Syntactic Parsing of Web Queries

Xiangyan Sun
Fudan University

Haixun Wang
Facebook

Yanghua Xiao[*]
Fudan University

Zhongyuan Wang
Microsoft Research

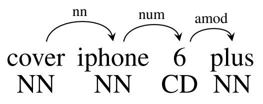
Abstract

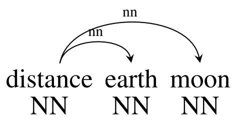
Syntactic parsing of web queries is important for query understanding. However, web queries usually do not observe the grammar of a written language, and no labeled syntactic trees for web queries are available. In this paper, we focus on a query's clicked sentence, i.e., a well-formed sentence that i) contains all the tokens of the query, and ii) appears in the query's top clicked web pages. We argue such sentences are semantically consistent with the query. We introduce algorithms to derive a query's syntactic structure from the dependency trees of its clicked sentences. This gives us a web query treebank without manual labeling. We then train a dependency parser on the treebank. Our model achieves much better UAS (0.86) and LAS (0.80) scores than state-of-the-art parsers on web queries.

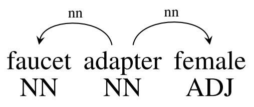
1 Introduction

Syntactic analysis is important in understanding a sentence's grammatical constituents, parts of speech, syntactic relations, and semantics. In this paper, we are concerned with the syntactic structure of a short text. The challenge is that short texts, for example, web queries, do not observe grammars of written languages (e.g., users often overlook capitalization, function words, and word order when creating a web query), and applying parsers trained on standard treebanks on queries leads to poor results.

Syntactic structures are valuable for query understanding. Consider the following web queries and their syntactic structures we would like to construct:







The syntactic structure of query `cover iphone 6 plus` tells us that the head token is `cover`, indicating its intent is to shop for the cover of an iphone, instead of iphones. With this knowledge, search engines show ads of iphone covers instead of iphones. For `distance earth moon`, the head is `distance`, indicating its intent is to find the distance between the earth and the moon. For `faucet adapter female`, the intent is to find a female faucet adapter. In summary, correctly identifying the head of a query helps identify its intent, and correctly identifying the modifiers helps rewrite the query (e.g., dropping non-essential modifiers).

Syntactic parsing of web queries is challenging for at least two reasons. First, grammatical signals from function words and word order are not available. Query `distance earth moon` is missing function words *between* (preposition), *and* (coordinator), and *the* (determiner) in conveying the intent

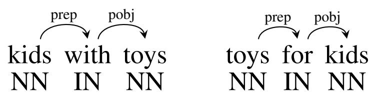
[*] Correspondence author. This paper was supported by National Key Basic Reserach Program of China under No.2015CB358800, by National NSFC(No.61472085, 61171132, 61033010, U1509213), by Shanghai Municipal Science and Technology Commission foundation key project under No.15JC1400900.

distance between the earth and the moon. Also, it is likely that queries {`distance earth moon`, `earth moon distance`, `earth distance moon`, $\cdots$} have the same intent, which means they should have the same syntactic structure. Second, there is no labeled dependency trees (treebank) for web queries, nor is there a standard for constructing such dependency trees. It will take a tremendous amount of time and effort to come up with such a standard and a treebank for web queries.

In this paper, we propose an end-to-end solution from treebank construction to syntactic parsing for web queries. Our model achieves a UAS of 0.830 and an LAS of 0.747 on web queries, which is dramatic improvement over state-of-the-art parsers trained from standard treebanks.

2 Our Approach

The biggest challenge of syntactic analysis of web queries is that they do not contain sufficient grammatical signals required for parsing. Indeed, web queries can be very ambiguious. For example, `kids toys` may mean either `toys for kids` or `kids with toys`, for which the dependency relationships between `toys` and `kids` are totally opposite.

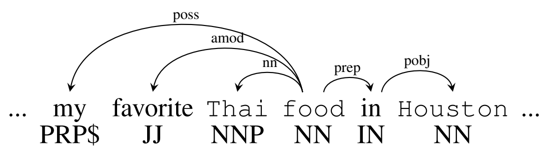


In view of this, why is syntactic parsing of web queries a legitimate problem? We have shown some example syntactic structures for 3 queries in Section 1. How do we know they are the correct syntactic structures for the queries? We answer these questions here.

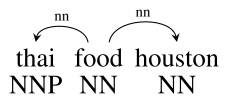
2.1 Derive syntax from semantics

In many cases, humans can easily determine the syntax of a web query because its intent is easy to understand. For example, for `toys kids`, we are pretty sure as a web query, its intent is to look for toys for kids, instead of the other way around. Thus, `toys` should be the head of the query, and `kids` should be its modifier. In other words, when the semantics of a query is understood, we can often recover its syntax.

We may then manually annotate web queries. Specifically, given a query, a human annotator forms a sentence that is consistent with the meaning he comes up for the query. Then, from the sentence's syntactic structure (which is well understood and can be derived by a parser), the annotator derives the syntactic structure of the query. For example, for query `thai food houston`, the annotator may formulate the following sentence:



Then we may project the dependency tree of the sentence to the query:



The above approach has two issues. First, `food` and `houston` are not directly connected in the dependency tree of the sentence. We connected them in the query, but in general, it is not trivial to infer synatx of the query from sentences in a consistent way. There is no linguistic standard for doing this. Second, annotation is very costly. A treebank project takes years to accomplish.

2.2 Semantics of a web query

To avoid human annotation, we derive syntactic understanding of the query from semantic understanding of the query. Our goal is to decide for any two tokens $x, y \in q$, whether there is a dependency arc between x and y, and if yes, what the dependency is.

Context-free signals. One approach to determine the dependency between x and y is to directly model $P(e|x, y)$, where e denotes the dependency ($x \rightarrow y$ or $x \leftarrow y$). It is context-free because we do not condition on the query where x and y appear in.

To acquire $P(e|x, y)$, we may consider annotated corpora such as Google's syntactic ngram (Goldberg and Orwant, 2013). For any x and y, we count the number of times that x is a dependent of y in the corpus. One disadvantage of this approach is that web queries and normal text differ significantly in distribution. Another approach (Wang et al., 2014) is to use search log to estimate $P(e|x, y)$, where x and y are nouns. Specifically, we find queries of pattern x PREP y, where PREP is a preposition {of, in, for, at, on, with, $\cdots$}. We have $P(x \rightarrow y|x, y) =$

$\frac{n_{x,y}}{n_{x,y}+n_{y,x}}$ where $n_{x,y}$ denotes the number of times pattern x PREP y appears in the search log. The disadvantage is that the simple pattern only gives dependency between two nouns.

Context-sensitive signals. The context-free approach has two major weaknesses: (1) It is risky to decide the dependency between two tokens without considering the context. (2) Context-free signals do not reveal the type of dependency, that is, it does not reveal the linguistic relationship between the head and the modifier.

To take context into consideration, which means estimating $P(e|x, y, q)$ for any two tokens $x, y \in q$, we are looking at the problem of building a parser for web queries. This requires a training dataset (a treebank). In this work, we propose to automatically create such a treebank. The feasibility is centered on the following assumption: The intent of q is contained in or consistent with the semantics of its **clicked sentences**. We call sentence s a clicked sentence of q if i) s appears in a top clicked page for q, and ii) s contains all tokens in q. For instance, assume sentence $s =$ "`... my favorite Thai food in Houston ...`" appears in one of the most frequently clicked pages for query $q =$ `thai food houston`, then s is a clicked sentence of q. It follows from the above assumption that the dependency between any two tokens in q are likely to be the same as the dependency between their corresponding tokens in s. This allows to create a treebank if we can project the dependency from sentences to queries. However, since x and y may not be directly connected by a dependency edge in s, we need a method to derive the dependency between $x, y \in q$ from the (indirect) dependency between $x, y \in s$. We propose such a method in Section 3.

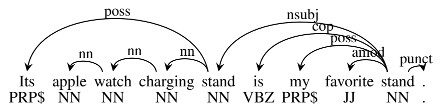
3 Treebank for Web Queries

We create a web query treebank by projecting dependency from clicked sentences to queries.

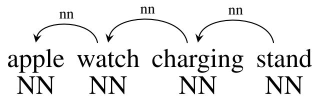
3.1 Inferring a dependency tree

A query q may have multiple clicked sentences. We describe here how we project dependency to q from such a sentence s. We describe how we aggregate dependencies from multiple sentences in Sec 3.2.

Under our assumption, each token $x \in q$ must appear in sentence s. But x may appear multiple times in s (especially when x is a function word). As an example, for query `apple watch stand`, we may get the following sentence:



Sentence s contains token `stand` twice, but only one subtree contains each token in q exactly once.



We use the following heuristics to derive a dependency tree for query q from sentence s.

1. Let T_s denote all the subtrees of the dependency tree of s.
2. Find the minimum subtree $t \in T_s$ such that each $x \in q$ has one and only one match $x' \in t$.
3. Derive dependency tree $t_{q,s}$ for q from t as follows. For any two tokens x and y in q:
   (a) if there is an edge from x' to y' in t, we create a same edge from x to y in $t_{q,s}$.
   (b) if there is a path[1] from x' to y' in t, we create an edge from x to y in $t_{q,s}$, and label it temporarily as *dep*.

We note the following. First, we argue that if the dependency tree of s has a subtree that contains each token in q once and only once, then it is very likely that the subtree expresses the same semantics as the query. On the other hand, if we cannot find such a subtree, it is an indication that we cannot derive reasonable dependency information from the sentence.

Second, it's possible x' and y' are not connected directly in s but through one or more other tokens. Thus, we do not know the label of the derived edge. We will decide on the label in Sec 3.3.

Third, we want to know whether it is meaningful to connect x and y in q while x' and y' are not directly connected in s. We evaluated a few hundreds

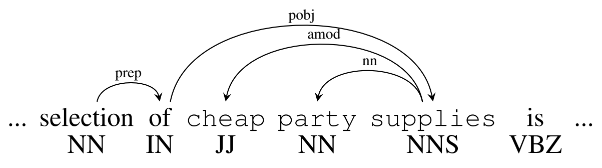
[1]A path consists of edges of the same direction.

of query-sentence pairs. Among the cases where dependency trees for queries can be derived successfully, we found that x' and y' are connected in 5 possible ways (Table 1). We describe them in details next.

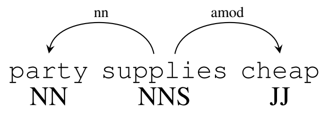
directly connected	46%
connected via function words	24%
connected via modifiers	24%
connected via a head noun	4%
connected via a verb	2%

Table 1: Dependency Projection

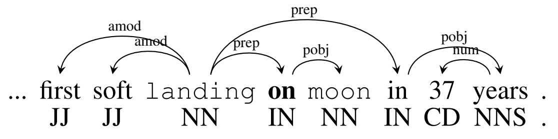
Directly connected. In this case, we copy the edge and its label directly. Consider query `party supplies cheap`'s clicked sentence below:



Here both (`party`, `supplies`) and (`supplies`, `cheap`) are directly connected. The query inherits the dependencies, but note that tokens `supplies` and `cheap` have different word orders in *q* and *s*:

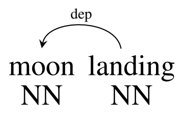


Connected via function words. It is quite common prepositions are omitted in a query. Consider query `moon landing`'s clicked sentence:

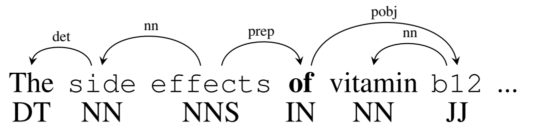


prep
amod
amod
prep
pobj
pobj
num
... first soft landing on moon in 37 years .
JJ JJ NN IN NN IN CD NNS .

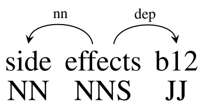
We can derive the following dependency tree:



For query `side effects b12`, suppose we have the following sentence:



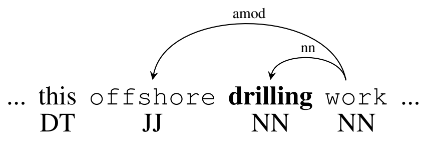
The derived dependency tree should be:



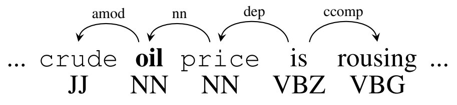
For these two cases, we need to introduce a derived edge for the query, which will be resolved later to a specific dependency label.

Connected via modifiers. Many web queries are noun compounds. Their clicked sentences may have more modifiers. Depending on the bracketing, we may or may not have direct dependencies.

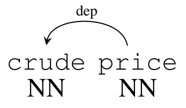
For `offshore work` and its clicked sentence below, missing `drilling` in the query does not cause any problem: `offshore` and `work` are still directly connected in the dependency tree.



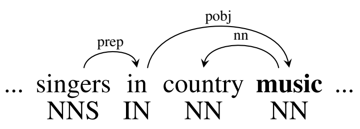
But not for `crude price` and its clicked sentence. Still, there is a path: `crude` ← `oil` ← `price`.



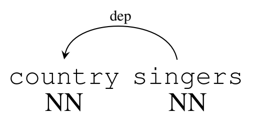
In this case, we create a dependency between `crude` and `oil` in the query and give it a temporary label *dep*. We will resolve it to a specific label later.



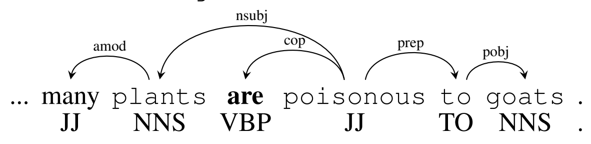
Connected via a head noun. In some cases, the head of a noun compound is missing. Consider `country singers` and its clicked sentence:



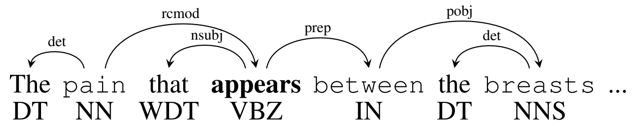
Clearly they mean the same thing, but the head (`music`) of the noun compound is missing in the query. Still, a path exists from `singers` to `country`, and we create a dependency:



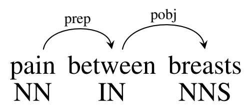
Connected via a verb. One common case is the omission of copular verbs. Consider `plants poisonous to goats` and its clicked sentence:



Here, the missing `are` does not cause any problem. But for query `pain between breasts` and its clicked sentence:



we need to introduce a derived edge, and it leads to:



3.2 Inferring a unique dependency tree

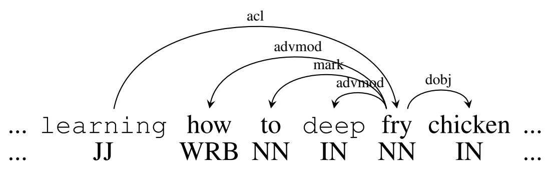
A query corresponds to multiple clicked sentences. From each sentence, we derive a dependency tree. These dependency trees may not be the same, because i) dependency parsing for sentences is not perfect; ii) queries are ambiguous; or iii) some queries do not have well-formed clicked sentences.

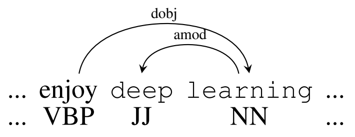
To choose a unique dependency tree for a query q, we define a scoring function f to measure the quality of a dependency tree t_q derived from q's clicked sentence s:

$$f(t_q, s) = \sum_{(x \to y) \in t_q} -\alpha dist(x, y) + \log \frac{count(x \to y)}{count(x \leftarrow y)} \quad (1)$$

where $(x \to y)$ is an edge in the tree t_q, $count(x \to y)$ is the occurrence count of the edge $x \to y$ in the entire query dataset, $dist(x, y)$ is the distance of words x and y on the original sentence parsing tree, and α is a parameter to adjust the importance between the two measures (its value is empirically determined).

The first term of the scoring function measures the *compactness* of the query tree. Consider two clicked sentences for query `deep learning`:


acl
advmod
mark
advmod
dobj
... learning how to deep fry chicken ...
... JJ WRB NN IN NN IN ...



In the first sentence, `deep` and `learning` are indirectly connected through `fry` so the total distance measure is 2. In the second query, the distance is 1. Therefore, query aligned with the second sentence is better than the first sentence.

The second term of the scoring function measures the *global consistency* among head modifier directions. For a word pair (x, y), if in the dataset, the number of edges $x \to y$ dominates the number of edges $x \leftarrow y$, then the latter is likely to be incorrect.

One important thing to note is word order. Word order may influence the head-modifier relations between two words. For example, `child of` and `of child` should definitely have different head-modifier relations. Therefore, we treat two words of different order as two different word pairs.

Correct	Wrong	Query	Sentence
side ← effects	side → effects	1110:1	11257:17
benefits → of	benefits ← of	144:63	5228:0
Full ← Movie	Full → Movie	128:5	1585:27
coconut ← oil	coconut → oil	91:10	1507:46
credit ← card	credit → card	96:2	4394:60

Table 2: Examples of globally inconsistent head modifier relations

Table 2 shows some examples of conflicting dependency edges and their corresponding occurrence count in queries and sentences.

3.3 Label refinement

In Section 3.1, some dependencies are derived with a placeholder label *dep*. Before we use the data to train a parser, we must resolve *dep* to a true label, otherwise they introduce inconsistency in the training data. For example, consider a simple query `crude price`. From clicked sentences that contain `crude oil price`, we derive `crude`$\overset{\text{dep}}{\longleftarrow}$`price`, but from those that contain `crude price`, we derive `crude`$\overset{\text{amod}}{\longleftarrow}$`price`.

To resolve *dep*, we resort to majority vote first. For any $x \overset{\text{dep}}{\longleftarrow} y$, we count the occurrence of $x \overset{\text{label}}{\longleftarrow} y$ in the training data for each concrete label. If the frequency of a certain label is dominating by a predetermined threshold (10 times more frequent than any other label), then we resolve *dep* to that label.

With our training data, the above process is able to resolve about 90% dependencies. We can simply discard queries that contain unresolvable dependencies. However, such queries still contain useful information, for example, the direction of this edge, and the directions and labels of all the other edges. We develop a bootstrapping method to preserve such useful information. First, we train a parser on data without *dep* labels. This skips about 10% queries in our experiments. Second, we use the parser to predict the unknown label. If the prediction is consistent with the annotation except for the *dep* label, we use the predicted label. Third, we add the resolved queries into the training data and train a final parser. Experiments show the bootstrapping approach improves the quality of the parser.

4 Dependency Parsing

We train a parser from the web query treebank data. We also try to incorporate context-free head-modifier signals into parsing. To make it easier to incorporate such signals, we adopt a neural network approach to train our POS tagger and parser.

4.1 Neural network POS tagger and parser

We first train a neural network POS tagger for web queries. For each word in the sentence, we construct features out of a fixed context window centered at that word. The features include the word itself, case (whether the first letter, any letter, or every letter in the word, is in uppercase), prefix, and suffix (we recognize a pre-defined set of prefixes and suffixes, for the rest we use a special token "UNK"). For the word feature, we use pre-trained word2vec embeddings. For word case and prefix/suffix, we use random initialization for the embeddings. The accuracy of the trained POS tagger is similar to that of (Ganchev et al., 2012), which outperforms POS taggers trained on PTB data.

Buffer features
$b_1.wt$, $b_2.wt$, $b_3.wt$
Stack features
$s_1.wt$, $s_2.wt$, $s_3.wt$
Tree features
$lc_1(s_1).wtl$, $lc_2(s_1).wtl$, $rc_1(s_1).wtl$, $rc_2(s_1).wtl$
$lc_1(lc_1(s_1)).wtl$, $rc_1(rc_1(s_1)).wtl$
$lc_1(s_2).wtl$, $lc_2(s_2).wtl$, $rc_1(s_2).wtl$, $rc_2(s_2).wtl$
$lc_1(lc_1(s_2)).wtl$, $rc_1(rc_1(s_2)).wtl$

Table 3: The feature templates. $s_i(i = 1, 2, ...)$ denote the i^{th} top element of the stack, $b_i(i = 1, 2, ...)$ denote the i^{th} element on the buffer, $lc_k(s_i)$ and $rc_k(s_i)$ denote the kth leftmost and rightmost children of s_i, w denotes words, t denotes POS tag, l denotes label.

We use the arc standard transition based dependency parsing system (Nivre, 2004). The architecture of the neural network dependency parser is similar to that of (Chen and Manning, 2014) designed for parsing sentences. The features used in parsing are shown in Table 3.

4.2 Context free features

In Section 2.2, we discussed context-free signals $P(e|x, y)$ and context-sensitive signals $P(e|x, y, q)$. Previous work (Wang et al., 2014) uses context-free signals for syntactic analysis of a query. Our approach outperforms the context-free approach.

An interesting question is, will context-free signals further improve our approach? The rationale is that although context-sensitive signals $P(e|x, y, q)$ are more accurate in predicting the dependency between x and y, such signals are also very sparse. Do context-free signals $P(e|x, y)$ provide backoff information in parsing?

It is not straightforward to include $P(e|x, y)$ in the neural network model. The head-modifier relations $P(e|x, y)$ may exist between any pair of tokens in the input query. Essentially, it is a pairwise graphical model and it is difficult to directly incorporate the signals in transition based dependency parsing.

We treat context-free signals as *prior knowledge*. We train head-modifier embeddings for each token, and use such embeddings as pre-trained embeddings. Specifically, we use an approach similar to training word2vec embeddings but focusing on head modifier relationships instead of co-occurrence relationships. More specifically, we train an one hidden layer neural network classifier to determine whether two words have head-modifier relations. The input of the neural network is the concatenation of the embeddings of two words. The output is whether the two words form a proper head-modifier relationship. We obtain a large set of head-modifier data from text corpus by mining "h PREP m" pattern in search log where h and m are nouns. Then, for each known head modifier pair h and m, we use (h, m) as positive example and (m, h) as negative example. For each word, we also choose a few random words as negative examples. During the training process, the gradients are back propagated to the word embeddings. After training, the embeddings should contain sufficient information to recover head modifier relations between any word pairs.

But we did not observe improvement over the existing neural network that are trained on context sensitive treebank data alone. The head-modifier embeddings has about 3% advantage in UAS over randomized embeddings. However, using pretrained word2vec embeddings, we also achieve 3% advantage. Thus, it seems that context-sensitive signals plus the generalizing power of embeddings contain all the context-free signals already.

5 Experiments

In this section, we start with some case studies. Then we describe data and compare models.

In experiments, we use the standard UAS (unlabeled attachment score) and LAS (labeled attachment score) score for measuring the quality of dependency parsing. They are calculated as:

$$UAS = \frac{\#\ correct\ arc\ directions}{\#\ total\ arcs} \tag{2}$$

$$LAS = \frac{\#\ correct\ arc\ directions\ and\ labels}{\#\ total\ arcs} \tag{3}$$

5.1 Case Study

We compare dependency trees produced by our QueryParser and Stanford Parser (Chen and Manning, 2014) for some web queries (Stanford Parser is trained from the standard PTB treebank). Table 4 shows that Stanford Parser heavily relies on grammar signals such as function words and word order, while QueryParser relies more on the semantics of the query. For instance, in the 1st example, QueryParser identifies `toys` as the head, regardless of the word order, while Stanford parser always assumes the last token as the head. In the 2nd example, the semantics of the query is a school (`vanguard school`) at a certain location (`lake wales`). QueryParser captures the semantics and correctly identifies `school` as the head (root) of the query, while Stanford parser treats the entire query as a single noun compound (likely inferred from the POS tags).

5.2 Clicked Sentences

For training data, we use one-month Bing query log (between July 25, 2015 and August 24, 2015). From the log, we obtain web query q and its top clicked URLs $\{url_1, url_2, ..., url_m\}$. From the urls, we retrieve the clicked HTML document, and find sentences $\{s_1, s_2, ..., s_n\}$ that contain all words (regardless to their order of occurrence) in q. Then we extract query-sentence tuples $(q, s, count)$ to serve as our training data to generate a web query treebank. The size (# of distinct query-sentence pairs) of the raw clicked sentences is 390,225,806.

5.3 Web Query Treebank

We evaluate the 3 steps of treebank generation. After each step, we sample 100 queries from the result and manually compute their UAS and LAS scores. We also count the number of total query instances in each step. The results are shown in Table 5.

- **Inferring a dependency tree**: For each (query, sentence) pair, we project dependency from the sentence to the query. The number of instances shown in Table 5 are the input number of (query, sentence) pairs. It shows that we obtain dependency trees for only 31% of the queries, while the rest do not satisfy our filtering criterion. This however is not a concern. By sacrificing recall in this process, we ensure high precision. Given that query log is large, precision is more important.

- **Inferring a unique dependency tree**: In this step, we group (query, sentence) pairs by unique queries. Using the method in Section

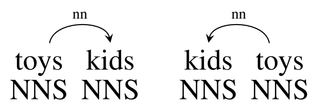
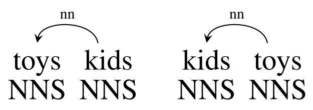
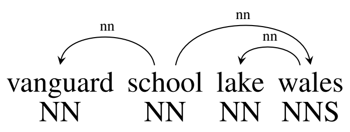
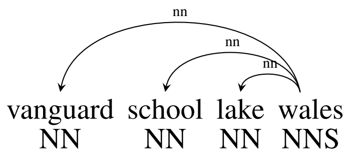
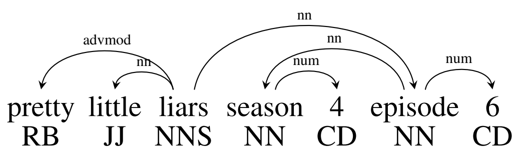
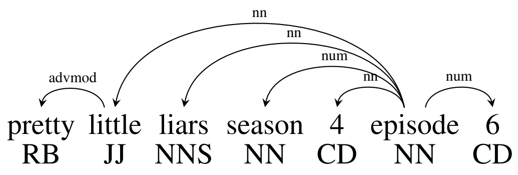
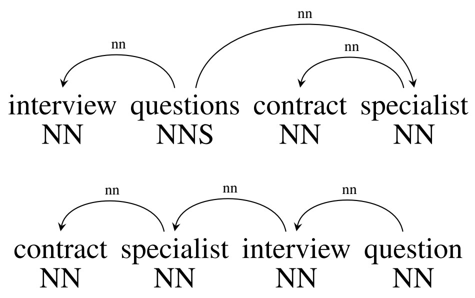
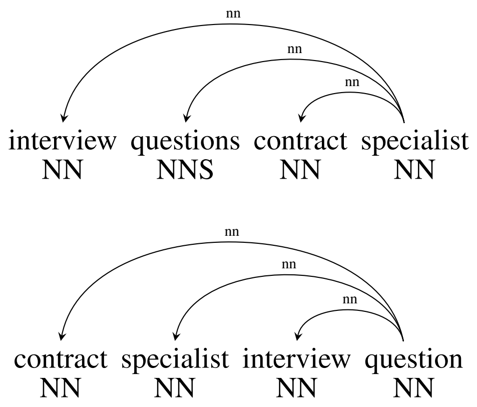
QueryParser	Stanford parser
	
	
 nn advmod nn nn num num pretty little liars season 4 episode 6 RB JJ NNS NN CD NN CD	 nn nn num advmod nn num pretty little liars season 4 episode 6 RB JJ NNS NN CD NN CD
 nn nn nn interview questions contract specialist NN NNS NN NN nn nn nn contract specialist interview question NN NN NN NN	 nn nn nn interview questions contract specialist NN NNS NN NN nn nn nn contract specialist interview question NN NN NN NN

Table 4: Case study of parsers.

3.2, each group produces one or zero dependency trees. The number of instances in Table 5 corresponds to the number of different query groups. The overall success rate is high. This is expected as the filtering process uses majority voting, and we already have high precision parsing trees after the first step.

- **Label refinement**: Dependency labels are refined using the methodology in Section 3.3. It shows that with majority voting and bootstraping, we are able to keep all the input.

5.4 Parser Performance

We compare QueryParser against three state-of-the-art parsers: Stanford parser, which is a transition based dependency parser based on neural network, MSTParser (McDonald et al., 2005), which is a graph based dependency parser based on minimum spanning tree algorithms, and LSTMParser (Dyer et al., 2015), which is a transition based dependency parser based on stack long short-term memory cells. Here, QueryParser is trained from our web query treebank, while Stanford Parser and MSTParser are trained from standard PTB treebanks.

For comparison, we manually labeled 1,000 web queries to serve as a ground truth dataset[2]. We produce POS tags for the queries using our neural network POS tagger. To specifically measure the ability of QueryParser in parsing queries with no explicit syntax structure, we split the entire dataset **All** into two parts: **NoFunc** and **Func**, which correspond to queries without any function word, and queries with at least one function word. The number of queries

[2]https://github.com/wishstudio/queryparser

Step	Total Instances	Produced Instances	Success Rate	UAS	LAS
Inferring a dependency tree	3986300	1229860	31%	0.906	0.851
Inferring a unique tree	716261	680857	95%	0.910	0.851
Label refinement	680857	680857	100%	0.917	0.855

Table 5: Training dataset generation statistics

System	**All** (n**=1000)**		**NoFunc** (n**=900)**		**Func** (n**=100)**	
	UAS	LAS	UAS	LAS	UAS	LAS
Stanford	0.694	0.602	0.670	0.568	0.834	0.799
MSTParser	0.699	0.616	0.683	0.691	0.799	0.766
LSTMParser	0.700	0.608	0.679	0.578	0.827	0.790
QueryParser + label refinement	0.829	0.769	0.824	0.761	0.858	0.818
QueryParser + word2vec	0.843	0.788	0.843	0.784	0.838	0.812
QueryParser + label refinement + word2vec	**0.862**	**0.804**	**0.858**	**0.795**	**0.883**	**0.854**

Table 6: Parsing performance on web queries

of the two datasets are 900 and 100, respectively.

Table 6 shows the results. We use 3 versions of QueryParser. The first two use random word embedding for initialization, and the first one does not use label refinement. From the results, it can be concluded that QueryParser consistently outperformed competitors on query parsing task. Pre-trained word2vec embeddings improve performance by 3-5 percent, and the postprocess of label refinement also improves the performance by 1-2 percent.

Table 6 also shows that conventional depencency parsers trained on sentence dataset relies much more on the syntactic signals in the input. While Stanford parser and MSTParser have similar performance to our parser on **Func** dataset, the performance drops significantly on **All** and **NoFunc** dataset, when the majority of input has no function words.

6 Related Work

Some recent work (Ganchev et al., 2012; Barr et al., 2008) investigated the problem of syntactic analysis for web queries. However, current study is mostly at postag rather than dependency tree level. Barr et al. (2008) showed that applying taggers trained on traditional corpora on web queries leads to poor results. Ganchev et al. (2012) propose a simple, efficient procedure in which part-of-speech tags are transferred from retrieval-result snippets to queries at training time. But they do not reveal syntactic structures of web queries.

More work has focused on resolving simple relations or structures in queries or short texts, particularly entity-concept relations (Shen et al., 2006; Wang et al., 2015; Hua et al., 2015), entity-attribute relations (Pasca and Van Durme, 2007; Lee et al., 2013), head-modifier relations (Bendersky et al., 2010; Wang et al., 2014). Such relations are important but not enough. The general dependency relations we focus on is an important addition to query understanding.

On the other hand, there is extensive work on syntactic analysis of well-formed sentences (De Marneffe et al., 2006). Recently, a lot of work (Collobert et al., 2011; Vinyals et al., 2015; Chen and Manning, 2014; Dyer et al., 2015) started using neural network for this purpose. In this work, we use similar neural network architecture for web queries.

7 Conclusion

Syntactic analysis of web queries is extremely important as it reveals actional signals to many downstream applications, including search ranking, ads matching, etc. In this work, we first acquire well-formed sentences that contain the semantics of the query, and then infer the syntax of the query from the sentences. This essentially creates a treebank for web queries. We then train a neural network dependency parser from the treebank. Our experiments show that we achieve significant improvement over traditional parsers on web queries.

References

Cory Barr, Rosie Jones, and Moira Regelson. 2008. The linguistic structure of english web-search queries. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 1021–1030, Stroudsburg, PA, USA. Association for Computational Linguistics.

Michael Bendersky, Donald Metzler, and W Bruce Croft. 2010. Learning concept importance using a weighted dependence model. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 31–40. ACM.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependeny parsing with stack long short-term memory. In *Proc. ACL*.

Kuzman Ganchev, Keith Hall, Ryan McDonald, and Slav Petrov. 2012. Using search-logs to improve query tagging. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, ACL '12, pages 238–242, Stroudsburg, PA, USA. Association for Computational Linguistics.

Yoav Goldberg and Jon Orwant. 2013. A dataset of syntactic-ngrams over time from a very large corpus of english books. In *Second Joint Conference on Lexical and Computational Semantics (* SEM)*, volume 1, pages 241–247.

Wen Hua, Zhongyuan Wang, Haixun Wang, Kai Zheng, and Xiaofang Zhou. 2015. Short text understanding through lexical-semantic analysis. In *International Conference on Data Engineering (ICDE)*.

Taesung Lee, Zhongyuan Wang, Haixun Wang, and Seung-won Hwang. 2013. Attribute extraction and scoring: A probabilistic approach. In *International Conference on Data Engineering (ICDE)*.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics.

Marius Pasca and Benjamin Van Durme. 2007. What you seek is what you get: Extraction of class attributes from query logs. In *IJCAI*, volume 7, pages 2832–2837.

Dou Shen, Jian-Tao Sun, Qiang Yang, and Zheng Chen. 2006. Building bridges for web query classification. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 131–138. ACM.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2755–2763.

Zhongyuan Wang, Haixun Wang, and Zhirui Hu. 2014. Head, modifier, and constraint detection in short texts. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 280–291. IEEE.

Zhongyuan Wang, Kejun Zhao, Haixun Wang, Xiaofeng Meng, and Ji-Rong Wen. 2015. Query understanding through knowledge-based conceptualization. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*.