NALMO: A Natural Language Interface for Moving Objects Databases

Xieyang Wang
Nanjing University of Aeronautics and Astronautics
Nanjing, China
xieyang@nuaa.edu.cn

Jianqiu Xu
Nanjing University of Aeronautics and Astronautics
Nanjing, China
jianqiu@nuaa.edu.cn

Hua Lu
Roskilde University
Roskilde, Denmark
luhua@ruc.dk

ABSTRACT

Moving objects databases (MODs) have been extensively studied due to their wide variety of applications including traffic management, tourist service and mobile commerce. However, queries in natural languages are still not supported in MODs. Since most users are not familiar with structured query languages, it is essentially important to bridge the gap between natural languages and the underlying MODs system commands. Motivated by this, we design a natural language interface for moving objects, named NALMO. In general, we use semantic parsing in combination with a location knowledge base and domain-specific rules to interpret natural language queries. We design a corpus of moving objects queries for model training, which is later used to determine the query type. Extracted entities from parsing are mapped through deterministic rules to perform query composition. NALMO is able to well translate moving objects queries into structured (executable) languages. We support four kinds of queries including time interval queries, range queries, nearest neighbor queries and trajectory similarity queries. We develop the system in a prototype system SECONDO and evaluate our approach using 240 natural language queries extracted from popular conference and journal papers in the domain of moving objects. Experimental results show that (i) *NALMO achieves accuracy and precision 98.1% and 88.1%, respectively*, and (ii) *the average time cost of translating a query is 1.47s.*

CCS CONCEPTS

• **Information systems** → **Spatial-temporal systems**; **Query languages for non-relational engines**.

KEYWORDS

moving objects database, natural language interface, semantic parsing, structured language, query processing

ACM Reference Format:
Xieyang Wang, Jianqiu Xu, and Hua Lu. 2021. NALMO: A Natural Language Interface for Moving Objects Databases. In *17th International Symposium on Spatial and Temporal Databases (SSTD '21), August 23–25, 2021, virtual, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3469830.3470894

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SSTD '21, August 23–25, 2021, virtual, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8425-4/21/08…$15.00
https://doi.org/10.1145/3469830.3470894

1 INTRODUCTION

Mobile computing devices such as PDAs and mobile phones have been widely used in real life. As a result, a large amount of mobile data is collected and stored in the system. Such data represents objects continuously changing locations over time. The system managing such data is called moving objects databases, MOD for short [11]. In the domain of moving objects databases, a number of queries and analysis tasks have been extensively studied including range queries [4], nearest neighbor queries [10] and trajectory similarity queries. Several prototype systems are developed such as HERMES [23], SECONDO [9], PLACE∗ [35] and TMOM [15].

Despite extensive research having been studied in this area, there is lack of a query interface for non-expert users sending their requests in natural languages. It is a non-trivial task for most users to write structured languages such as SQL. Developing a natural language interface for databases (NLIDBs) has become an urgent task as databases support an increasing number of applications. NLIDBs will significantly benefit users as they can use natural languages to request a variety of queries instead of mastering the structured language as well as the underlying structure such as the database schema. Figure 1 depicts the task that we would like to achieve to empower MODs. That is, a natural language query is precisely and correctly translated into a structured and executable expression in MODs.

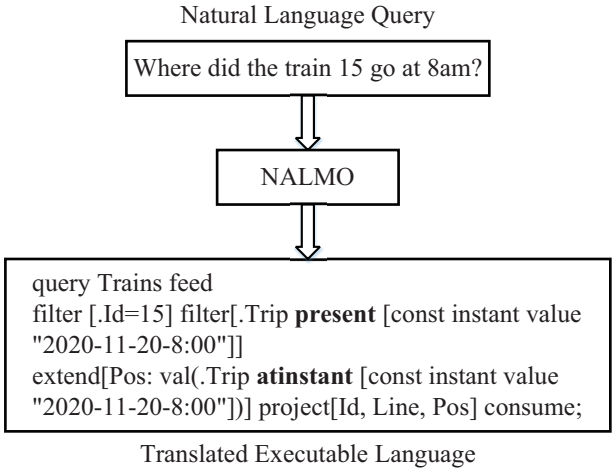

Figure 1: Example of translating

NLIDB is a challenging task mainly due to two difficulties: (i) *semantic understanding and analysis*, and (ii) *query translation*. The

questions asked by users are usually ambiguous or in-complete in terms of semantic. In the domain of moving objects databases a variety of queries have different key entities including time, location and moving objects identification. Constructing an effective semantic parser for identifying and extracting entities is a crucial task. Although MODs are already equipped with a rich set of operators and access methods, effectively and accurately combining entities and operators into structured languages is a non-trivial task due to the query diversity.

NLIDB systems have achieved satisfactory results in specific domains. Rule-based and machine learning-based methods are usually used to perform the translation from natural language to SQL. Some NLIDB systems use machine learning [30] to understand and translate natural language queries such as *NL$_2$CM* [1] and *SpatialNLI* [18]. Although there are training datasets and corpora for relational databases, there is lack of datasets and corpora in many specific domains. Corpora that can be used for query classification of MODs are rare or even not available. *PRECISE* [26], *NaLIR* [17] and *ATHENA* [27] use rule-based semantic analysis and query construction methods. These methods limit expressive power and can only be applied to specific domains. Natural language queries often contain ambiguities and grammatical errors which make mistakes and produce erroneous output. However, a purely rule-based method will greatly restrict the expression of queries that users can put forward. Some NLIDBs ask users for intervention to perform query reformulation because natural language is inherently ambiguous. Sometimes when the system gives users an answer, users usually do not know whether the answer is correct. Verifying the correctness of the answer is hard for users. Helping NLIDB systems understand query semantics is challenging and all users can do is to modify the query and re-query. The state-of-the-art systems based on methods above have been able to translate natural language queries relatively well in different domains. However, there is no NLIDB that supports MODs. Besides, most of the existing benchmarks are used for testing natural language queries of relational databases. There is no dataset for moving objects queries, and non-moving object queries cannot be used as test cases to determine the effect of the translation queries to MODs. Most current traditional database management systems including relational databases cannot handle continuously changing data, such as the location information of moving objects. This is because the data remains unchanged in a traditional database unless the data is modified. However, if we want to represent a moving object and answer a query about its location, the location of the moving object needs to be updated continuously. For relational databases, frequent updates of location information are impractical. Compared with relational databases, MODs provide users with many basic operators (e.g., *filter*(•)) and special operators (e.g., *deftime*(•)) for different queries. Without these operators, natural language queries for moving objects are difficult to be effectively translated into corresponding relational database queries.

In this paper, we design and develop a natural language interface for MODs including (i) *semantic parsing and analysis*, (ii) *structured language composition*, and (iii) *selection of suitable datasets*. At the moment, the system supports four kinds of queries including time interval queries, range queries, nearest neighbor queries and trajectory similarity queries, but not limited to the four types. This is because the corresponding operators and functions are implemented. If the operators and functions of other queries and analysis are well handled in the system, those queries and analysis can be well supported. To support natural language queries, the first issue is semantic understanding. We preprocess user-specified spatio-temporal queries in which there may exist punctuation that can affect the processing result or be unnecessary. These punctuations should be replaced with spaces, before subsequent work can be carried out. After preprocessing, word segmentation and entity recognition are performed by appropriate natural language processing tools. The extraction of key semantic information is one of the most important parts of natural language translation. We propose a location knowledge base and a domain methodology for entity analysis such that a word like *"Nanjing"* can be parsed as a location.

We collect more than 60 conference and journal papers to extract query examples, and manually generate moving objects queries for knowledge enhancement to build a moving objects query corpus. The model trained through the corpus can accurately determine the query type. After the type determination, query interpretations are mapped into query fragments based on inference rules, before finally being assembled into complete queries. We implement our proposal in the system SECONDO that provides users with many basic operators and special operators. According to different query types, the parsed entities are mapped to corresponding query fragments and these fragments are combined with operators to obtain the final structured query. Efficiency is also a significant part of the user experience. In order to improve the translation efficiency of the system, we use a location-based prefix index in the process of extracting location entities. It is worth noting that most of the query objects in our examples are public transportation, because one of our test datasets is moving underground trains. However, we can still query moving objects excluding public transportation as long as there are corresponding datasets.

The main contributions of this paper are summarized as follows.

- We design an algorithm for spatio-temporal data parsing in which a location knowledge base is utilized to extract location information. A location-based prefix index is developed to improve the extraction efficiency.
- We propose a corpus for query type determination and develop methods for constructing structured query languages which combine the parsed key entities and system operators.
- We develop NALMO in an extensible database system SECONDO and conduct experimental evaluation using two datasets. The experiment results demonstrate that our method achieves accuracy and precision of 98.1% and 88.1%, respectively. The average time cost of translating a query is 1.47s.

The rest of the paper is organized as follows. We review the related work in Section 2 and provide the framework of NALMO in Section 3. We introduce natural language understanding and the query translation in Sections 4 and 5, respectively. The experimental evaluation is presented in Section 6, followed by the conclusion in Section 7.

Table 1: Query examples and types

Id	Natural Language Query Example	Type
Q1	Where were the trains at a certain time t_1?	Time Interval Query
Q2	Where did the trains go during the time period [t_2, t_3]?	Time Interval Query
Q3	When did the train 123 pass underground Nanjing Station?	Range Query
Q4	What trains pass the station Nantong during the time interval [t_4, t_5]?	Range Query
Q5	Show me five nearest neighbors to the train 3 from 8am to 10am.	Nearest Neighbor Query
Q6	Find k nearest neighbors to the train O_1 at any time instance of the time period [t_6, t_7].	Nearest Neighbor Query
Q7	Find trains which are similar to the train 7 at any time instance of the time period [t_8, t_9].	Trajectory Similarity Query

2 RELATED WORK

MODs have been extensively studied in a range of applications including transportation systems, express delivery services and mobile commerce. In the literature, a number of techniques are proposed such as data models [8] [13] [36], index structures [33] [31] [24] [3] and query algorithms [4] [10]. However, there is still lack of a natural language interface in moving objects databases which supports transforming natural language expressions into executable languages.

To have good NLIDBs, we need to understand natural language problems well. A number of systems use rule-based methods to understand natural language in a specific domain. These systems are usually restricted in natural language support such as grammar and vocabulary to improve the accuracy of semantic understanding [37]. *PRECISE* [26] defines the notion of semantic tractability and identifies a subset of natural language queries that can be precisely translated into SQL. *NaLIX* [20] limits natural language queries to a controlled subset based on a predefined grammar. An ideal NLIDB enables users to make arbitrarily complex temporary queries on the underlying database and obtain accurate information with the least effort. Therefore, in the process of semantic understanding, *NALIX* and *DiaSQL* [7] modify the query in subsequent user interactions to reformulate the parse tree, but the reformulation usually needs excessive user interactions. *NaLIR* [17] detects the parse tree so that users can directly modify the parse tree instead of the questions raised by the user. *ATHENA* [27] constructs a translation index through specific entity extraction, and uses ontology-based domain knowledge to bridge the semantic gaps caused by leaky abstraction [19]. *ATHENA++* [28] further translates complex nested SQL queries. *NL2TRANQUYL* [2] stores and models relevant information by employing ontology which contains different concepts. *QUERIX* [38] analyzes the syntax of natural language by a syntactic analyzer, which requires that the natural language components must be complete. Otherwise, the results obtained by the syntax analyzer are not accurate enough, which may decrease the accuracy of final results. Neural networks [12] are increasingly used to train models of parsing natural language sentences, encoding and decoding for natural language semantic understanding.

On the basis of a positive understanding of natural language, how to translate natural language is also of crucial importance. The common method of query construction is to use a unique algorithm to construct a structured query from the meaning of natural language query and the domain knowledge corresponding to the underlying database [19]. *ATHENA* proposes a unique two-stage approach that first translates an input natural language query into an ontology query language (OQL) query defined over the ontology, and then translates the OQL query into its corresponding SQL query. *NLPQC* [29] accepts queries based on specific domain templates. *QUERIX* extracts specific components from the syntax tree to match knowledge with the knowledge base to obtain the final result. State-of-the-art machine learning technologies [16] [32] are gradually applied to natural language translation. *NL_2CM* [1] mines the crowd for individual by translating questions of the audience into OASSIS-QL which is defined as an extension of SPARQL. SpatialNLI [18] uses seq2seq translation to enhance its translation model. A number of systems combine deterministic algorithm approaches and machine learning methods to construct structured languages [21]. A parser generates corresponding SQL clauses, then further constructs SQL query candidates according to the corresponding templates and rules, and finally ranks the candidates of the constructed languages through the SVM ranker [6].

NLIDB for spatio-temporal databases has also received attention in recent years [5]. *TEQUILA* [14] proposes a knowledge graph to deal with temporal questions, where cues for temporal relations need to be discovered and handled. *Wang et al.* propose a spatial understanding model *SpatialNLI* based on contextual semantics [18], which can recognize the meaning of spatial entities. Spatial terms are extracted from natural language descriptions through symbolic expressions to represent spatial features and relations between them [25]. *NL2TRANQUYL* [2] is a system primarily for trip-planning requests in a large multimodal transportation system with different constraints such as taking the least time, travelling the shortest distance and choosing the least expensive route.

Some NLIDBs [2] [17] use different parsers to obtain dependency parser tree for a given query, and the quality of parser tree depends largely on the parser. For MODs, the location and time information is very critical, so we do not rely much on natural language processing tools and will further process natural language queries to extract key entities. The majority of the existing NLIDBs, such as *NL_2CM* [1], *NaLIR* [17] and *ATHENA++* [28] construct queries via query composition. These NLIDBs all map the processed key information to the corresponding structured components, but for various queries, different DBMS and structured languages provide diverse operators and clauses. So the required mapping values are different. NALMO is designed for the moving objects query. NALMO considers multiple query types in moving objects and designs different mapping templates for each query type, while others mainly

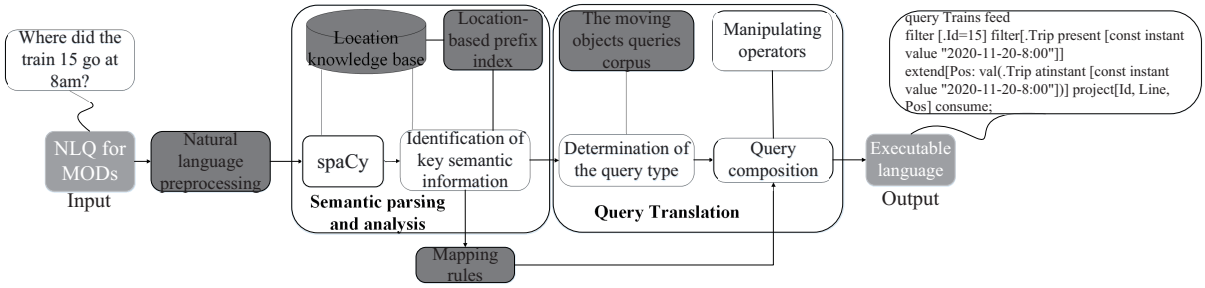

Figure 2: The NALMO architecture in SECONDO

Table 2: Frequently used notations

Name	Abbreviation
A natural language interface for moving objects	NALMO
Natural language interfaces to database	NLIDB
Moving objects database	MOD
Moving objects queries	MOQ
Berlin trains	BT

use parse tree nodes and ontology elements for mapping information. At present, most natural language interfaces of semantic web rely on the use of general dictionaries for resource matching, and WordNet [22] is also used for dictionary expansion. However, general dictionaries are far from meeting the needs of users in certain domains, and thus, specific domain dictionaries should be added. What is more, the lack of benchmark and datasets of NLIDBs also catches much attention. There are already a few datasets such as MAS [17] in *NaLIR* which is suitable for academic papers, authors, conferences and universities. GEO [26] in *PRECISE* is for spatial queries but without time information. None of those works deal with natural language queries in moving objects databases. NALMO proposes a dataset containing multiple moving object queries.

3 THE FRAMEWORK

We provide an overview of NALMO in Figure 2. Key components are (i) *a semantic parser*, (ii) *a location knowledge base*, (iii) *a location-based prefix index*, (iv) *a corpus for query type determination* and (v) *a rule-based algorithm for structured language generation*. At first, the module of natural language processing performs semantic analysis and extracts key entities from the query expression. Next, the query type is determined by a machine learning model and there are four kinds of queries in total, summarized in Table 1. Finally, NALMO maps the entities to queries and combines moving objects operators with entities to produce structured queries. Frequently used notations are summarized in Table 2. Using the query *Q5* in Table 1, we illustrate the workflow as follows.

Semantic interpreter. We use a semantic parser to identify and extract key entities. We preprocess the punctuation of natural language and leverage the tool *spaCy* for word segmentation and entity recognition. The *spaCy* parser enables NALMO to realize that token numbers (e.g., five and 3) and time (e.g., 8am and 10am) are attached to labels CARDINAL and TIME, respectively. Then, we develop a rule-based algorithm to further extract the location, time (e.g., 8am and 10am), the number of nearest neighbors (e.g., five) and the moving objects identifier (e.g., 3) from parsed entities.

Location knowledge base and location-based prefix index. In addition to moving objects, the database also stores relevant point and region objects representing interesting places, e.g., POIs. We find out those objects and put them into the location knowledge base that serves as the dictionary. If the query location does not exist in the dictionary, the location information misses and users will not get the expected results. As location descriptions usually contain a large number of prefixes, we use prefix indices to increase the efficiency of matching between natural language queries and the location knowledge base.

Query type determination. We build a corpus for moving object queries in natural languages and utilize an LSTM neural network to train a model for predicting the query type. For example, in Table 1 the expression *Q5 "Show me five nearest neighbors to the train 3 during [8am, 10am]"* will be reported as a nearest neighbor query.

Structured language generation. NALMO maps the identified key information to the corresponding module in moving objects queries and combines them with defined operators to construct structured queries. The expression *Q5* is a nearest neighbor query, and thus corresponding operators (e.g., *nearestneighbor*(•)) are used to form the expression.

4 NATURAL LANGUAGE UNDERSTANDING

The task is to precisely capture the meaning of a natural language expression and accurately translate the expression into the query language. Database queries have domain specific features or key attributes, which help analyze the semantics. Effectively extracting those information is essential in understanding natural languages. The procedure is as follows. We retain or delete the punctuations, and utilize the tool *spaCy* to perform word segmentation and do entity recognition.

4.1 Natural Language Processing

NALMO deletes character interference items such as double quotation marks (“ ”), single quotation marks (‘ ’), exclamation (!) and the right square bracket (]). The left square bracket ([) should be kept because the symbol guarantees the correct identification of time.

Algorithm 1: SemanticParsing

Input: Q: query expression,
KDB: location knowledge base,
DB: database,
N_list:number list
Output: Q': the parsing result

```
LKB ← {l_1, l_2, ..., l_n};
if token_i.label = TIME then
    Q'.time←token_i;
end
foreach L in LKB do
    if L.l_i = KDB.l_i then
        Q'.location ← the location L.l_i in LKB;
    end
end
if token_i.attribute = mpoint ∧ object_i ∈ token_j then
    Q'.relation ← the relation in DB;
end
foreach token_i in NLQ do
    if token_i = find("neighbor") then
        Q'.number_NN ← token_{i-1};
    end
end
if Q.relation = TRUE ∧ token_i ∈N_list then
    Q'.object_id = token_i;
end
return Q'
```

After the deletion, we use the industrial-grade spaCy parser for word segmentation and entity recognition. Time symbols are labelled TIME, and number symbols are labelled CARDINAL. These time and number symbols are stored for further identification. Numbers may be in the form of English and must be converted to Arabic numbers. If a number is combined with a connector, we replace the connector with a space. For numbers with space such as *"one hundred"*, we use the following regular expression to cut the space and identify each word.

$$re = re.split('[]', n) \tag{1}$$

After removing the conjunction word *"and"* in numbers, we traverse the remaining English digits in reverse order to obtain the digits such as ten, hundred and thousand. These numbers are eventually converted to Arabic numbers.

4.2 Identifying Key Semantic Information

Since entities are approximately processed by spaCy, key information for moving objects queries is not accurate and the results are not sufficient for structured language construction. To increase the translation quality, we propose an algorithm to further parse the spatio-temporal data (Algorithm 1), and l_1 represents the location in the query.

Given a time word with colon such as *"8:00 o'clock"*, we use the following expression for matching and extraction.

$$r'\backslash d + [: :]\backslash d+' \tag{2}$$

The content in "[]" are optional characters. We take both Chinese and English colons into account as users may type them in a mixed way. A Chinese colon may appear in an English sentence, causing the time extraction error. If the time in the query includes a Chinese colon, the colon is replaced by an English one. For the hour without a colon such as *"8am"*, we use the following expression for matching and extraction.

$$r'\backslash d+' \tag{3}$$

Using a regular expression for matching a full hour is simple, but the result cannot be used in the database. The full hour needs to be attached with *":00"*. That is, *"8am"* will be rewritten as *"8:00"* and the terms *"am"* and *"pm"* will be converted to 24-hour format. We traverse the word segmentation results and evaluate each segmented word using the location knowledge base to determine whether the query location exists or not. Query relations are extracted from the database. We analyze the objects in the database and find the relation containing an attribute *mpoint*. The data type *mpoint* is for representing moving objects. This relation has target moving objects. There may be an atomic flow containing only one object with the attribute *mpoint* or a relation that may have multiple moving objects, and the latter is required. An example is provided in Figure 3 in which there is a relation with the schema *Trains*(*Id*:*int*, *Trip*:*mpoint*). *Trains()* is suitable for time interval query and range query. Based on the extracted relation, we can extend the existing relation for different query types. *UnitTrains()* adds a new unit attribute *UTrip* on the basis of *Trains()*. *UTordered()* is sorted by start time on the basis of *UnitTrains()*. The two relations are applied to nearest neighbor query and similarity query. We use the relations in Berlin Trains dataset as examples, but not limited to these relations.

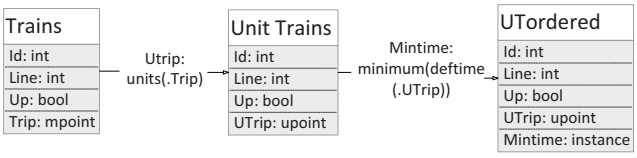

Figure 3: Example of extracted relations in Berlin Trains dataset

The semantics of numbers may be ambiguous in some cases as the number could be the identification of a query object or the number of neighbors. For example, in *Q5*, the moving objects relation in the database is *train*, and the number 5 exists in the number list. Thus, 5 will be an object identifier. In an English query, the number of neighbors must appear before the word *"neighbor"*, or the user may only query the nearest neighbor without the number of neighbors in the sentence. We utilize this knowledge to determine the semantics.

4.3 Location Knowledge Base and Index

Spatial information needs to be captured in moving objects queries. In order to determine the locations, we generate a location knowledge base for each database which contains all the locations included in the database. The location knowledge base extracts objects whose attribute is *point* or *region*, because in SECONDO only objects whose attribute is *point* or *region* can be locations. If the

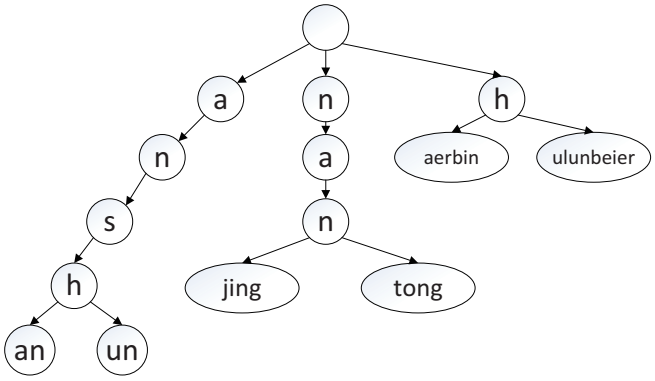

Figure 4: Example of location-based prefix tree

location does not exist in the location knowledge base, we generate corresponding structured languages, but users will not get the expected results.

In order to improve the query efficiency, we construct a location-based prefix index for matching the location knowledge base. A large number of locations often contain the same prefix. Therefore, the index can effectively improve the extraction efficiency of location information. For example, *"Nanjing"* (*Q3*) and *"Nantong"* (*Q4*) have the same prefix *"Nan"*. The root node of the prefix tree is empty, and every child node contains an English character. From the root node to a certain leaf node, the characters passing on the path are connected, which is the location corresponding to the nodes. The characters contained in all child nodes of each node are different from each other. Characters that repeat continuously from the first character occupy only one node. Besides, we merge some nodes with only one child node as the leaf nodes to reduce memory consumption. In the process of prefix tree construction, whenever there is a location, we will start matching from the root node. If there is no matching node for the character, we will create a new node and continue to match the next character. If the node already exists, we follow the path to continue matching. An example of a location-based prefix tree is shown in Figure 4. With such prefix trees, locations can be determined efficiently.

5 QUERY TRANSLATION

The translation consists of two steps: (i) *determining the query type* and (ii) *constructing the structured language*. Semantic understanding and parsing of natural languages produce key entities and map them to structured query languages. There are different types of queries and a corpus is built to determine the query type. The corpus makes use of an LSTM neural network to train a model for identifying the query type. The natural language query needs to be proposed according to a certain grammar, and the moving objects query must contain the key information. Therefore, the corpus is more important for the result of query type determination, and the choices of neural network classifiers have less influence. NALMO selects the mapping method and combines the affluent operators to form structured queries.

5.1 Corpus: Determining Query Type

Providing a common entity mapping and query combination method is a challenging task because moving object queries contain specific key information which require different operators. One solution of determining the query type is to use defined rules, but the strategy will produce a set of fixed outputs. When the natural language expression changes, the method probably produces incorrect results. The determination effect primarily depends on the quality of natural language processing tools.

To have high quality processing, we build a moving object corpus. The corpus contains 1200 moving objects queries classified into four categories including (1) time interval query, (2) range query (3) nearest neighbor query, (4) trajectory similarity query. It is worth noting that time interval query includes time point query. Range query includes spatial query and spatio-temporal range query. In the process of classification and training, we also consider the special cases of time interval query and range query. We initially collect 300 moving objects query examples from more than 60 published conference and journal papers, and then use experts' knowledge, entity information replacement and sentence expression replacement to enhance the knowledge. Experts utilize various key information with different semantic expressions according to the query type. These queries satisfy the diversity of semantic expression properly, and the corpus is extended up to 1200 moving objects queries. The experts participating in our work specialize in database, specifically MODs. We manually mark each query and utilize the LSTM neural network to train the model.

Data extraction is guided by two criteria: (i) *knowledge of moving objects experts* and (ii) *citation-based impact factors from the Institute for Scientific Information (ISI)*. We use popular academic conferences and journals as references, including 34 journal papers and 28 conference papers. Experts judge the quality of natural language queries in articles and rank these queries. ISI calculates the impact factor of the journal based on its citation index. The assumption is that articles in higher-quality journals and conferences are likely to provide comprehensive queries. Each journal and conference receives a ranking score. The highest ranking journal or conference receives a score 1, and the next highest ranking receives a score 1/2. The ranking score is used as a reference, and the final score is given by the expert. Articles published in the same journal have the same scores. Domain experts rank query examples according to the query robustness, and the maximum score is 1. Then, the article score and the expert score are summarized as the total score, and moving objects queries ranking from high to low will be added in the corpus. Journal and conference scores are given in Table 3.

We can support other kinds of queries such as join, as long as the underlying related functions or operations are implemented.

5.2 Constructing Structured Languages

After the semantic understanding, we get the query type and map the entities to the corresponding positions according to defined rules. Finally, we combine the suitable operators to form the final structured sentence. The steps are reported in Figure 5. The template of the query expression is as follows:

*query <data relation> feed filter [<**special operators**> <values>] extend <**special operators**> [<values>] consume;*

Table 3: Scores of journals and conferences

Journal and Conference Proceeding	Score
ACM Conference on Management of Data	1
IEEE Transactions on Knowledge and Data Engineering	1
Proceedings of the VLDB Endowment	1
IEEE International Conference on Data Engineering	1
The VLDB Journal	1
GeoInformatica	1/2
International Conference on Extending DB Technology	1/2
ACM SIGSPATIAL	1/2
Journal of Computer Science and Technology	1/2
Journal of Database Management	1/3
International Symposium on Spatial and Temporal Databases	1/3
International Conference on Mobile Data Management	1/3

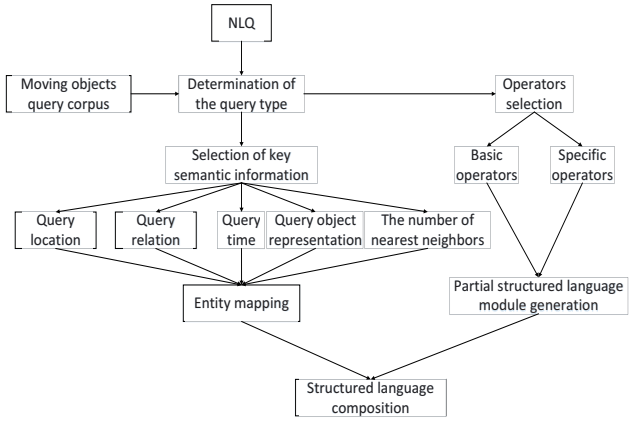

Figure 5: Building structured language queries

Extracted entities are mapped to the data relation and corresponding values according to the query type. At the same time, operators are selected to constitute the query structure. Some queries with specific objects need to add *filter [.<Id> = <number>]* or the id of the object to fill in special operators. We support four types of queries: (i) *time interval queries*, (ii) *range queries*, (iii) *nearest neighbor queries* and (iv) *trajectory similarity queries*. To help explain the procedure, we use an example relation storing underground trains as moving objects. The schema is *Trains*(Id: int, Trip: mpoint). Table 4 summarizes kernel operators for processing moving objects.

Definition 1. ***Time interval queries***
Given an interval $[t_1, t_2]$*, the query looks for objects whose time intervals intersect* $[t_1, t_2]$.

In order to compare the query time with a trajectory's time, the operator *deftime* is utilized to extract the defined time interval of a trajectory. The operator *filter* evaluates a query predicate and returns objects fulfilling the condition. For this query, an intersecting between two time intervals is evaluated. The template of the query expression is as follows in which the content in <> will be extracted from natural language expressions.

query <data relation> feed filter[***deftime****(.Trip)* ***intersects*** *<query time>*] *consume;*

Table 4: Operators for moving objects

Operator	Signature
trajectory	$mpoint \longrightarrow line$
atperiods	$mpoint \times periods \longrightarrow mpoint$
present	$mpoint \times periods\ (instant) \longrightarrow bool$
deftime	$mpoint \longrightarrow periods$
at	$mpoint \times region \longrightarrow mpoint$
passes	$mpoint \times region\ (point) \longrightarrow bool$
atinstant	$mpoint \times instant \longrightarrow ipoint$
inst	$ipoint \longrightarrow instant$
val	$ipoint \longrightarrow point$
intersects	$xrectangle \times xrectangle \longrightarrow bool$

Assuming the query time is "*[7:00, 8:00] on November 20, 2020*", we get the following structured language:

query Trains feed filter[***deftime****(.Trip)* ***intersects*** [*const periods value (("2020-11-20-7:00" "2020-11-20-8:00" TRUE TRUE))*]] consume;

There is another approach to accomplish the task by evaluating whether an object is defined at the query time. The operator *atperiod* restricts the movement to a time interval and the operator *present* determines whether an object is valid in a certain time interval. The template is:

query <data relation> feed filter[***present****(.Trip* ***intersects*** *<query time>)*] *consume;*

Definition 2. ***Spatio-temporal range queries***
Given a spatio-temporal window Q and a set of trajectories O, the query returns a set of trajectory O', and for $\forall$ *O'* $\subset$ *O, O' intersects Q(t,r).*

The special operator *intersect* finds objects that satisfy both time and space conditions. The operator *intersect* can be used to check whether two rectangles cross each other. The operator *deftime* can get the time interval when the moving object is limited in the specific space condition. We use the operation *intersects* to get the intersection between the limited space and the limited time to construct the query. Assuming that the location is *"Tiergarten"* and the time interval is November 20, 2020 from 8:00 to 10:00, we query trains that meet two conditions above:

query Trains feed filter [(deftime(.Trip at Tiergarten) intersects [const periods value (("2020-11-20-8:00" "2020-11-20-10:00" TRUE TRUE))])]consume;

When the query does not specify the specific query object identifier, the structured expression is as follows:

*query <relation> feed filter [(****deftime****(.<Trip> at <region>)* ***intersects*** *<period>)]consume;*

Definition 3. ***Nearest Neighbor Queries***
Given a query mq, a distance set D and a relation E with an attribute MO of type mpoint, the query which holds all the distances between

the queried moving objects returns a subset $E' \subset E$, where each tuple meets the condition that for the tuple $e \in E'$, there exists an instance of time t. The distance d(e.MO(t), mq(t)) $\in D$ is among the k smallest distances.

We use the operator *knearest* for nearest neighbor query. Before querying, we need to create a unit representation of the moving object. The relation *Trains*(*Id*:*int*, *Trip*:*mpoint*) is used as an example:

let UnitTrains = Trains feed ***projectextendstream****[Id, Line, Up; UTrip: units(.Trip)] consume;*

The operator *projectextendstream* creates a unit stream from the *Trip* attribute of each input tuple, and outputs a copy of the original tuple for each unit, which is limited to attributes *Id*, *Line*, *Up* and the new unit attribute *UTrip*. Parameters of the operator *knearest* are ordered stream of unit tuples, the name of the attribute *a*, the queried moving object and the number of neighbors *k*.

The operator *knearest* returns a stream of tuples with the same structure as the input stream. However, the output units in attribute belong to the *k* nearest moving points represented in the input stream and are limited by the time when objects arrive between the *k* nearest moving points, and requires the first parameter stream to arrive in the order of units' start time. We create a version of *UnitTrains*, where units are sorted by start time:

*let UTOrdered = UnitTrains feed extend[Mintime: minimum(****deftime****(.UTrip))] sortby[Mintime asc] consume;*

We can get the final structured language by using *UTOrdered*:
query ***UTOrdered*** *feed filter [(****deftime****(.UTrip)* ***intersects*** *<period>)]* ***knearest****[UTrip, <object>, <k>] consume;*

Definition 4. ***Trajectory similarity queries***
Given a set of trajectories denoted by O, the query returns the trajectory $o_i \in O$, of which the similarity distance measure m_i is the smallest during the same point t or time interval $[t_1, t_2]$.

If we can find the closest moving object to the query object within a period of time, we can judge that the trajectories of the two moving objects are similar. The operator *knearest* looks for the nearest neighbors in a trajectory segment, but multiple trajectory segments may have appeared in the time interval of a query. We count each nearest neighbor, and the object that has the most frequent occurrence in the time interval is judged to be similar. If the number of occurrences is the same, the longest trajectory has priority. Using *UTOrdered* sorted by start time, we get the final structured language:

query ***UTOrdered*** *feed filter [(****deftime****(.UTrip)* ***intersects*** *<period>)]* ***knearest****[UTrip, <object>, 1] consume;*

6 EXPERIMENTAL EVALUATION

The evaluation is conducted in a desktop PC (Intel(R) Core(TM) i7-8700CPU, 3.6 GHz, 8 GB memory, 1 TB disk) running Ubuntu 14.04 (64 bits, kernel version 4.8.2-19). We develop the proposal in C/C++ and integrate the implementation into an extensible database system SECONDO. Our system has been developed in a prototype system for translating natural language queries into structured query languages [34].

6.1 Datasets

To the best of our knowledge, there is no benchmark of moving objects queries in natural languages. Therefore, we generate two datasets named Moving Objects Queries (MOQ) and Berlin Trains (BT), respectively.

MOQ contains 240 moving objects queries divided into four categories: (1) time interval queries, (2) range queries, (3) nearest neighbor queries and (4) trajectory similarity queries. These queries are extracted from more than 60 conference and journal papers in the domain of moving objects and then manually processed by experts. Specifically, we select high-quality queries from the corpus to construct MOQ. MOQ is used to test whether different kinds of moving object queries can get correct structured languages. BT contains moving underground trains and is used to evaluate whether the converted structured sentences can provide expected results for users. Some queries can be seen in the appendix, and these queries satisfy the diversity of expressions as much as possible.

6.2 Measurement

NALMO is designed for users who are not experts in writing query expressions for moving objects. Two criteria are defined to evaluate the translation quality: (i) *effectiveness* and (ii) *time cost*. Effectiveness mainly contains two measurements: *accuracy* and *precision*.

Effectiveness. NALMO is designed for users to conveniently query and obtain query results, so whether executable languages can be generated and whether users can get expected results are important factors. The way to verify the effectiveness is to check whether natural language expressions are correctly and precisely translated into structured query languages, that is, *accuracy*. Such a value is defined as the ratio of the number of correctly translated sentences to the overall number of sentences. MOQ is used for test. In the evaluation, domain experts' judgement is considered to determine whether the produced structured query correctly corresponds to the natural language query. However, only using the accuracy is not sufficient to verify the effectiveness of NLIDB systems. This is because users may be not concerned about the generation of structured sentences but pay attention to whether the outputs are the expected results. Motivated by this, we use the *precision* to further evaluate the translation effect. The *precision* is defined as the ratio of the number of queries producing the expected results in the system to the overall number of queries. BT is used for test. In some cases, the natural language expression is unclear or the key information is missing. Therefore, NALMO generates a structured query, but the query fails to output the expected results.

Time Cost. We do the evaluation by considering (i) *the overall time cost* and (ii) *the time cost in each module*, including parsing, analyzing and constructing the structured languages. We take the average value over three runnings as the final result.

6.3 Results

Figure 6 shows the effectiveness. The overall accuracy of NALMO is 98.1%. Among 7 wrong queries, there are 2 time point queries, 2 time interval queries, 1 spatio-temporal range query, and 2 trajectory similarity queries. The time format of 3 expressions cannot be recognized by spaCy, and NALMO terminates the constructing

Table 5: Accuracy, precision and time cost between template matching and model

Query Types	Accuracy1[a] (%)	Accuracy2[b] (%)	Precision1 (%)	Precision2 (%)	Time Cost1 (s)	Time Cost2 (s)
Time Point Query	96.7	96.7	86.7	86.7	1.33	1.26
Time Interval Query	96.7	96.7	86.7	86.7	1.34	1.36
Spatial Query	100	90	86.7	86.7	1.39	1.28
Spatio-temporal Query	98.3	88.3	88.3	88.3	1.4	1.36
Nearest Neighbor Query	100	86.7	96.7	88.3	1.66	1.41
Trajectory Similarity Query	96.7	85	88.3	76.7	1.68	1.49

[a]Using the trained model to determine the query type. [b]Using template matching to determine the query type.

procedure. There are 2 queries vaguely expressing the time information. Specifically, the word *"right now"* in *"Where are the trains right now?"* cannot be recognized by NALMO as moving objects queries require accurate time information. In trajectory similarity queries, we only support returning the most similar trajectory at the moment but not k (> 2) similar trajectories.

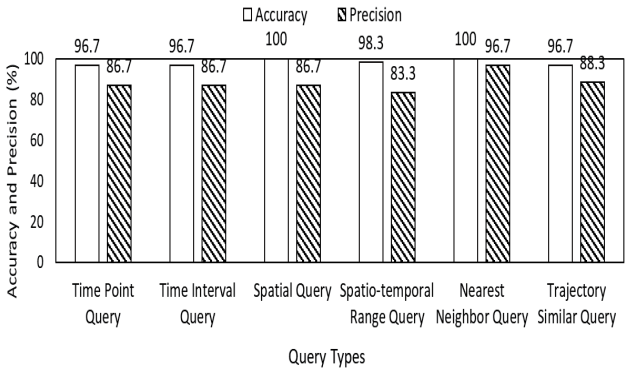

Figure 6: Accuracy and precision

The overall precision is 88.1%. The value is lower than accuracy. This is because some queries are translated into executable structured languages but did not report expected results. Among 34 wrong queries, the time point query contains 6 wrong sentences. The reason is that the relation in the structured sentence is *Planes* which does not exist in the dataset. The time interval query contains 6 wrong sentences among which 4 are spatio-temporal range queries and 2 queries do not find required relations. The spatial query contains 8 wrong sentences. There are 6 queries without locations in the knowledge base such that structured languages are not executable in the database. There are 2 queries looking for non-existent objects. For example, *"train G55"* in *"Where was the train G55 at 12:00 o'clock?"* is an object that cannot be queried. The spatio-temporal range query contains 10 wrong sentences among which 6 queries have locations not in the knowledge base and 4 queries do not have required relations. The trajectory similarity query contains 5 wrong sentences. The queries are answered by using algorithms of nearest neighbor queries. The nearest neighbor query contains 2 wrong sentences both of which are identified as similarity queries. The dataset size has a significant impact on the precision of users' expected query results.

The time costs are reported in Figure 7. Nearest neighbor queries and trajectory similarity queries take more time than other queries, the average time of which are 1.66s and 1.68s, respectively. The two queries consider the time, location, query object identifier, and the number of nearest neighbors, requiring more entities than other queries. The time cost includes two parts: (i) *semantic parsing and analysis*, and (ii) *the determination of query types and the combination of structured languages*. The average cost of natural understanding is 1.17s, and the average time cost of query translation is 0.3s. The main part of time is spent on semantic parsing and analysis. The parsing time is affected by the speed of natural language processing tools to a certain extent.

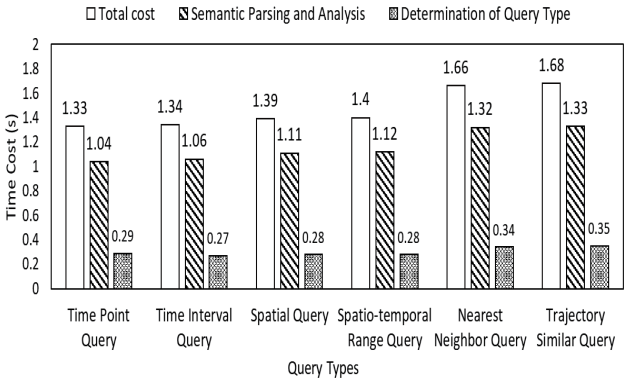

Figure 7: Time cost

When we use template matching instead of the trained model to determine the query type, accuracy and precision decrease. Time cost is almost the same as before. Table 5 shows that the overall accuracy and precision are 90.6% and 83.9%, respectively, which are significantly lower than using the trained model. In terms of accuracy, there is no change in time point query and time period query. Spatial query and spatio-temporal range query all add 6 new wrong queries. Although the location is not included in the location knowledge base, the model can still determine the query type and structured queries can be generated. However, template matching lacks location information to directly generate structured queries. The nearest neighbor query and trajectory similarity query add 8 and 7 wrong sentences, respectively. This is because nearest neighbor query and trajectory similarity query are matched by keywords such as "*neighbor*" and "*similar*". Due to the limitations

of template matching, when the keywords do not appear in the sentence, the query cannot be recognized. In terms of precision, there is no change in time point query and time interval query. Spatial query and spatio-temporal range query remain unchanged. This is because the use of model determination does not generate structured sentences, and NALMO will not give user expected results. Nearest neighbor query and trajectory similarity query add 8 and 7 wrong queries which all do not generate structured sentences. In terms of time cost, there is not much difference compared with using the model which has little effect on user experience.

7 CONCLUSION

We design and develop a natural language interface for moving objects databases. The proposal supports translating four types of queries. Experiment results demonstrate the advantage of our method in terms of accuracy, precision and efficiency. An important task in the future is to improve the fault tolerance and the association prompts of input errors. When the input is not successfully sent into the system, NALMO is able to determine whether the user has entered similar information before. Another important task is to make NALMO support more query types such as join and efficiently utilize index structures.

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China (61972198), Natural Science Foundation of Jiangsu Province of China (BK20191273).

REFERENCES

[1] Yael Amsterdamer, Anna Kukliansky, and Tova Milo. 2015. NL_2CM: A Natural Language Interface to Crowd Mining. In *ACM SIGMOD*. 1433–1438.
[2] Joel Booth, Barbara Di Eugenio, Isabel F. Cruz, and Ouri Wolfson. 2015. Robust Natural Language Processing for Urban Trip Planning. *Appl. Artif. Intell.* 29, 9 (2015), 859–903.
[3] V. Prasad Chakka, Adam Everspaugh, and Jignesh M. Patel. 2003. Indexing Large Trajectory Data Sets With SETI. In *CIDR*.
[4] Hae Don Chon, Divyakant Agrawal, and Amr El Abbadi. 2003. Range and kNN Query Processing for Moving Objects in Grid Model. *Mob. Networks Appl.* 8, 4 (2003), 401–412.
[5] Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. 2011. FREyA: An Interactive Way of Querying Linked Data Using Natural Language. In *ESWC (Lecture Notes in Computer Science, Vol. 7117)*. 125–138.
[6] Alessandra Giordani and Alessandro Moschitti. 2012. Translating Questions to SQL Queries with Generative Parsers Discriminatively Reranked. In *COLING*. 401–410.
[7] Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. DialSQL: Dialogue Based Structured Query Generation. In *ACL*. 1339–1349.
[8] R.H. Güting, V.T. de Almeida, and Z.M. Ding. 2006. Modeling and Querying Moving Objects in Networks. *VLDB Journal* 15, 2 (2006), 165–190.
[9] Ralf Hartmut Güting, Thomas Behr, and Christian Düntgen. 2010. SECONDO: A Platform for Moving Objects Database Research and for Publishing and Integrating Research Implementations. *IEEE Data Eng. Bull.* 33, 2 (2010), 56–63.
[10] Ralf Hartmut Güting, Thomas Behr, and Jianqiu Xu. 2010. Efficient k-nearest neighbor search on moving object trajectories. *VLDB J.* 19, 5 (2010), 687–714.
[11] Ralf Hartmut Güting and Markus Schneider. 2005. *Moving Objects Databases*. Morgan Kaufmann.
[12] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a Neural Semantic Parser from User Feedback. In *ACL*. 963–973.
[13] C.S. Jensen, H. Lu, and B. Yang. 2009. Graph Model Based Indoor Tracking. In *MDM*.
[14] Zhen Jia, Abdalghani Abujabal, Rishiraj Saha Roy, Jannik Strötgen, and Gerhard Weikum. 2018. TEQUILA: Temporal Question Answering over Knowledge Bases. In *ACM CIKM*. 1807–1810.
[15] Joung-Joon Kim, Dong-Suk Hong, Hong-Koo Kang, and Ki-Joon Han. 2006. TMOM: A Moving Object Main Memory-Based DBMS for Telematics Services. In *W2GIS 2006 (Lecture Notes in Computer Science, Vol. 4295)*. 259–268.
[16] Bofang Li, Aleksandr Drozd, Yuhe Guo, Tao Liu, Satoshi Matsuoka, and Xiaoyong Du. 2019. Scaling Word2Vec on Big Corpus. *Data Sci. Eng.* 4, 2 (2019), 157–175.
[17] Fei Li and H. V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *Proc. VLDB Endow.* 8, 1 (2014), 73–84.
[18] Jingjing Li, Wenlu Wang, Wei-Shinn Ku, Yingtao Tian, and Haixun Wang. 2019. SpatialNLI: A Spatial Domain Natural Language Interface to Databases Using Spatial Comprehension. In *ACM SIGSPATIAL*. 339–348.
[19] Yunyao Li and Davood Rafiei. 2018. *Natural Language Data Management and Interfaces*.
[20] Yunyao Li, Huahai Yang, and H. V. Jagadish. 2007. NaLIX: A generic natural language search environment for XML data. *ACM Trans. Database Syst.* 32, 4 (2007), 30.
[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*. 3111–3119.
[22] George A. Miller. 1995. WordNet: A Lexical Database for English. *Commun. ACM* 38, 11 (1995), 39–41.
[23] Nikos Pelekis, Elias Frentzos, Nikos Giatrakos, and Yannis Theodoridis. 2008. HERMES: aggregative LBS via a trajectory DB engine. ACM, 1255–1258.
[24] Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis. 2000. Novel Approaches in Query Processing for Moving Object Trajectories. In *VLDB*. 395–406.
[25] Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability. In *COLING*.
[26] Ana-Maria Popescu, Oren Etzioni, and Henry A. Kautz. 2003. Towards a theory of natural language interfaces to databases. In *IUI*. 149–157.
[27] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R. Mittal, and Fatma Özcan. 2016. ATHENA: An Ontology-Driven System for Natural Language Querying over Relational Data Stores. *Proc. VLDB Endow.* 9, 12 (2016), 1209–1220.
[28] Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Özcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish R. Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. 2020. ATHENA++: Natural Language Querying for Complex Nested SQL Queries. *Proc. VLDB Endow.* 13, 11 (2020), 2747–2759.
[29] Niculae Stratica, Leila Kosseim, and Bipin C. Desai. 2005. Using semantic templates for a natural language interface to the CINDI virtual library. *Data Knowl. Eng.* 55, 1 (2005), 4–19.
[30] Alane Suhr, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to Map Context-Dependent Sentences to Executable Formal Queries. In *NAACL-HLT*. 2238–2249.
[31] Yufei Tao and Dimitris Papadias. 2001. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In *VLDB*. 431–440.
[32] Peihao Tong, Qifan Zhang, and Junjie Yao. 2019. Leveraging Domain Context for Question Answering Over Knowledge Graph. *Data Sci. Eng.* 4, 4 (2019), 323–335.
[33] Michalis Vazirgiannis, Yannis Theodoridis, and Timos K. Sellis. 1998. Spatio-Temporal Composition and Indexing for Large Multimedia Applications. *Multim. Syst.* 6, 4 (1998), 284–298.
[34] Xieyang Wang, Jianqiu Xu, and Yaxin Wang. 2020. NLMO: Towards a Natural Language Tool for Querying Moving Objects. In *MDM 2020*. IEEE, 228–229.
[35] Xiaopeng Xiong, Hicham G. Elmongui, Xiaoyong Chai, and Walid G. Aref. [n.d.]. Place: A Distributed Spatio-Temporal Data Stream Management System for Moving Objects. In *(MDM*.
[36] J. Xu and R. H. Güting. 2013. A Generic Data Model for Moving Objects. *GeoInformatica* 17, 1 (2013), 125–172.
[37] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. SQLizer: query synthesis from natural language. *Proc. ACM Program. Lang.* 1, OOPSLA (2017), 63:1–63:26.
[38] Zhicheng Zheng, Fangtao Li, Minlie Huang, and Xiaoyan Zhu. 2010. Learning to Link Entities with Knowledge Base. In *HLT-NAACL*. 483–491.

APPENDIX

Table 6: Moving objects queries in natural language

Id	Query Expression
Q1	Which places did the trains go between 9:00 and 10:00 o'clock?
Q2	Which trains pass the station Mehringdamm from 8:00 to 10:00?
Q3	Please tell me the position of train 3 between 11am and 1pm?
Q4	Who can tell me which train has the similar trajectory of the train 134 at 7am?
Q5	Which trains are the 5 continuous nearest neighbors of the train 345 from 8:00 to 9:00 o'clock?
Q6	Did any trains pass Westhafen between 1pm and 1:30pm?
Q7	Find the trains that pass Mehringdamm from 8:20 to 10:59.
Q8	Which train is the nearest neighbor of the train 123 from 9am to 10am?
Q9	Which train is the nearest neighbor to the train 345 between [9am, 10am]?
Q10	Could you tell me which train is similar to the train 77 during [5pm, 9pm]?
Q11	What is the location of the train 1 from 8:00 to 9:10?
Q12	Please find almost 20 continuous nearest neighbors of the train 28 which is in Nanjing between 2am and 4am.
Q13	Which trains arrive at the Zoogarten between 8:18 and 10:27?
Q14	May I know the similar trajectory of the taxi 2 from 1:13 to 2:24?
Q15	What is the location of train 44 at 23:01?
Q16	Do you know the route of the train 56 between 7:00am and 16:00pm?
Q17	Query the trains that have left from 11am to 2pm.
Q18	Find the periods that the train 11 went to Alexanderplatz.
Q19	Did any trains pass the station Alexanderplatz during [11am, 12am] today?
Q20	Please tell me which trains have been to the Mehringdamm station from 8:20 to 10:00?
Q21	Find precisely seven closest neighbors of the train 8 during [10am, 6pm].
Q22	Which train has the similar trajectory of the train 17 between 17:00 and 19:00 o'clock?
Q23	Would you mind telling me the trajectory that is close to the fastest train 29 at 18:19 in Beijing?
Q24	May I know which train is similar to the train 1 that passes Sydney Opera House near me?
Q25	Tell me which train has the similar trajectory of the train 37 at Shanghai station during [7:19, 11:19].
Q26	Please tell me which trains have been to the Koepenick from 8:20 to 10:00?
Q27	Which train is the nearest neighbor to the train 345 between [9am, 10am]?
Q28	Where was the train 5 at 17:00 o'clock on Saturday?
Q29	Find the periods that train 17 passed by the Arch of Triumph?
Q30	From 8am to 9am, which trains are five nearest neighbors to the train 47?