Translating Questions to SQL Queries with Generative Parsers Discriminatively Reranked

Alessandra Giordani and Alessandro Moschitti
Computer Science and Engineering Department
University of Trento, Povo (TN), Italy
{giordani,moschitti}@disi.unitn.it

ABSTRACT
In this paper, we define models for automatically translating a factoid question in natural language to an SQL query that retrieves the correct answer from a target relational database (DB). We exploit the DB structure to generate a set of candidate SQL queries, which we rerank with an SVM-ranker based on tree kernels. In particular, in the generation phase, we use (i) lexical dependencies in the question and (ii) the DB metadata, to build a set of plausible SELECT, WHERE and FROM clauses enriched with meaningful joins. We combine the clauses by means of rules and a heuristic weighting scheme, which allows for generating a ranked list of candidate SQL queries. This approach can be recursively applied to deal with complex questions, requiring nested SELECT instructions. Finally, we apply the reranker to reorder the list of question and SQL candidate pairs, whose members are represented as syntactic trees. The F1 of our model derived on standard benchmarks, 87% on the first question, is in line with the best models using external and expensive hand-crafted resources such as the question meaning interpretation. Moreover, our system shows a Recall of the correct answer of about 94% and 98% on the first 2 and 5 candidates, respectively. This is an interesting outcome considering that we only need pairs of questions and answers concerning a target DB (no SQL query is needed) to train our model.

KEYWORDS: Natural Language Interface to Databases, Semantic Parsing, Reranking.

Proceedings of COLING 2012: Posters, pages 401–410,
COLING 2012, Mumbai, December 2012.

TABLES

TABLE_SCHEMA	TABLE_NAME	...
geoquery	state	
geoquery	city	
geoquery	river	
geoquery	border	
geoquery	highlow	
...	...	

COLUMNS

TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	DATA_TYPE	...
geoquery	state	state_name	varchar	
geoquery	state	population	float	
geoquery	city	city_name	varchar	
geoquery	city	state_name	varchar	
geoquery	river	traverse	varchar	
...			...	

KEY_COLUMN_USAGE

TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	REFERENCED_ TABLE_SCHEMA	REFERENCED_ TABLE_NAME	REFERENCED_ COLUMN_NAME	...
geoquery	city	state_name	geoquery	state	state_name	
geoquery	river	traverse	geoquery	state	state_name	
...						

Figure 1: A DBMS catalog containing GeoQuery database

1 Introduction

In the last decade, a variety of approaches have been developed to automatically convert natural language questions into machine-readable instructions. A considerable amount of research work has tackled such problem along the line of semantic parsing, e.g., (Ge and Mooney, 2005; Wong and Mooney, 2006) defined algorithms for mapping natural language questions to logical forms, (Minock et al., 2008) made use of a specific semantic grammar and (Zettlemoyer and Collins, 2005; Wong and Mooney, 2007) applied lambda calculus to the meaning representation of the questions.

In the perspective of question answering (QA) targeting the information of databases (DBs), the automatic system only needs to execute one or more Structured Query Language (SQL) queries that retrieve the answer to the posed natural language question. In our recent work (Giordani and Moschitti, 2009a,b, 2010, 2012), we have shown that machine learning algorithms, exploiting syntactic representations of both questions and queries, can be used to automatically associate a question with the corresponding SQL queries. One limitation of this approach is that the set of possible queries that a user would execute on the DB must be known in advance. This because such method cannot generate new queries, it can only verify if a given query probably retrieves the correct answer for the asked question. This limitation is critical as the design of a generative parser which, given a question, feeds the model above with a reasonable set of candidate queries seems inevitably to fall in the category of semantic parsers.

This paper will demonstrate that it is possible to avoid full-semantic interpretation by relying on (i) a simple SQL generator, which both exploits syntactic lexical dependencies in the questions along with the target DB metadata; and (ii) advanced machine learning such as kernel-based rerankers, which can improve the initial candidate list provided by the generative parser.

The idea of point (i) can be understood by noting that database designers tend to choose names for entities, relationships, tables and columns according to the semantics of the application domain. Such logic organization is referred to as *catalog*, and in SQL systems it is stored in a database called Information_Schema (IS for brevity). The values stored in IS along with their constraints and data types are important metadata, which is useful to decode a natural language question about the DB domain in a corresponding SQL query. For example, given the IS associated with a DB, shown in Figure 1 and 2, if we ask a related question, q_0:*Which rivers run through New York?*[1], a human being will immediately derive the semantic predicate

[1]GeoQueries (Tang and Mooney, 2001) is available at http://www.cs.utexas.edu/m̃l/geo.html

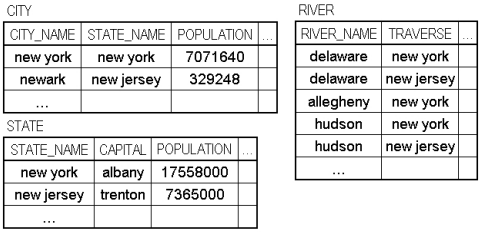

Figure 2: GEOQUERY database fragment

run_through(*rivers*, *New York*) from it. Then she/he will associate the argument *river*, which is also the question focus, with the table `RIVER`. Once the latter is targeted, she/he will select the column `TRAVERSE`, which, being a synonym of *run through*, provides the same predicative relation asked in the question. Finally, by instantiating the available argument, *New York*, in such predicate, she/he will retrieve the set {`Delaware`, `Allegheny`, `Hudson`} from the column `RIVER_NAME`, i.e., the missing argument (as they are in the same row of *New York*).

The above example shows that several inference steps must be performed to retrieve the correct answer. In particular, lexical relations must be extracted from the questions, e.g., using dependency syntactic parsing and predicate arguments must be expanded with their synonyms or related concepts, e.g., using Wordnet (Miller, 1995).

Additionally, ambiguity and noise play a critical role in deriving the interpretation of the question described above but we can exploit metadata to verify that the selected sense is correct, e.g., from the fact that *New York* in this database is in the column `TRAVERSE`, we can gather evidence that the sense of *running-through* matches the one of *traverse*. Therefore, the general idea is to generate all possible (even ambiguous) queries exploiting related metadata information (i.e., primary and foreign keys, constraints, datatypes, etc.) to select the most probable one using a ranking approach.

Last but no least, we deal with nested SQL queries and complex questions containing subordinates, conjunctions and negations. We designed a generative algorithm based on the matches between lexical dependencies and SQL structure, which allows for building a set of feasible queries. Starting from the general syntactic formulation of an SQL query, i.e.,:

$$\text{SELECT } column \text{ FROM } table \text{ [WHERE } condition\text{]}, \tag{1}$$

we generate the set of column, table and condition terms using the lexical relations in the questions. The relation arguments can be generalized using Wordnet and disambiguated using metadata and the execution of the resulting query candidates in the reference DB. Once the list of candidates is available, we can apply supervised rerankers to improve the accuracy of the system. For this step, we improved on the model, we proposed in (Giordani and Moschitti, 2009b), by designing a preference reranker based on structural kernels. The input of such model consists of pairs of syntactic trees of the questions and queries, where for the query we use their derivation tree provided by the SQL compiler.

We tested our model on three subsets of GeoQuery data (Tang and Mooney, 2001) such that we could compare with several systems of previous work. The results show that our generative model is robust and accurate achieving a Recall of 95.0% on the first 5 candidate answers. Additionally, when we apply our structural reranker to the generated list, we obtain state-of-the-art results, i.e., an F1 of 87.2 on the top answer and a Recall of about 98% on the top 5

answers. The major contribution of our approach is that it is simple and does not require any heavy annotation or handcrafted semantic resources. It just relies on the database and the availability of a training set of correct question and answer pairs targeting a DB.

2 Syntactic Dependencies and Relational Algebra

We extract lexical relations from the question using the Stanford Dependencies Parser (de Marneffe et al., 2006). This provides a set of binary grammar relations existing between a *gov*ernor and a *dep*endent, where each dependency has the format *abb_rel_name* (*gov*, *dep*), while *gov* and *dep* are words in the sentence associated with a number indicating the position of the word in the sentence. In particular, we consider collapsed representations, i.e., the dependencies, involving prepositions, conjuncts, as well as information about the referent of relative clauses, are collapsed to get direct dependencies between content words. For example, the Stanford Dependencies Collapsed (SDC) representation for the question,
q_1: "*What are the capitals of the states that border the most populated state?*"
is the following:

$$SDC_{q_1} = \Big\{ attr(\text{are-2, what-1}), root(\text{ROOT-0, are-2}), det(\text{capitals-4, the-3}), nsubj(\text{are-2, capitals-4}), \\ nsubj(\text{border-9, states-7}), rcmod(\text{states-7, border-9}), det(\text{states-13, the-10}), \\ advmod(\text{populated-12, most-11}), amod(\text{state-13, populated-12}), dobj(\text{borders-9, state-13}) \Big\}$$

A general SQL query structure is shown in Eq. 1. Its execution starts from the relation in the FROM clause by selecting DB tuples that satisfy the condition indicated in the WHERE clause (optional) and then projects the target attribute specified in the SELECT clause. In relational algebra, selection and projection are performed by σ and π operators respectively. The meaning of the SQL query above is the same as that of the relational expression:

$$\pi_{COLUMN}\left(\sigma_{CONDITION}(TABLE)\right) \quad (2)$$

It is worth noting that while relational algebra formally applies to sets of tuples (i.e. relations), in a DBMS relations are bags so it may contain duplicate tuples (Garcia-Molina et al., 2008). For our purposes the fact of having duplicates in the result adds noise[2]. In our QA task we expect that questions can be answered with a single result set (e.g., we can deal with "*Cities in Texas*" and "*Populations in Texas*" but not with the combined query "*Cities and their population in Texas*"). That is, even if in general *COLUMN* could be a - possibly empty - list of attributes, in our system it just contains one attribute. We can apply aggregation operators to this attribute that summarize it by means of SUM, AVG, MIN, MAX and COUNT, always combined with DISTINCT keyword, e.g. `SELECT COUNT(DISTINCT state.state_name)`.

Instead, *CONDITION* is a logical expression where basic conditions e_L OP e_R, with OP={<, >, LIKE, IN}, are combined with AND, OR, NOT operators. While e_L is always in the form `table.column`, e_R could be:

- numerical value (e.g. `city.population > 15000`) or
- string value (e.g. `city.state_name LIKE "Texas"`) or
- nested query (e.g. `city.city_name IN (SELECT state.capital FROM state)`

An example of a complex WHERE condition could be the following, selecting "*major non-capital cities excluding texas*"): `city.population > 15000 AND city.city_name NOT IN (SELECT state.capital FROM state)) AND NOT city.state_name LIKE "Texas"`.

The meaning of *TABLE* is more straightforward, since it should contain the table name(s) to which the other two clauses refer. This clause could just be a single relation or a JOIN operation, which selectively pairs tuples of two relations. In practice we take the Cartesian product of two relations and exclusively select those tuples that satisfy a condition C. We use the SQL keyword ON to keep this condition C separated from the other WHERE conditions since it reflects a database requirement and should not match any term of the question (e.g., `city JOIN state ON city.city_name = state.capital`). The

[2]We always delete multiple copies of a tuple by using the keyword DISTINCT in the *COLUMN* field

(1)$root$(ROOT, are),	$\Pi = \{$capital, state$\}$
(2)$nsubj$(are, capital),	$\Sigma = \{$are$\} \Rightarrow \Sigma = \phi$
(3)$prep\_of$(capital, state),	
(4)$nsubj$(border, state),	$\Pi' = \{$state, border$\}$
(5)$rcmod$(state, border),	$\Sigma' = \{$border, state$\}$
(6)$advmod$(populat, most),	
(7)$amod$(state, populat),	$\Pi'' = \{$most, populat, state$\}$
(8)$dobj$(border, state)	$\Sigma'' = \phi$

Figure 3: Categorizing stems into projection and/or selection oriented sets

$$\mathcal{S} = \{state.capital^3, state.state\_name^2, border\_info.state\_name^1, ...\}$$
$$\mathcal{S}' = \{border\_info.state\_name^3, border\_info.border^2, state.state\_name^2, ...\}$$
$$\mathcal{S}'' = \{max(state.population)^4, max(city.population)^3, state.population^3, ...\}$$

Figure 4: A subset of SELECT clauses for q_1

complexity of generated queries is fairly high indeed, since we can deal with questions that require nesting, aggregation and negation in addition to basic projection, selection and joining (e.g. "*How many states have major non-capital cities excluding Texas*").

3 Automatic Generation of SQL Queries from a Question

The basic idea of our generative parser is to produce queries of the form:

$$\exists s \in \mathcal{S}, \exists f \in \mathcal{F}, \exists w \in \mathcal{W} \text{ s.t. } \pi_s\left(\sigma_w(f)\right) \text{answers q}, \quad (3)$$

where q is the starting question represented by means of SDC_q and $\mathcal{S}, \mathcal{F}, \mathcal{W}$ are the three sets of clauses (argument of SELECT, FROM and WHERE, respectively). The query answering a question, $\pi_s\left(\sigma_w(f)\right)$, can be chosen among the set of all possible queries $\mathcal{A}$ ={SELECT $s\times$ FROM $f\times$ WHERE w} in a way that maximizes the probability of generating a result set answering q.

Clause set construction. To build $\mathcal{S}$ and $\mathcal{W}$ sets, we identify the stems that can most probably participate in the role of projection (i.e., composing the SELECT argument) and/or selection (composing the WHERE condition). Accordingly, we create two sets of terms Π and Σ. The main idea is that some terms can be used to choose the DB table and column where the answer is contained whereas others tend to indicate properties (i.e., table rows) useful to locate the answer in the column. For categorizing terms we use a heuristic based on the grammatical relations provided by a dependency parser[3]. For example, let us consider the list of preprocessed dependencies SDC_{q_1} in Fig. 3. At the first iteration, we use *ROOT* to add *are* to Σ. Then, the *nsubj(are,capital)* suggests that the subject *capital* may be a focus of a projection thus we included in Π. Additionally, given *prep_of(capital, state)*, *state* is a modifier of the subject thus it may have the same role and we include it in Π. We immediately verify this assumption by automatically checking that there is an occurrence of `state.capital` in IS.

We use the set Π to retrieve all the metadata terms that match with its elements: this will produce S according to our generative grammar[4]. For example, considering the IS scheme in Figure 1, the SELECT clauses that are generated from Π, whose elements are listed in the right side of Fig. 3, are shown in Fig. 4 [5]. For generating the WHERE clauses, we need to divide Σ in two distinct sets: Σ_L and Σ_R, for the left-and right-hand side of the condition, respectively. The set Σ_L contains stems matching the IS metadata terms. Σ_L is used to generate the left condition $\mathcal{W}_L$.

Query Generation. Since each clause set may contain up to ten items, the cartesian product between clause set can be very large. Thus, we verify some conditions during the generative process, e.g., tables

[3]For lack of space we cannot report such heuristics in this paper.
[4]Again for lack of space we cannot report it here.
[5]The superscript numbers just indicate the weight associated with each statement.

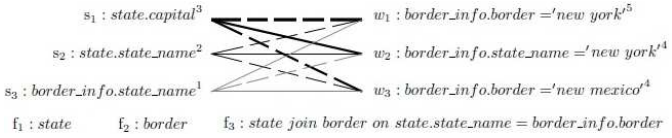

Figure 5: Possible pairing between clauses for q_2

appearing in SELECT and WHERE clauses must also appear in the FROM clause to avoid the failure of the execution of the query. As an example, Figure 5 shows generated clauses from the question q_2, together with possible combinations; the tuple $\langle s_1, f_1, w_1 \rangle$ is not correct as it leads to the MySQL error: `Unknown table: border_info`.

4 The Experiments

We ran several experiments to evaluate the accuracy of our approach for automatic generation and selection of correct SQL queries from questions. We experimented with a well-known dataset GeoQuery developed in the context of semantic parsing.

To generate the set of possible SQL queries we applied our algorithm described in Section 3 to the GeoQueries corpus. We considered the full GeoQuery annotation (Geo880) but we used the subset of 700 pairs (henceforth Geo700) since they are translated by (Popescu et al., 2003) from Prolog data to SQL queries. Additionally, to compare with latest systems (Clarke et al., 2010; Liang et al., 2011), which used a subset of 500 pairs, hereafter Geo500, we annotated the remaining 180 pairs as they were included in Geo500. The latter was randomly split by (Clarke et al., 2010) in 250 pairs for training and 250 pairs for testing. The data is slightly *easier* since the number of logical symbols per word are limited to an average of 13 logical symbols. It is worth noting that even if we manually annotated missing questions with their answering SQL queries, we only used them for extracting the answer from the database and evaluate the pair correctness (so we do not really use the SQL queries).

To learn the reranker, we used SVM-Light-TK[6], which extends the SVM-Light optimizer (Joachims, 1999) with tree kernels (Moschitti, 2006) as described in (Giordani and Moschitti, 2009b). We modeled many different combinations described in the next section. We used the default parameters, i.e. the cost and trade-off parameters $= 1$ (for normalized kernels) and $\lambda = 0.4$.

4.1 Generative Results

We carried out the first experiment on Geo700. Our algorithm could generate a correct SQL query in the first 25 candidates for 95.3% of the cases but could not answer to 33 questions. This was due to (i) empty clauses set $\mathscr{S}$ and/or $\mathscr{W}$, for example, "*How many square kilometers in the US?*" does not contain useful stems; and (ii) mismatching in nested queries, for example, "*Count the states which have elevations lower than what Alabama has*" contains an implicit reference to the missing information. In addition, there were incomplete questions like "*Which states does the Colorado?*" from which we retrieved an incomplete dependency set. When our algorithm can generate an ordered list of possible queries, the top query is correct for 82% of the cases. Additionally, the correct answer is contained in the first 10 candidates for 99% of the cases (excluding the 33 questions above). Note also that the correct query is found among the first three in 93% of the cases. This shows that our ranking based on heuristic weights is rather robust and produce high recall. The accuracy on the top candidate can then be promisingly increased with reranking.

We obtain similar results with the Geo500 subset: we fail to generate an answer in 18 out of the 250 pairs of the test set. We also found that the correct answer is 78% of the times in the top position while it can be retrieved among the first top seven in 98% of the cases.

[6] `http://disi.unitn.it/~moschitt/Tree-Kernel.htm`

Combination	Rec@1	Rec@2	Rec@3	Rec@4	Rec@5
NO RERANKING	81.4±5.8	87.6±3.8	90.8±3.1	94.0±2.4	95.0±2.0
STK + STK	83.5±3.6	90.4±3.5	94.2±2.9	95.8±2.0	96.7±1.7
STK × STK	86.5±4.0	92.6±3.7	95.3±3.2	97.0±1.8	97.7±1.4
BOW × STK	86.7±4.1	92.1±3.2	95.6±2.5	97.1±1.4	97.6±1.2
$(1+\text{STK} \times \text{STK})^2$	**87.2±3.9**	**94.1±3.4**	95.6±2.7	97.1±1.9	97.9±1.4

Table 1: Kernel combination recall (± Std. Dev) for GEO700 dataset

4.2 Reranking Results

To improve the accuracy of our generative model, we used a preference reranking approach (Moschitti et al., 2006, 2012; Severyn and Moschitti, 2012). The reranker uses the following kernels: $\text{STK}_n+\text{STK}_s$, $\text{STK}_n \times \text{STK}_s$, $\text{BOW}_n \times \text{STK}_s$ and $(1+\text{STK}_n \times \text{STK}_s)^2$, where n indicates kernels for questions and s for queries, respectively. We applied standard 10-fold cross validation and measured the average Recall in selecting a correct query for each question. The results for different models on GEO700 are reported in Table 1. The first column lists the kernel combination by means of product and sum between pairs of basic kernels used for the question and the query, respectively. The other columns show the Recall of at least 1 correct answer in the top k positions (more precisely the average of Recall@k over 10 folds ± Std. Dev). Additionally, we evaluated the same kernels for reranking pairs generated from the GEO500 dataset. Using $\text{STK}_n+\text{STK}_s$ we obtain a Recall of 84.77%, while if we exploit the product $\text{STK}_n \times \text{STK}_s$, we achieve 87.31%. These results are rather exciting since they compare favorably with the state-of-the-art.

5 Related Work and Discussion

Early work on semantic parsing (Tang and Mooney, 2001) required either the definition of rules and constrains in an ILP framework or manually produced meaning representations (Ge and Mooney, 2005; Wong and Mooney, 2006), which are costly to produce. Additionally, authoring systems where developed by specifying a semantic grammar (Minock et al., 2008), which requires large effort of human experts.

Table 2 shows the f-measure of some state-of-the-art systems to which we compare. Such systems were tested on GEOQUERIES, according to different experimental setups and data versions. The first half of the table reports on systems exploiting the annotated logical form (deriving the answer) whereas the last five rows show the f-measure of systems only exploiting the training pairs, questions and answers.

PRECISE (Popescu et al., 2003) is the only system evaluated on GEO700 in terms of correct SQL queries. The value reported in the table refers to the correctness of answering questions if the expected SQL query (i.e., one with equivalent result) is produced by one of the top k queries[7]. Also our system can provide multiple answers and if we select the first k candidates, we highly increase the Recall (within the first 2 we have an F1 of 90%). Note that (Popescu et al., 2003) needed to rephrase some queries to achieve their result. Another system similar to ours, which applies SVMs and string kernel is KRISP (Kate and Mooney, 2006). The major difference is that it requires meaning representations (following a user-defined MR grammar), instead of SQL queries.

Recent work has also explored learning to map sentences to meaning representations suitable for applying lambda-calculus (Zettlemoyer and Collins, 2005; Wong and Mooney, 2007). This kind of system require a large amount of supervision. In particular, the system in (Zettlemoyer and Collins, 2005) shows a Precision of 96.3% and a Recall of 79.3%, for an f-measure of 86.9%, while our system shows a Precision of 82.8% and a Recall of 87.2%, for an f-measure of 85.0%. Thus, our system trades-off 2 points of accuracy for avoiding large work for handcrafting resources, i.e., the semantic trees manually annotated for each question. Moreover, our system is much simpler to implement. A more recent work (Lu et al., 2008) does not rely on annotation and shows a Precision of 89.3% and a Recall of 81.5%, for an f-measure of 85.2%. Their generative model coupled with a discriminative reranking technique (MODELIII+R) is conceptually similar to our approach.

SEMRESP (Clarke et al., 2010) also learn a semantic parser from question-answer pairs. They achieve the

[7] Being k a small constant, not better defined by the authors.

System Name	Human Supervision	GEO500	GEO700	GEO880
PRECISE	Rephrase question	-	87%	-
KRISP	Specify the grammar	-	-	81%
MODELIII+R	-	-	-	85%
SEMRESP	Define a Lexicon	80%	-	-
UBL	Specify a CCG Lexicon	-	-	89%
SEMRESP	Define a Lexicon	73%	-	-
UBL	Specify a CCG Lexicon	-	-	85%
DCS	Define Lexical Triggers	79%	-	89%
DCS$^+$	Define an Augmented Lexicon	87%	-	91%
OUR SYSTEM/SQL*	-	**87%**	85%	-

Table 2: Comparison between state-of-the-art systems in terms of F1.

highest accuracy when tested on annotated logical forms whereas when tested on answers their accuracy is lower (80% vs. 73% in f-measure). In contrast, our system, evaluated on answers, outperforms their best system in all setting, e.g., (85% vs. 80%).

Another system evaluated both with logical forms and with answers is UBL (Kwiatkowski et al., 2010). Starting from a restricted set of lexical items and CCG combinatory rules, it is able to learn new lexical entries and achieves the best performance with GEO880 when trained with logical forms.

In contrast, the best performing system that does not exploit the annotation of the GEO880 is DCS (Liang et al., 2011). The comparison of the systems above with ours on GEO500 shows that ours largely outperforms DCS (87% vs. 78% in f-measure). Our system performs comparably to the version enriched with prototype triggers, DCS$^+$, even though we do not exploit such manual resources.

In summary, our system is competitive with other supervised parsers as it: (i) only relies on the answers, i.e., without using any annotated meaning representations (e.g. Prolog data, MR, Lambda calculus, SQL queries); and (ii) requires much less supervision since there is no need to build semantic representation. Our manual intervention only regards the definition of few synonym relations, i.e., *border* and *next to* as synonyms for *traverse*, since there are not such relations in Wordnet. The rest of the lexicon is induced by the database metadata or obtained exploiting Wordnet.

Finally, our system is competitive with the state-of-the-art defined in (Lu et al., 2008). This is not surprising since we use a very similar approach, i.e., a generative model coupled with discriminative reranking. However, while the system above learns a parser on meaning representations, we only need natural language questions their answers (of course targeting a DB).

6 Conclusion

In this paper, we have approached question answering targeting database information by automatically generating SQL queries in response of the posed question. Our method exploits grammatical dependencies and metadata matching. To our knowledge, our approach to build and combine clauses sets is novel. Additionally, we firstly experimented with a preference reranking kernel, which is able to boost the accuracy of our generative model.

Given the high accuracy, the simplicity and the practical usefulness of our approach, (e.g., we can generate the correct question in the first 5 candidates in 98% of the cases), we believe that it can be successfully used for real-world applications.

Acknowledgments

The research described in this paper has been partially supported by the European Community's Seventh Framework Programme (FP7/2007-2013) under the grants #247758: ETERNALS – Trustworthy Eternal Systems via Evolving Software, Data and Knowledge, and #288024: LIMOSINE – Linguistically Motivated Semantic aggregation engiNes

References

Clarke, J., Goldwasser, D., Chang, M., and Roth, D. (2010). Driving semantic parsing from the world's response. In *CoNLL*.

de Marneffe, M.-C., MacCartney, B., and Manning, C. D. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings LREC 2006*.

Garcia-Molina, H., Ullman, J. D., and Widom, J. (2008). *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition.

Ge, R. and Mooney, R. (2005). A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 9–16, Ann Arbor, Michigan. Association for Computational Linguistics.

Giordani, A. and Moschitti, A. (2009a). Semantic mapping between natural language questions and SQL queries via syntactic pairing. In *NLDB*, pages 207–221.

Giordani, A. and Moschitti, A. (2009b). Syntactic structural kernels for natural language interfaces to databases. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part I*, ECML PKDD '09, pages 391–406, Berlin, Heidelberg. Springer-Verlag.

Giordani, A. and Moschitti, A. (2010). Corpora for automatically learning to map natural language questions into SQL queries. In *Proceedings of LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

Giordani, A. and Moschitti, A. (2012). Generating SQL queries using natural language syntactic dependencies and metadata. In *NLDB*, pages 164–170.

Joachims, T. (1999). Making large-scale SVM learning practical. In Schölkopf, B., Burges, C., and Smola, A., editors, *Advances in Kernel Methods*.

Kate, R. J. and Mooney, R. J. (2006). Using string-kernels for learning semantic parsers. In *Proceedings of the 21st ICCL and 44th Annual Meeting of the ACL*, pages 913–920, Sydney, Australia. Association for Computational Linguistics.

Kwiatkowski, T., Zettlemoyer, L. S., Goldwater, S., and Steedman, M. (2010). Inducing probabilistic ccg grammars from logical form with higher-order unification. In *EMNLP*, pages 1223–1233. ACL.

Liang, P., Jordan, M. I., and Klein, D. (2011). Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*, pages 590–599.

Lu, W., Ng, H. T., Lee, W. S., and Zettlemoyer, L. S. (2008). A generative model for parsing natural language to meaning representations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 783–792, Stroudsburg, PA, USA. Association for Computational Linguistics.

Miller, G. A. (1995). WordNet: A lexical database for English. *Communications of the ACM*.

Minock, M., Olofsson, P., and Näslund, A. (2008). Towards building robust natural language interfaces to databases. In *NLDB '08: Proceedings of the 13th international conference on Natural Language and Information Systems*, Berlin, Heidelberg.

Moschitti, A. (2006). Efficient convolution kernels for dependency and constituent syntactic trees. In *Proceedings of ECML'06*.

Moschitti, A., Ju, Q., and Johansson, R. (2012). Modeling topic dependencies in hierarchical text categorization. In *ACL (1)*, pages 759–767.

Moschitti, A., Pighin, D., and Basili, R. (2006). Semantic role labeling via tree kernel joint inference. In *Proceedings of CoNLL-X*, New York City.

Popescu, A.-M., A Etzioni, O., and A Kautz, H. (2003). Towards a theory of natural language interfaces to databases. In *Proceedings of the 2003 International Conference on Intelligent User Interfaces*, pages 149–157, Miami. Association for Computational Linguistics.

Severyn, A. and Moschitti, A. (2012). Structural relationships for large-scale learning of answer re-ranking. In *SIGIR*, pages 741–750.

Tang, L. R. and Mooney, R. J. (2001). Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the 12th European Conference on Machine Learning*, pages 466–477, Freiburg, Germany.

Wong, Y. W. and Mooney, R. (2006). Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 439–446, New York City, USA. Association for Computational Linguistics.

Wong, Y. W. and Mooney, R. (2007). Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967, Prague, Czech Republic. Association for Computational Linguistics.

Zettlemoyer, L. S. and Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*, pages 658–666.