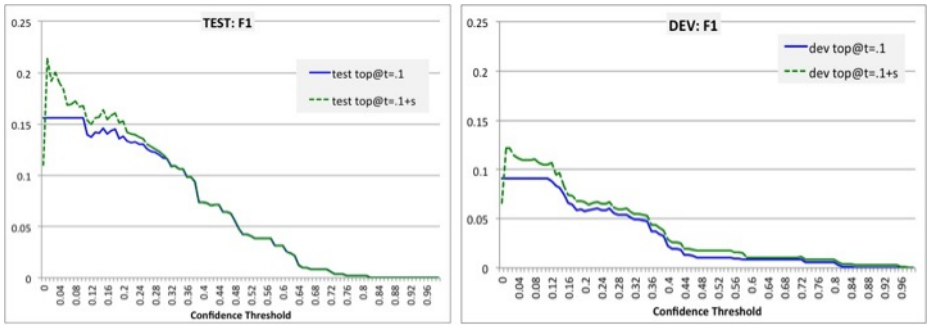

Fig. 2. F-Measure for the .1-threshold primary graph (with and without secondary graph for hypothesis validation) on test and dev sets

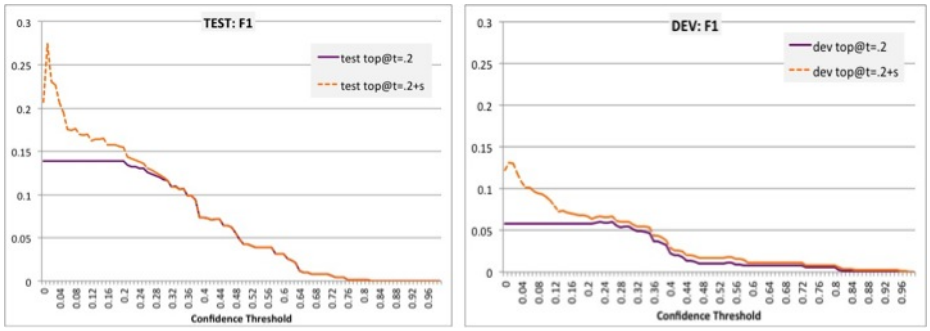

Fig. 3. F-Measure for the .2-threshold primary graph (with and without secondary graph for hypothesis validation) on test and dev sets

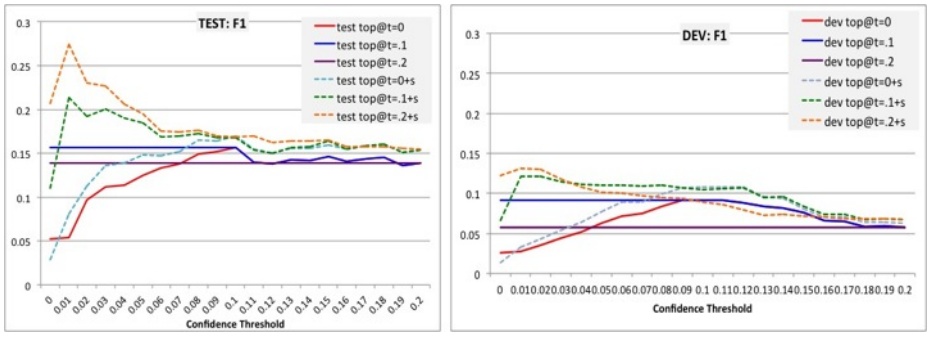

Fig. 4. A comparison of the six configurations on test and dev sets for the confidence range $0 - .2$

and $top@.2 + s$ experiments in both sets occur at a threshold of .01 (note again that below the primary threshold, the threshold value is only for statements added from the secondary graph) .

The threshold values themselves are artifacts of how the RelEx component computes its probabilities and was trained, but the result does strongly support our hypothesis that the probability of a relation holding between two mentions increases significantly if that relation would complete a conjunctive query result. The low max F1 threshold has certainly given us cause to more seriously consider the "cruft" produced at very low confidence levels that is normally discarded.

The results for $top@0$ in both sets also demonstrates that selecting a primary graph at some optimal threshold is better than a high Recall graph.

In the following table we summarize our results by choosing a configuration based on dev set performance and applying that configuration to the test set. The best performing configuration on the dev set was for $top@.2 + s$ with a threshold of 0.01 on the secondary graph hypotheses. We can see that the difference at the chosen threshold on the test set significantly outperforms the baseline on the same set.

Doc Set	**Configuration**	**P**	**R**	**F**
Dev	top@.2 (baseline)	0.10	0.04	0.06
Dev	top@.2+s@.01	0.16	0.11	0.13
Test	top@.2 (baseline)	0.28	0.09	0.14
Test	top@.2+s@.01	0.32	0.24	0.27

Table 2: Performance summary at chosen threshold

All the differences from baselines to results with secondary graph validation, in the figures and tables, are statistically significant to six decimal places (paired t-test with one-tailed distribution).

Overall performance (P, R, F) of the test set is also significantly higher than overall performance of the development set. Upon inspection, this is due to an uneven split of low-performing relations, in particular the *locatedAt, isLedBy* and *employedBy* relations, between the two sets. We did not look at per-relation performance in balancing the relations, only at the total counts, since absolute performance was not our focus.

6 Related Work

Modifying queries to control Precision and/or Recall is an old and well-explored idea. On the one hand, the area of *Query Expansion* [6] investigates adding terms to queries as either conjuncts (often to improve Precision by disambiguating original query terms) or disjuncts (to improve Recall by complementing original query terms with related terms). Terms to add may be found through corpus analysis (for example, finding terms that co-occur with query terms), through relevance feedback or through existing semantic resources, such as thesauri or ontologies. On the other hand, *Ontology-based Query Relaxation* [7] seeks to improve Recall not by deleting query terms, but by relaxing them through generalization in a taxonomy of relations and types. For example, an original query to find all people married to actors could be relaxed via *superrelations* (all

people related to actors) or via *supertypes* (all people married to entertainers) or both. In our system, the *Knowledge Base hypothesis checker* allows more specific relations and types to satisfy hypotheses based on taxonomic rules from an ontology. It could also easily reverse those rules to allow evidence of more general relations and types to support hypotheses. Alternatively, in *Query Approximation* [8], a complex, conjunctive query is evaluated by starting with a simpler query (typically a single conjunct) and successively adding query terms. At each step, the set of solutions includes the solutions to the original, complex query, along with (successively fewer) false positives. The goal is to produce these approximate solutions quickly, while converging on exact solutions given more time.

In *Question Answering*, document retrieval is used to produce a smaller candidate subset for subsequent processing. Recall is preferred over Precision in performance tradeoff for document retrieval, and incorrect answers are filtered by downstream modules [9]. To improve Recall of document retrieval in question answering, Bilotti et al. [9] use inflectional variants of the terms in the question, to expand the query for retrieving relevant documents. The final query is a conjunction of disjunct clauses, consisting of original query term and its inflectional variants. Query expansion with inflectional variants improves Recall of the document retrieval. Mollá et al. [10] and Mcnamee et al. [11] also highlight the importance of high Recall information extractors in question answering. They focus on a high-Recall Named Entity recognizer that allows allocation of multiple Named Entity tags to the same string, to handle ambiguous entities by not committing to a single tag; leaving it to the final answer extraction and scoring module to filter out incorrect answers.

7 Conclusions

We have presented a framework for hypothesis generation and validation in RDF NLP graphs. The hypotheses are generated given a query, and propose new statements to add to the graph that would increase the number of results for the query. We have shown that a useful way to validate such generated hypotheses is to look in the secondary interpretations of the text considered by NLP components and ultimately discarded.

In previous work [1] we demonstrated the value of hypothesis generation and validation as a way to circumscribe reasoning tasks when employing rules and background knowledge. In future work we plan to combine background knowledge and reasoning with secondary graphs for validation. Another promising direction is to consider making or breaking coreference decisions as hypotheses, and validating them with a secondary graph and other knowledge.

One way of viewing our result is to change the role of NLP and semantic technologies in end-to-end systems. A traditional view of NLP is as a layer below that of knowledge representation from which information flows strictly upwards. In this view, NLP components must therefore make their best guess, without any knowledge of the specific task at hand. In this paper, we have presented an approach that improves on this traditional view, by allowing the knowledge layer to “go back” to the NLP layer and ask for more targeted information about certain missing information that will help

it complete its task. Specifically, we are able to target further consideration of discarded interpretations when they will meet some user need (i.e. provide new results to queries).

8 Acknowledgments

The authors gratefully acknowledge the support of Defense Advanced Research Projects Agency (DARPA) Machine Reading Program under Air Force Research Laboratory (AFRL) prime contract no. FA8750-09-C-0172. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the DARPA, AFRL, or the US government. We would like to also thank Hideki Shima, Eric Riebling and Eric Nyberg from Carnegie Mellon University, as well as Sid Patwardhan and James Fan from IBM Research for their contribution to the implementation.

References

1. Barker, K., Fan, J., Welty, C.: Query driven hypothesis generation for answering queries over nlp graphs. In: IJCAI 2011 Workshop on Learning by Reading and its Applications in Intelligent Question-Answering. (2011)
2. Bunescu, R., Mooney, R.: Subsequence kernels for relation extraction. In Weiss, Y., Schölkopf, B., Platt, J., eds.: Advances in Neural Information Processing Systems 18. MIT Press, Cambridge, MA (2006) 171–178
3. Welty, C.A., Murdock, J.W.: Towards knowledge acquisition from information extraction. In Cruz, I.F., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L., eds.: International Semantic Web Conference. Volume 4273 of Lecture Notes in Computer Science., Springer (2006) 709–722
4. Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A.A., Lally, A., Murdock, J.W., Nyberg, E., Prager, J., Schlaefer, N., Welty, C.: Building watson: An overview of the deepqa project. AI Magazine **31** (2010) 59–79
5. Kalyanpur, A., Murdock, J.W., Fan, J., Welty, C.A.: Leveraging community-built knowledge for type coercion in question answering. In Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N.F., Blomqvist, E., eds.: International Semantic Web Conference (2). Volume 7032 of Lecture Notes in Computer Science., Springer (2011) 144–156
6. Efthimiadis, E.N.: Query expansion. Annual Review of Information Systems and Technology **31** (1996) 121–187
7. Hurtado, C.A., Poulovassilis, A., Wood, P.T.: Query relaxation in rdf. Journal on Data Semantics X, Lecture Notes In Computer Science **4900** (2008) 31–61
8. Stuckenschmidt, H., van Harmelen, F.: Approximating terminological queries. In Carbonell, J., Siekmann, J., Andreasen, T., Christiansen, H., Motro, A., Larsen, H., eds.: Flexible Query Answering Systems. Volume 2522 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2002) 329–343
9. Bilotti, M.W., Katz, B., Lin, J.: What works better for question answering: Stemming or morphological query expansion In: Proceedings of the Information Retrieval for Question Answering (IR4QA) Workshop at SIGIR 2004. (2004)
10. Mollá, D., van Zaanen, M., Cassidy, S.: Named entity recognition in question answering of speech data. In Colineau, N., Dras, M., eds.: Proc. ALTW 2007. Volume 5. (2007) 57–65
11. Mcnamee, P., Snow, R., Schone, P.: Learning named entity hyponyms for question answering. In: In Proceedings of the Third International Joint Conference on Natural Language Processing. (2008)