A Natural Language Interface for Querying General and Individual Knowledge (Full Version)

Yael Amsterdamer, Anna Kukliansky, and Tova Milo

Tel Aviv University

{yaelamst,annaitin,milo}@post.tau.ac.il

Preprint

Abstract

Many real-life scenarios require the joint analysis of *general knowledge*, which includes facts about the world, with *individual knowledge*, which relates to the opinions or habits of individuals. Recently developed crowd mining platforms, which were designed for such tasks, are a major step towards the solution. However, these platforms require users to specify their information needs in a formal, declarative language, which may be too complicated for naïve users. To make the joint analysis of general and individual knowledge accessible to the public, it is desirable to provide an interface that translates the user questions, posed in natural language (NL), into the formal query languages that crowd mining platforms support.

While the translation of NL questions to queries over conventional databases has been studied in previous work, a setting with mixed individual and general knowledge raises unique challenges. In particular, to support the distinct query constructs associated with these two types of knowledge, the NL question must be partitioned and translated using different means; yet eventually all the translated parts should be seamlessly combined to a well-formed query. To account for these challenges, we design and implement a modular translation framework that employs new solutions along with state-of-the art NL parsing tools. The results of our experimental study, involving real user questions on various topics, demonstrate that our framework provides a high-quality translation for many questions that are not handled by previous translation tools.

1 Introduction

Real-life data management tasks often require the joint processing of two types of knowledge: *general knowledge*, namely, knowledge about the world

independent from a particular person, such as locations and opening hours of places; and *individual knowledge*, which concerns the distinct knowledge of each individual person about herself, such as opinions or habits. Distinguishing the two types of knowledge is crucial for harvesting and processing purposes [1]. Consider the following scenario, based on a real question in a travel-related forum: a group of travelers, who reserved a room in Forest Hotel, Buffalo, NY[1], wishes to know "What are the most interesting places near Forest Hotel, Buffalo, we should visit in the fall?" Note that answering this question requires processing mixed general and individual knowledge: the sights in Buffalo and their proximity to Forest Hotel is general knowledge that can be found, e.g., in a geographical ontology; in contrast, the "interestingness" of places and which places one "should visit in the fall" reflect individual opinions, which can be collected from people. (See additional, real-life examples in Section 6.)

In recent work, we introduced *crowd mining* as a novel approach for answering user questions about a mix of individual and general knowledge, using crowdsourcing techniques [1, 2]. In particular, we have implemented the `OASSIS` platform [2], which supports a declarative query language, `OASSIS-QL`, enabling users to specify their information needs. Queries are then evaluated using both standard knowledge bases (ontologies) and the crowd of web users. While this approach is a major step towards a solution, the requirement to specify user questions as queries is a major shortcoming. One cannot expect naïve users (like the travelers in our example above) to write such complex queries. To make the joint analysis of general and individual knowledge (and crowd mining in particular) accessible to the public, it is desirable to allow users to specify their information needs in natural language (NL).

We therefore develop a principled approach for the translation of NL questions that mix general and individual knowledge, into formal queries. This approach is demonstrated for translating NL questions into `OASSIS-QL`, however, we explain throughout the paper how the principles that we develop can apply to other scenarios.

Previous work The NL-to-query translation problem has been studied in previous work for queries *over general, recorded data*, including SQL/XQuery/SPARQL queries (e.g., [10, 18, 19]). The work of [18], for instance, studies the translation of NL requests to SQL queries, while matching the question parts to the database and interacting with users to refine the translation. An important prerequisite for such translation is the availability of NL tools for parsing and detecting the semantics of NL sentences. Such parsers have also been extensively studied [21, 22, 25]. *We therefore build upon these foundations in our novel solution.* We do not aim to resolve clas-

Preprint

[1]The hotel name is masked, for privacy.

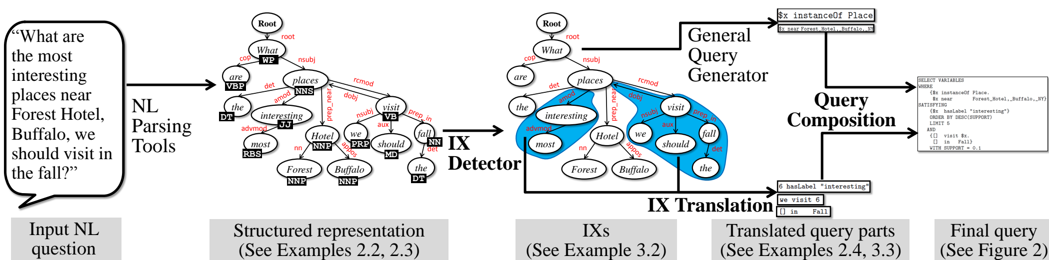

Figure 1: Overview of the NL to query translation process. Enlarged figures of each step appear separately in the sequel.

sic problems such as handling ambiguities and uncertainty in NL parsing, but rather embed existing tools as black-boxes in a novel, generic framework, and focus on developing the new components required in our setting.

New challenges The mix of general and individual knowledge needs lead to unique challenges, as follows.

- The general and individual parts of an NL question may be mixed in an intricate manner, and *must be distinguished* from each other to allow separate evaluation. This is non-trivial: naïve approaches, such as checking which parts of the user question *do not match* items in a general ontology, cannot facilitate this task since most ontologies do not contain all and only general knowledge.
- The parts of the NL question that deal with individual knowledge may need to be translated into formal query constructs *without aligning them first to a structured knowledge base* (as they may relate to the not-yet-collected knowledge of the crowd). Hence, techniques of existing translators, which are mostly based on such alignment [10, 18, 19], cannot be applied.
- The translated general and individual query parts need to be seamlessly *integrated* into a well-formed query. This must be done carefully, since the two types of query parts are translated by separate means.

Note that existing NL tools such as opinion mining *can detect only a fragment of individual expressions*, namely, references to opinions and preferences, but cannot be used for identifying, e.g., phrases related to individual habits.

Contributions To address the new challenges, we make two distinct contributions. **Our first contribution is the modular design of a translation framework**, which embeds both state-of-the-art NL parsing tools, where possible, and novel modules. The main steps of the translation process are sketched in Figure 1. (For convenience, beneath each step there are pointers to relevant examples and enlarged figures.) **The development of**

Preprint

new modules is our second contribution. (these modules are highlighted in bold in Figure 1). From left to right: The input NL question is converted, by standard NL parsing tools, into a well-defined structure that captures the semantic roles of text parts. A new *Individual eXpression (IX) Detector* then serves to decompose the structure into its general and individual parts (the latter are highlighted by a colored background). A major contribution here is in providing a semantic and syntactic definition of an IX, as well a means for detecting and extracting IXs. This is done via declarative selection patterns combined with dedicated vocabularies. Each general and individual part is separately processed, and in particular, an existing General Query Generator is used to process the general query parts, whereas the individual parts are processed by our new modules. The processed individual and general query parts are integrated, via another new module, to form the final output query.

Implementation and Experiments We have implemented these techniques in a novel prototype system, $\mathtt{NL_2CM}$ (Natural Language interface to Crowd Mining). Beyond the automated translation, this system interacts with the user in order to complete missing query parameters. We have then conducted an extensive experimental study, for testing the effectiveness of $\mathtt{NL_2CM}$. To examine the quality of the translation, we have allowed users to feed questions into $\mathtt{NL_2CM}$ in a real usage scenario, and analyzed the resulting queries. Then, we have tested the applicability of our approach to real questions on wide-ranging topics, by using $\mathtt{NL_2CM}$ to translate questions from Yahoo! Answers, the question-and-answer platform [34]. In both cases, we show that by composing simple techniques and standard NL tools our system can achieve high success rates, for a variety of topics and question difficulties. *In particular, we have shown that* $\mathtt{NL_2CM}$ *accounts, for the first time, for a significant portion of the real-life user questions, concerning individual knowledge, that was not handled by previous translation systems.*

The $\mathtt{NL_2CM}$ system was demonstrated at SIGMOD'15 [3]. The short paper accompanying the demonstration provides only a high level description of the system design and its usage scenario. The present paper provides a full description the principles and novel modules underlying the system.

Paper organization We start in Section 2 by reviewing technical background. Then, in Sections 3 and 4 we explain the translation process and the new modules, elaborating on the IX detection module. Our implementation of $\mathtt{NL_2CM}$ and the experimental study are described in Sections 5 and 6, respectively. Related work is discussed in Section 7, and we conclude in Section 8.

Preprint

2 Preliminaries

NL to query translation requires representing two types of information: the representation of the original NL question should facilitate an automated analysis of its meaning; and the representation of the queried knowledge, as reflected by the query language, should capture the way it is stored and processed. The two types of representation may be very different, and thus, an important goal of the translation process is mapping one to the other. We next overview the technical background for our translation framework, including the knowledge representation and the NL parsing tools used to generate the NL representation, with the concrete example of translating of NL to `OASSIS-QL`.

2.1 Knowledge Representation

In our setting, knowledge representation must be expressive enough to account for both general knowledge, to be queried from an ontology, and for individual knowledge, to be collected from the crowd. One possible such representation is RDF[2], which is commonly used in large, publicly available knowledge bases such as DBPedia [11] and LinkedGeoData [20]. This is also the representation used in `OASSIS`.

We next provide some basic definitions for individual and general knowledge in RDF representation, starting with *RDF facts*, which form the basic building block for knowledge.

Definition 2.1 (Facts and fact-sets) *Let $\mathcal{E}$ be a set of* element names *and $\mathcal{R}$ a set of* relation names*. A* fact *over $(\mathcal{E}, \mathcal{R})$ is defined as $f \in \mathcal{E} \times \mathcal{R} \times (\mathcal{E} \cup \Sigma^*)$, where Σ^* denotes unrestricted* literal terms *over some alphabet. A fact-set is a set of facts.*

Preprint

The names in facts correspond to meaningful terms in natural language. Elements in $\mathcal{E}$ can be nouns such as `Place` or `Buffalo`; and relations in $\mathcal{R}$ can be, e.g., relative terms such as `inside` or `nearby`, verbs such as `visited` or `purchased`, etc. We denote facts using the RDF notation $\{e_1 \ \ r \ \ e_2\}$ or $\{e_1 \ \ r \ \ \text{“}l\text{”}\}$ where $e_1, e_2 \in \mathcal{E}$, $r \in \mathcal{R}$ and $l \in \Sigma^*$.[3] As in RDF, facts within a fact-set are concatenated using a dot.

General knowledge is captured in our model by an *ontology* Ψ, which stores this type of knowledge in some representation form. In the case of RDF representation, it is a fact-set of general knowledge facts. E.g., Ψ may contain the fact {`Buffalo,_NY inside USA`}.

Individual knowledge can be modeled by facts capturing the actions and opinions of a crowd member [2]. For example, the belief that Buffalo,

[2] http://www.w3.org/TR/rdf-schema

[3] The curly brackets around facts are added for readability.

```
1  SELECT VARIABLES $x
2  WHERE
3      {$x instanceOf Place.
4       $x near       Forest_Hotel,_Buffalo,_NY}
5  SATISFYING
6      {$x  hasLabel "interesting"}
7      ORDER BY DESC(SUPPORT)
8      LIMIT 5
9     AND
10     {[]  visit $x.
11      []  in   Fall}
12     WITH SUPPORT THRESHOLD = 0.1
```

Figure 2: Sample OASSIS-QL Query, $\mathcal{Q}$

NY is interesting can be expressed by the fact {Buffalo,_NY hasLabel "interesting"}, and the habit of visiting Buffalo can be expressed by {I visit Buffalo,_NY}. These facts virtually reside in per-crowd-member personal knowledge bases (reflecting the knowledge of each crowd member), and are queried by posing questions to the relevant crowd members.

Preprint

2.2 Query Language

The query language to which NL questions are translated, should naturally match the knowledge representation. We consider here the OASSIS-QL query language, which extends SPARQL, the RDF query language, with crowd mining capabilities (see full details in [2]). We stress however that the generic approach developed in the paper is applicable to other languages as well.

To briefly explain the syntax of OASSIS-QL, we use the example question from the Introduction, "What are the most interesting places near Forest Hotel, Buffalo, we should visit in the fall?". The corresponding OASSIS-QL query $\mathcal{Q}$ is given in Figure 2. Intuitively, the output of $\mathcal{Q}$ would list ontology elements that adhere to the conditions in the question, i.e., names of *places*, that are geographically *nearby Forest Hotel*, that crowd members frequently *visit in the fall*, and among those, are considered by the crowd to be the *most interesting* ones, e.g., Delaware Park or the Buffalo Zoo.

The SELECT clause (line 1) specifies the projection, i.e., which variables bindings would be returned. E.g., $\mathcal{Q}$ returns bindings of $x to places that fulfill the query conditions.

The WHERE clause (lines 2-4) defines a SPARQL-like selection query on the ontology Ψ, i.e., general knowledge. The result of the selection consists of all the variable bindings such that the resulting fact-set is contained in Ψ.

The SATISFYING clause (lines 5-12), defines the data patterns (fact-sets) the crowd is asked about, i.e., individual knowledge. Each such pattern appears in a separate sub-clause (lines 6 and 10-11), where [] stands for

an unrestricted variable, or "anything". A binding to the pattern variables is returned only if it is considered *significant*, intuitively meaning that a sufficient amount of crowd members sufficiently agree to the statement that the binding represents. In `OASSIS-QL`, significance can be defined in two ways: (i) the patterns with the k-highest(lowest) support scores, using `ORDER` and `LIMIT` (lines 7-8); and (ii) the patterns whose support pass a minimal `THRESHOLD` (line 12). The former option allows capturing, e.g., the 5 places with highest "interestingness" score according to the crowd, and the latter allows defining, e.g., which places have a high enough score to be worthy of a visit in the fall.

Note that in principle, the `SATISFYING` clause could be evaluated by means other than the crowd, e.g., over a knowledge base of individual data harvested from blogs. In this case, the translation of this clause can be performed similarly to previous NL-to-query translation tools. However, we are interested in the more challenging case, *when individual knowledge is not recorded*, which is common in crowd mining settings [2]. We address this in Section 3.2.

Preprint

2.3 NL Processing Tools

Our goal in this work is not to invent yet another NL parser from scratch, thus we instrument in our translation framework state-of-the-art NL tools, where possible. We next describe these tools and the representation they generate for NL text.

Dependency Parser This tool parses a given text into a standard structure called a *dependency graph* [21]. This structure is a directed graph (typically, a tree) with labels on the edges. It exposes different types of *semantic* dependencies between the terms of a sentence. Denote by $v(t)$, $v(t')$ the vertices corresponding to the textual terms t, t'. In an edge $(v(t), v(t'))$ we say that $v(t)$ is the *governor* and $v(t')$ the *dependent* term. The function $\mathrm{dep}(v(t), v(t'))$ returns the edge label which states the type of dependency. We would not elaborate here on the relevant linguistic theory; yet it is important to observe that within this structure, semantically related terms are adjacent.

Example 2.2 *Figure 3 depicts the dependency graph generated by Stanford Parser [21] for the example sentence from Figure 1 (ignore, for now, the highlighted parts of the graph). The edge* amod *stands for an adjectival modifier – in this case, "interesting" is an adjective which modifies, or describes "places". The edge connected to "we" is labeled* nsubj*, which stands for the speakers being the grammatical subject in the visiting event expressed by the verb "visit" [21].*

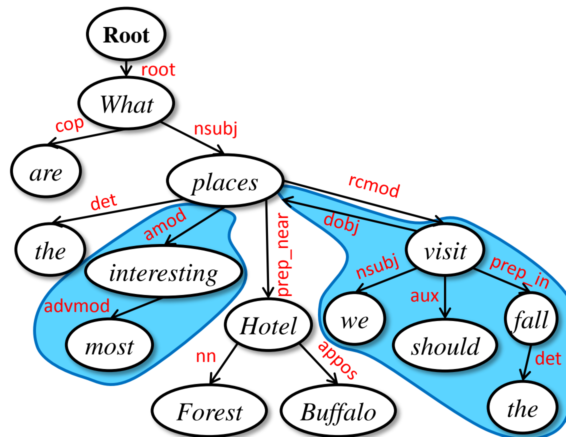

Figure 3: Example dependency graph

Preprint

What	*are*	*the*	*most*	*interesting*	*places*	*near*	*Forest*
WP	VBP	DT	RBS	JJ	NNS	IN	NNP
Hotel,	*Buffalo,*	*we*	*should*	*visit*	*in*	*the*	*fall?*
NNP	NNP	PRP	MD	VB	IN	DT	NN

WP=WH-pronoun, VBP=verb, DT=determiner, RBS=superlative, JJ=adjective, NNS=plural noun, IN=preposition, NNP=proper noun, PRP=pronoun, MD=modal auxiliary

Figure 4: An example sentence with POS tagging.

POS tagger A Part-of-Speech (POS) tagger assigns every meaningful term in a given text with its linguistic category. This category depends on the grammatical role of the term in the particular context. E.g., in "*They visit Buffalo during their annual visit to the US*", the first "visit" functions as a verb, while the second functions as a noun. We denote the tag of a term t by POS(t).

Example 2.3 *The Stanford Parser can also serve as a POS tagger. Its tags for the running example sentence are displayed in Figure 4. For example,* POS(*"interesting"*) = `JJ` *standing for an adjective, and* POS(*"Buffalo"*) = `NNP` *which denotes a proper noun (a name of a specific entity).*

Parsing quality NL parsing is the topic of extensive ongoing research in the NL processing community, but is yet far from being fully achieved, due to challenges inherent to NL text such as ambiguity, vagueness, errors and sensitivity to the context. Moreover, the quality of parsing depends on factors such as the parsing method, the training corpus, etc. For instance, in [8], a comparative study of dependency parsing studied the tradeoff between parser accuracy and efficiency of different methods, and reported precision results ranging from 48% to 91% for the same training and test sentences.[4]

We stress that our goal in this work is not to improve the existing NL parsing methods. Instead, the modular framework that presented in Section 3 allows both leveraging technological advances in NL parsing, and adjusting the NL tools to specific applications, by replacing or re-configuring the NL parsing module. The output of this module is used as-is by our translation framework and thus correct NL parsing is necessary for a successful translation by our system. In our experimental study we analyze the effect of NL parsing, and show that the obtained translation is of high quality (see Section 6).

2.4 General Query Generator

It is left to perform the translation from the NL representation to the knowledge representation. As mentioned in the Introduction, various tools enable translating questions *about general knowledge* to queries (e.g., [10, 19, 18, 35]). We refer to such tools as *General Query Generators* (GQGs, for short). If there exists a GQG suitable for the chosen knowledge representation, it makes sense to reuse it *for translating the general parts* of a question about mixed individual and general knowledge. We detail, in the sequel, how GQGs are embedded in our translation framework. For now, we show an example SPARQL query, that may be generated, e.g., by the GQG of [10, 35].

[4]Note that these results only refer to the % of the dependency edges, and the ratio of fully correctly parsed sentences is therefore lower.

Preprint

```
1 SELECT DISTINCT $x
2 WHERE
3     {$x instanceOf Place.
4      $x near       Forest_Hotel,_Buffalo,_NY}
```

Figure 5: Example SPARQL Query

Example 2.4 *Consider a variation of the running example question without the individual parts: "What places are near Forest Hotel, Buffalo?" After undergoing parsing, the question can be translated by a GQG to the SPARQL query in Figure 5.*[5] *Observe that the* `WHERE` *clause of this query and of the* `OASSIS-QL` *query in Figure 2 are identical.*

To generate queries that are evaluated over general knowledge bases, GQGs typically try to match every part of the NL request to the knowledge base, or otherwise ignore it [10, 35]. On the one hand, this approach allows to overcome some NL parsing ambiguities, since one can eliminate interpretations that do not match the ontology, or choose the interpretations that fit the ontology the best [10, 35]. On the other hand, the dependency of GQGs in a recorded knowledge base renders them useless for translating individual knowledge, which we assume to be independent of a knowledge base.

GQG quality The performance and quality of a GQG depend on many factors. These include the used NL parsing tools, the choice of knowledge base and algorithm, and user interaction capabilities. For instance, the QALD-1 challenge results show translation precision ranging from 50% to 80% [28]. We do not aim here to improve on existing GQGs, but rather to embed a GQG as a module in our framework, allowing users to plug in and reconfigure any GQG according to their needs. In our experimental study, we analyze the effect of our specific choice of GQG on the translation quality (see Section 6).

3 Translation Framework

Having established the technical background, we next describe the modules of our translation framework, according to its logical process illustrated in Figure 1. The modularity of the framework allows to easily replace each of these modules, while keeping the defined interfaces between them. This may be done, e.g., for replacing the NL parsing tools as improved ones become available. Additionally, while we explain (and implement) here a translation from NL to `OASSIS-QL`, if other crowd mining query languages are developed

[5]The query in Figure 5 contains a slightly simplified SPARQL syntax, for presentation purposes.

Preprint

(e.g., relational or XML-based) our modular framework would cater for these languages by adapting the modules that depend on the choice of language.

3.1 Individual Expressions (IXs)

Recall that our framework uses an IX Detector to distinguish individual query parts from general ones, a task which is not accounted for by existing tools. For now, we will precise the function of and IX Detector, by formalizing the notion of an IX. Later, in Section 4, we will elaborate on our new IX detection technique.

Definition 3.1 (Graph substructure) *Given a directed graph $G = (V, E)$, a* connected graph substructure $X = (V', E')$ *is such that $V' \subseteq V$, $E' \subseteq E$ and V', E' are* connected*, i.e., the 3 following criteria hold*

a. $\forall (u, v), (v, w) \in E'$ it holds that $v \in V'$

b. $\forall v \in V', (u, u') \in E'$ there exists an (undirected) path on the edges of E' from v to u or u'.

c. Between every $(u, u'), (v, v') \in E'$ there exists an (undirected) path on the edges of E'.

IXs are *connected substructures of the dependency graph with individual meaning*. Since a dependency graph shows semantic relations between sentence elements, it should (ideally) connect all the elements of a semantically coherent IX, *regardless of the particular choice of parser*. By the definition above, some edges in E' may not have both of their endpoints in V'. Intuitively, the elements corresponding to such vertices should be matched to the ontology – and are thus excluded from the IX substructure. We consider here only IXs that are *maximal* (not contained within a larger IX) and non-overlapping.

Preprint

Example 3.2 *Consider again the dependency graph in Figure 3. The highlighted parts denote connected substructures which are also IXs. Consider the IX containing the term "interesting", which denotes an opinion about the "interestingness" of places. If we remove, e.g., the* advmod *edge or add the vertex labeled "Buffalo", this IX will no longer be connected. "places" is not included in the graph since names of places are general knowledge that should be fetched from the ontology. The other IX, of the term "visit", corresponds to a request for recommendation of where to visit in the fall.*

3.2 Query Parts Creation

After executing the IX Detector, we have at hand the dependency graph, and a set $\mathcal{X}$ containing all the IXs within it. The framework now processes these IXs and the rest of the query separately to form the building blocks of the query – in the case of `OASSIS-QL`, the SPARQL-like triples.

	Pattern	Mapping
Adjectives	`{$x amod $y}`	`{tran($x) hasLabel "tran($y)"}`
Events	`{$x nsubj $y.` `$x dobj $z}`	`{tran($y) tran($x) tran($z)}`
Adjuncts	`{$x prep_in $y}` (Similarly for `prep_on`, `prep_near`,...)	`{[] in tran($y)}`

Table 1: Sample NL to `OASSIS-QL` triple mappings.

General query parts We obtain the query triples corresponding to general information needs, which are evaluated against the ontology, from the output of the GQG. More precisely, the triples are extracted from the `WHERE` clause of the generated SPARQL query, and used in the `WHERE` clause of the constructed `OASSIS-QL` query. The one difficulty in this step is in determining the *input* of the GQG. It is desirable to feed into the GQG only the non-individual parts of the NL request, e.g., the dependency graph minus the detected IXs, and the matching POS tags. However, the relevant parts of the graph may not form a coherent, parsable NL sentence. For example, in the question "**Which dishes** should we try at **local restaurants in Buffalo?**", the two general parts are highlighted in bold. If we remove the individual part between them, the general parts would be semantically detached, both in the text and in the dependency graph. For now, we will assume that the GQG can handle such a decomposed input. Later, in Section 5.1, we remove this assumption and explain our solution.

Individual query parts Can the GQG be used for creating the individual query parts as well? The answer is negative, since we assume individual knowledge cannot be aligned to a knowledge base, a method that GQGs necessarily rely on. However, the individual triples will eventually be used to generate questions to the crowd. Thus, while their structure still has to adhere to the `OASSIS-QL` syntax, it is *less restricted* than the structure of the general triples that must be aligned with the ontology structure. E.g., in the context of our running example the term "check out" can be used instead of "visit", and both options would be understood by crowd members.

We therefore translate IXs to `OASSIS-QL` by identifying grammatical patterns within the IX structure, and mapping them into corresponding triples of the target query language. The grammatical patterns are written as selection queries in SPARQL-like syntax, i.e., we use meta-queries to compute the output query. In order to use a SPARQL-like syntax, the IXs should first be converted from the dependency graph format into an RDF triple format. We next define this conversion, and then provide examples for what the pattern queries over the converted IXs can look like.

Preprint

```
1 6 hasLabel "interesting"
```

```
1 we visit 6
2 [] in    Fall
```

Figure 6: Triples translated from the IXs of Figure 3

Let $X = (V', E')$ be an IX. For an NL term t and a corresponding vertex $v(t) \in V'$, we define `tran`$(v(t))= t$. For an edge $(v(t), v(t')) \in E'$ such that $v(t) \notin V'$, `tran`$(v(t))= k$, where t is the k-th term in the original sentence (similarly for $v(t') \notin V'$). The placeholder k will be later replaced by the variable assigned to this term. An edge $(v(t), v(t')) \in E'$ maps to the triple $\{v(t) \;\; \mathrm{dep}(v(t), v(t')) \;\; v(t')\}$.

The main pattern queries that we have defined and their mappings into `OASSIS-QL` format are outlined in Table 1, for adjectives, events (typically verbs, with subjects and direct objects) and adjuncts (which describe the vertex that governs them, e.g., the time, the location). We explain the ideas behind the mapping via our running example.

Example 3.3 *Consider the IX substructures described in Example 3.2. For the first IX corresponding to "**most interesting** places", we only have one pattern matching the substructure, namely that of an adjective. v("places") $\notin V'$, and this vertex corresponds to the sixth term in the original sentence, hence* `tran`$(v(\text{"places"})) = 6$ *and we get the triple in the left-hand box in Figure 6. For the second IX, "**we should visit** (these places) **in the fall**", we have two matching patterns: for the event governed by "visit", we have the subject "we" and the direct object "places". This gives the first triple in the right-hand box of Figure 6. Then, we also have an edge labeled* prep_in, *which signifies an adjunct, and yields the second triple in the same box.*

Expressivity. In any query language, and specifically in `OASSIS-QL`, not every NL request can be captured. See the discussion about the expressivity of `OASSIS-QL` in [2]. In our experimental study, we tested, among others, the percentage of real user questions that were not supported by `OASSIS-QL`. This number turned out to be small in practice, and we further suggest which extensions to the language would be the most useful to improve its coverage (see Sections 6 and 8).

3.3 Query Composition

At the last step of the translation process, the individual and general query parts are composed to a well-formed query. As explained in Section 2.2,triples in the `SATISFYING` clause are composed into subclauses, in `OASSIS-QL`. We next describe how the Query Composition module forms the subclauses and obtains the final query structure.

Preprint

Forming the Subclauses Intuitively, every `SATISFYING` subclause should correspond to a *single event or property* that we wish to ask the crowd about. For instance, in the running example sentence from Figure 1, there is one individual property ("most interesting") and one event (visiting places in the fall). For simplicity, assume that each IX captures exactly one event or property, and thus its triples form a subclause. Recall that in every subclause, there is an `ORDER`, `LIMIT` or `THRESHOLD` expression, which defines which of the assignments to the subclause are eventually returned.

In simpler cases, the `ORDER`, `LIMIT` and `THRESHOLD` expressions can be directly derived from the input question. For instance, for the input "What are the *5 most interesting places* in Buffalo?" it is clear that the places should be ordered by decreasing "interestingness", and limited to the top-5 places. However, in our running example question, like in the vast majority of sentences in our experimental study (Section 6), such explicit numbers are not given, except for when using the singular form ("What is the most interesting *place*...?"). In such sentences, when an IX contains a superlative, e.g., "most", "best", it is translated to a top-k selection in the corresponding subclause, where k is a missing parameter. Otherwise, a support threshold is the missing parameter. We complete the missing parameters via interacting with the user (see Section 5.2) or by using predefined default values.

Completing the Query Given the `WHERE` clause and the `SATISFYING` subclauses that were computed in previous steps (e.g., the `WHERE` clause from Figure 5 and the subclauses from Figure 6 along with `ORDER`, `LIMIT` or `THRESHOLD`), we need to connect them together to form a meaningful query.

At this point, "individual" nouns such as "we", "you" or "crowd members" are removed from the query. The reason is that in the crowd interface, questions to crowd members are phrased in the second person: questions like "Where should *we* visit in Buffalo?" and "Where do crowd members visit in Buffalo?" will be presented as "Where do *you* visit in Buffalo?", addressing the crowd member directly. Hence, we replace nouns from the list of individual participants by `[]` (standing for "anything").

Second, we ensure the consistency of query variables. Recall that placeholders were used for missing endpoints, i.e., vertices not included in an IX (e.g., in Example 3.3 we used `6` as a placeholder for "places"). If the GQG assigned variables to these vertices during the processing of general query parts, we can now replace the placeholders by the appropriate variables (assuming that the assignment of variables to sentence terms is known). If a placeholder was not mapped to any query variable, we replace all of its occurrences by a fresh variable.

It remains to generate the `SELECT` clause. By inspecting user questions, we have observed that it is often beneficial for users to obtain more data than what they explicitly asked for. For example, for the sentence "Which

places should we visit in the neighborhoods of Buffalo?", one only asks explicitly for places (the *focus* of the question), but may also want to know the neighborhood each place resides in. Thus, one possibility is to set the `SELECT` clause to return the bindings of *all* the query variables, by default. In our implementation, we allow users to choose the (un)interesting output variables, see Section 5.2.

4 IX Detector

To complete the picture, we need to explain the implementation of the IX Detector, so far considered a black-box.

4.1 First Attempts

We first consider two options for IX detection by straightforward reuse of existing tools. However, as we show next, they do not account for all the challenges of IX detection. We then present our generic approach (Section 4.2), based on a semantic and syntactic definition of IXs.

Opinion mining The existing NL tools most related to our problem are tools for automated *opinion mining* (also called sentiment analysis), namely, the identification of opinions and subjective statements in NL text [29, 30]. By our definition of the individuality notion, every opinion expression in NL is also an individual expression. For example, opinion mining tools could identify "most interesting" and possibly also "should visit" in our running example question as a subjective opinions. Thus, one may attempt to reuse such opinion-detection tools as-is for IX detection. However, these tools *do not account for identifying information needs involving the habits of people.* For example, the question "Which places do you visit in Buffalo?" refers to an individual habit, but contains no opinion.

Ontology matching Another straightforward approach for IX detection is by ontology matching, i.e., declaring every expression that cannot be matched to the ontology as individual. However, this approach fails in the following cases, which are commonly encountered in practice.

- *False negatives:* general-purpose ontologies may contain (parts of) IXs in their concept taxonomy (e.g., emotion-related concepts), leading to their mistaken identification as general.
- *False positives:* due to an incomplete ontology, an expression that cannot be matched is wrongly identified as an IX.

Note that general knowledge that cannot be matched to the ontology should be treated differently than individual knowledge. E.g., it may be completed via web search or information extraction. One may also choose

Preprint

to complete it with the help of the crowd, but the phrasing of questions and the analysis of answers would still be different from those of individual knowledge, as exemplified next.

Example 4.1 *Consider the two questions "For which team does Tim Duncan play?" and the question "For which team* should *Tim Duncan play?". They only differ by a single term, yet the former is a general question (and has one true answer at a given time) while the latter asks for an individual opinion. If the ontology contains the answer to the first question, a naïve IX Detector will incorrectly identify the same facts as the answer to the second question, and miss the IX in the question (false negative).*

Alternatively, suppose that the answer to both questions is not in the ontology – in this case the former question would be wrongly identified as an IX (false positive). The problem in this case is ***not*** *that the crowd would be asked where Tim Duncan plays; but that the* ***analysis of crowd answers should be quite different****. E.g., for the former question one should ask crowd members with high expertise and gain confidence in the true answer; for the latter question one should account for diversified opinions.*

To overcome the flaws of the two basic approaches described above, our IX detection technique, described next, (i) accounts for all types of IXs including, but not restricted to opinions; and (ii) employs means of directly detecting IX within a sentence, without relying on a knowledge base. We show, in our experimental study in Section 6, that this approach indeed outperforms the two baseline approaches by a large margin.

4.2 Pattern-based IX Detection

In the NL processing literature, there are two main approaches for detecting certain types of knowledge (e.g., opinions, negation, etc.): first, using training sentences and employing machine learning techniques; and second, using predefined structural patterns and vocabularies. The approach we take here is pattern-based, where the patterns partly use vocabularies of opinion mining, which can be constructed by machine learning techniques. While our approach is simple and transparent (to, e.g., a human administrator), it allows flexibility with respect to pattern creation (manually or via learning techniques) and also performs well in our experiments, identifying 100% of the IXs (with a few false positives, see Section 6).

Based on a preliminary examination of real-life NL questions (taken from Yahoo! Answers [34]), we identify the following typical sources of individuality in NL IXs:

- **Lexical individuality.** A term (or phrase) in the IX has a lexical meaning which conveys individuality. For example, the term "awesome" conveys individuality, as in "Buffalo is an awesome city".

Preprint

In contrast, the term "northern" in "Buffalo is a northern city" is not individual (even though there may be different ways of defining "northern").

- **Participant individuality.** The IX refers to an event that includes a participant which indicates individuality. For example, consider the participant "we" in "we visited Buffalo". In contrast, "Tim Duncan" is not an individual participant (the meaning is the same regardless of the speaker).

- **Syntactic individuality.** A syntactic structure may indicate individuality. For example, consider the sentence "Tim Duncan should play for the NY Knicks", where the verb auxiliary "should" indicates individuality (the sentence conveys an opinion of the speaker).

For each type of individuality source, we have manually constructed a set of *detection patterns*, in terms of the dependency graph, POS tags and relevant vocabulary. We express these patterns as SPARQL-like queries over the vertex-edge-vertex representation of dependency graph, similarly to Section 3.2 for converting IX pieces to the query triples. We exemplify the syntax of IX detection queries next, and provide the full set of patterns in the Appendix.

Example 4.2 *The detection pattern below identifies a particular case of participant individuality, where a verb subject is individual.*

```
1 $x nsubj $y
2 filter($y in V_participant &&
3        (POS($x) = "VBP" ||
4         POS($x) = "VB"))
```

Explanation: line 1 selects a vertex-edge-vertex triple of a verb (assigned to `$x`*) and its grammatical subject (assigned to* `$y`*).* `$y` *must be an individual participant, i.e., belong to the vocabulary* `V_participant`*.* `$x` *should be a verb, i.e., marked by one of the POS tags that denote a verb (we show only two of the possible tags in the pattern above). Given the example dependency graph and POS tags, and assuming "we" is in the vocabulary, this pattern would capture the vertices corresponding to "we", "visit" and the* nsubj *edge between them.*

To obtain the full IX, the part of the dependency graph matching the pattern must be *extended to a meaningful expression.* For that, we use additional selection patterns that, e.g., add to verbs all their subjects and objects; add adjuncts to captured terms; etc.

Preprint

5 Implementation

Next, we describe the implementation of our translation framework in a novel prototype system, NL$_2$CM.

5.1 The NL$_2$CM System

NL$_2$CM is implemented in Java 7. Figure 7 depicts its modular design, where our new developments are painted black. The UI of NL$_2$CM is web-based, and implemented in PHP 5.3 and jQuery 1.x. Through this UI users can write NL questions, interactively translate them into a formal query language, and finally submit the resulting queries for execution by OASSIS. The entered NL text is first parsed using a black-box module, where we require that the output dependency graph and POS tags would be aligned (which is the case when using a combined parsing tool such as Stanford Parser). These outputs are passed on to subsequent modules.

One of the subsequent modules is our newly developed IX Detector, which is split in Figure 7 into two components: the *IXFinder* uses vocabularies and a set of predefined patterns in order to find IXs within the dependency graph, as described in Section 4.2. We use a dedicated vocabulary for each type of IX pattern: for *lexical* individuality, there are many publicly available vocabularies that contain expressions of opinions or subjectivity. We use the Opinion Lexicon [6]. For the other types, the required vocabularies are very small and can be easily composed.[7] The second module of the IX Detector, the *IXCreator*, is responsible for completing the subgraphs representing the IXs. For example, if some verb is found to have an individual subject, this component further retrieves other parts belonging to the same semantic unit, e.g., the verb's objects.

The GQG is the black-box responsible for translating the general query parts into SPARQL triples. Since many GQGs only accept full parsed sentences as input, NL$_2$CM feeds the module with the dependency graph and POS tags of the original user request, *including the IXs*. Some of the IXs may be wrongly matched by the GQG to the ontology and translated into general query triples. To overcome this problem, the Query Composition module later deletes generated SPARQL triples that correspond to terms within IXs.

The Individual Triple Creation module receives the IXs, and converts them, in this case, into OASSIS-QL triples. These triples are then composed

[6]http://www.cs.uic.edu/ liub/FBS/sentiment-analysis.html

[7]A list of modal auxiliaries for grammatical individuality can be found, e.g., in Wikipedia (http://en.wikipedia.org/wiki/Modal_verb#English), and for the list of individual participants we used pronouns (see, e.g., http://en.wikipedia.org/wiki/English_personal_pronouns), and a few words that refer specifically to the crowd ("the crowd", "people", "everybody",...). See the full list in the Appendix.

Preprint

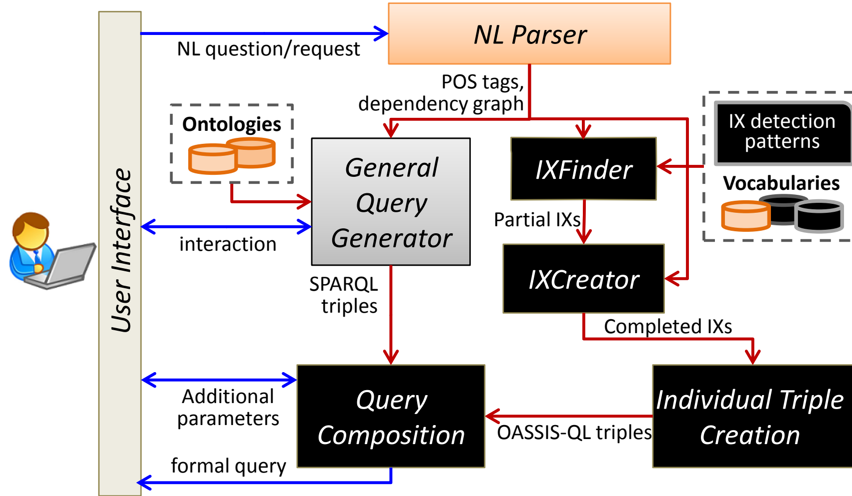

Figure 7: System Design

with the generated SPARQL triples into a well-formed query by the Query Composition module, as described in Section 3.3.

5.2 Resolving Missing Query Parameters

As described in Section 3.3, some parameters necessary for the translation may not be explicitly specified by the NL question. One possibility, in such cases, is using default values. However, when possible, a better solution is asking the users targeted questions about the missing parameters in their queries. Of course, these questions must be phrased in manner suitable for a naïve user, i.e., in terms of the NL question and desired answers rather than the constructed query. In our implementation of `NL2CM`, we thus enhance the Query Composition module with the following optional parameter completion capabilities.

1. *Completing missing* `LIMIT`. To ask for a missing value, the UI presents the NL question, highlighting the terms corresponding to the relevant subclause. The user can then adjust the limit by moving a cursor. For instance, in the running example, the number of most interesting places to return is not specified. The system would highlight the term “places” and ask the user to specify how many of these she wants to obtain in the output.
2. *Completing missing* `THRESHOLD` *values.* The treatment of such missing values is similar to the previous case, but with a different template question posed to the user and different possible answers.For example, for “should visit in the fall” in the running example, the threshold of what is considered recommended is not explicitly specified. The user would be asked to choose the “level of popularity” for

the desired answers, between "Above zero" (which would return any place recommended by anyone) and "Very high" (requiring a consensus among crowd members). This answer would be translated to a numeric threshold.

3. *Deciding which variables to project out.* Instead of returning all the variables by default, as described in Section 3.3, $\texttt{NL}_2\texttt{CM}$ can ask the users to select what they would like to see in the output, among the terms of the sentence. It further makes use of the capability of the FREyA GQG [10] (which we also use in our Experiments in Section 6) to identify the *focus* of a question. The focus is intuitively the grammatical placeholder in an NL question for the expected answer. The question is presented with check boxes next to terms that correspond to query variables, where initially only the focus term is selected. For instance, in "Which places should we visit in the neighborhoods of Buffalo?", only the term "places", which is the focus of the question, will be initially selected. The user can decide to further select "neighborhoods", to receive names of neighborhoods along with the names of places in the output.

We also test the option of asking users to verify detected IXs, by highlighting the corresponding NL question parts, and asking the user whether the crowd should be asked about each part (Section 6.2). In practice, this option is proved to be ineffective, as our IX detector makes very few errors, and user interaction mostly introduces new errors.

6 Experimental Study

We have conducted an extensive experimental study, in order to evaluate the translation of $\texttt{NL}_2\texttt{CM}$. In order to test the variability of the system when no prior knowledge is available about the input question, we have plugged into $\texttt{NL}_2\texttt{CM}$ general-purpose black-boxes. For NL parsing, we have compared the performance of three state-of-the-art parsers: Stanford Parser [21], MaltParser [25] and TurboParser [22]. The latter two have parsed correctly 3% and 34% less sentences than Stanford Parser, respectively. Hence, in the sequel we show only the experimental results for Stanford Parser 3.3.0, trained over the Penn Treebank Corpus using the EnglishPCFG model [21]. For a GQG $\texttt{NL}_2\texttt{CM}$ employs FREyA [10]. Among state-of-the-art GQGs that translate NL to SPARQL, we chose FREyA since it allows easily configuring the ontology, has published code, and provides user interaction options to improve the matching of sentence terms to ontology entities [10]. We have configured FREyA with the general ontology DBPedia, along with geographical ontology LinkedGeoData.

Our experimental study consists of two parts. In the **real user experiment**, users were asked to input their questions to the system through

$\mathtt{NL_2CM}$'s UI, and we examined the generated queries. This experiment investigates a real usage scenario of $\mathtt{NL_2CM}$, and allows evaluating the system's output quality. We then examine the translation of questions taken from a repository of the popular question-and-answer platform Yahoo! Answers [34] in the **Yahoo! Answers experiment**. This repository contains questions on a large variety of topics, posted by a large body of users, and is therefore useful for testing the versatility of $\mathtt{NL_2CM}$.

As a baseline, we compare the results of $\mathtt{NL_2CM}$ with the two basic alternatives described in Section 4.1:

1. *Opinion mining*. By analyzing the distribution of detected individuality types, we can compute the portion of questions that can be detected by this baseline alternative (see Section 6.4).
2. *Ontology matching*. We have tested a few variations and show here only the results for the best one, namely exact matching of terms to the ontology – terms that are not matched are considered parts of an IX. Matching sentence parts has resulted in a higher rate of false positives (general sentence parts that cannot be matched), and loose matching, taking into account synonyms and similar strings, resulted in a higher rate of false negatives (individual sentence parts that are wrongly matched to the ontology). As the results were similar for the two experiments, we present the baseline results only for Yahoo! Answers (see Section 6.3).

6.1 Methodology

Two approaches are possible for evaluating the translation quality: executing the query and asking users whether they are satisfied with the query results, and manually examining the translated query. We have taken the latter, stricter approach, because the former depends on the quality of the query evaluation system (`OASSIS` in this case) which is orthogonal to the translation quality, and moreover, users may not be able to identify irrelevant/missing query results caused by erroneous translation.

We have manually analyzed and classified the questions according to their properties and to the translation results. The classifications are as follows.

- **Non-individual.** Questions that do not contain an individual part. In the real user experiment we have further divided these questions into "good" and "bad" to test which ones were correctly identified as non-individual.
- **Descriptive questions.** Questions with individual parts, that require a complex, non-structured answer and thus does not fit well with structured query languages such as `OASSIS-QL`. These questions usually start with "How to...", "Why...", "What is the purpose of...", and so on (see Example 6.1 below).

Preprint

- **Good.** The question was translated correctly as-is.
- **Good (minor edit).** As previously mentioned, black-box parsers are not perfect and parse incorrectly some sentences. In order to test the ability of our new components to handle various real-life questions regardless of incorrect parsing, we have checked in which questions the incorrect parsing was the cause of incorrect translation. When possible, we manually made minor, syntactic edits to such questions, which were sufficient to make the questions translate correctly, and preserved the original sentence meaning. These edits mostly include changes of word order, connective words or punctuation that affect the identification of dependent sentence parts (See Example 6.1 below). Parsing problems of this sort should ideally be automatically handled, e.g., by interacting with the user. Adding such capabilities to the parser, however, are out of the scope of this paper.
- **Bad `OASSIS-QL`.** The sentence meaning could not be captured in `OASSIS-QL`.
- **Bad black-box.** The sentence translation failed due to errors of Stanford Parser or FREyA, which could not have been overcome by minor edits.
- **Bad $\texttt{NL}_2\texttt{CM}$.** The sentence translation failed in one of $\texttt{NL}_2\texttt{CM}$'s newly-developed modules.

The goal we have set for our system is to translate *every question that has individual parts except for descriptive questions.* We estimate the quality of our newly developed modules independently (as much as possible) from the black-boxes used in $\texttt{NL}_2\texttt{CM}$.

To provide an intuition about the practical meaning of the annotations, we show below example questions for each annotation. For one question we further show and explain its translation to `OASSIS-QL` by $\texttt{NL}_2\texttt{CM}$.

Example 6.1 *Consider the following user questions/requests, collected from both of our experiments.*

Q.1: *(non-individual)* "Find a radio station that plays classical and jazz music." *This question is correctly translated as non-individual, which means that the* `SATISFYING` *clause of the resulting* `OASSIS-QL` *query is empty.*

Q.2: *(descriptive question)* "What's the best way to create a bootable windows/dos CD?" *The answer to this question requires a potentially complex explanation, thus, we consider it a descriptive question which is not supported by the query language.*

Q.3: *(good)* "Where should I eat in Hong Kong if I prefer European food?" *This example is correctly translated by* $\texttt{NL}_2\texttt{CM}$ *as-is. See its translation to*`OASSIS-QL` *below.*

Q.4: *(good after minor edit)* "What is the best **and** most powerful portable GPS unit?" *In the original form of the question there was a comma*

Preprint

```
1  SELECT VARIABLES
2  WHERE
3      {$i0 instanceOf* Location.
4       $i4 instanceOf* HongKong.
5       $c8 subclassOf* Food.
6       $c8 hasLabel "European"}
7  SATISFYING
8      {[]  eat [].
9       []  at $i0.
10      []  in $i4.
11      []  prefer $c8.}
12     WITH SUPPORT THRESHOLD = 0.5
```

Figure 8: The translation of question 3 (Example 6.1) to `OASSIS-QL`

instead of "and", and this caused Stanford Parser to wrongly decide that "best" describes "powerful" rather than "unit". Our minor edit fixed this.

Q.5: (good after minor edit) Where **next to Central Park** should I drink coffee?" *(original: "Where should I drink coffee next to Central Park?") In the original form of the sentence Stanford Parser does not identify the phrase "next to", and its relation to "Where". Changing the word order fixes these parsing errors.*

Q.6: (bad OASSIS-QL) "What words should I learn before I go to China?" *The temporal precedence "before" cannot be expressed by OASSIS-QL.*

Q.7: (bad black-box) "Any suggestions on fun, non-wine or golf related things to do in Carmel?" *Stanford Parser does not manage to identify all the semantic relations in this sentence, which cannot be fixed by minor edits.*

Q.8: (bad $\texttt{NL}_2\texttt{CM}$*)* "What voltage adapter should I use for my 5.4v pda?" *The current implementation of* $\texttt{NL}_2\texttt{CM}$ *does not support certain adjuncts of elements that are not queried from the ontology. In this case, the "5.4v" property of the PDA is lost in the translated query.*

The example query in Figure 8 is the translation of Q.3 to OASSIS-QL.[8] *In the* `WHERE` *clause, the variable* `$i0` *is assigned to an instance of location (the answer to "where");* `$i4` *is assigned to Hong Kong*[9]*; and* `$c8` *is assigned to European food. The* `SATISFYING` *clause consists of only one subclause, with two verbs – eating and "preferring", and with two locations descriptions – the place to eat at, and Hong Kong. When evaluated, the query would return all combinations of locations and types of European food (and the entity for "Hong Kong"), such that people jointly eat at the location, are in*

[8]For readability, we have simplified the code output by FREyA, but the `SATISFYING` clause has not been modified.

[9]`instanceOf*` denotes a path of ≥ 0 `instanceOf` relations. In this case, `HongKong` is the only value assignment to `$i4` (via a 0-length path from itself).

Preprint

Hong Kong, and prefer the type of food. This indeed captures the intention of the original question.

Note that the second dataset (Yahoo! Answers) is analyzed a posteriori, unlike the real user experiment. We have thus carefully chosen classification criteria that are not affected by user interaction, i.e., by missing values that we can ask the user to complete (as described in Section 5.2). For the real-users experiment, we separately analyze below the effect of such user interaction.

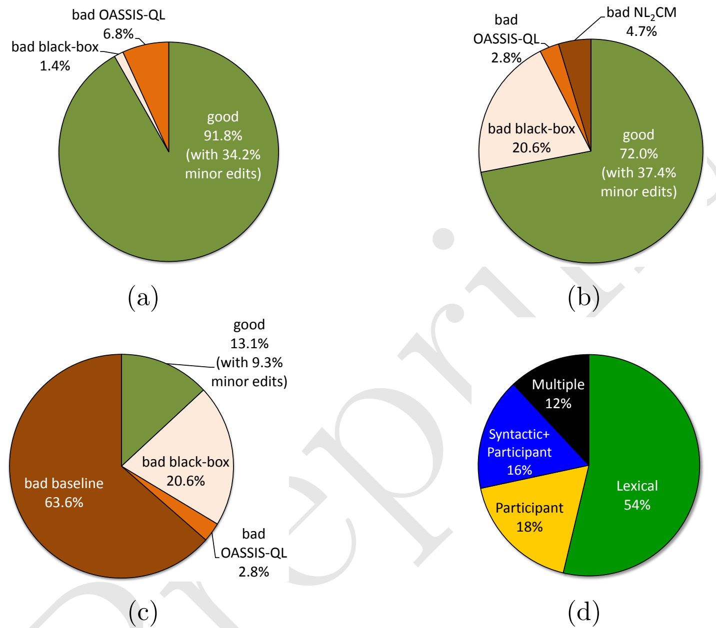

Figure 9: Experimental results. (a) the results of the user experiment, for individual questions; (b) the results of the Yahoo! Answers experiment; (c) the results of the baseline algorithm over Yahoo! answers data; (d) statistics about the type of individuality, in correctly translated queries from the user experiment.

6.2 User Experiment

In this experiment, 10 volunteer users from different backgrounds were asked to feed questions to the UI of $\texttt{NL}_2\texttt{CM}$. They were told that their questions would be evaluated with the crowd, and were asked to write questions that *explicitly ask for individual information*, i.e., the habits and opinions of people. The system usage instructions provided the users with brief, general guidelines on how to phrase questions, and the types of questions the system supports.

In total, we have collected 100 questions, out of which 73 were individual and not descriptive. The analysis results for this subset of questions

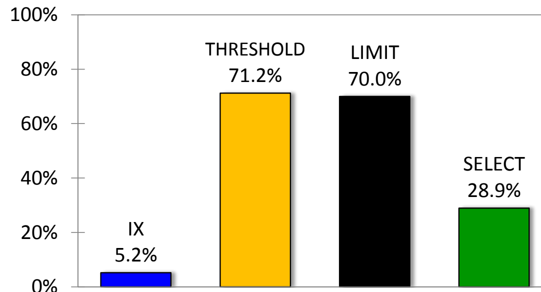

Figure 10: User interaction. % of sentences where users changed: the IXs, the threshold, the limit or the projected variables.

are displayed in Figure 9a. First, note that about 92% of these questions were correctly translated (out of which, 34% required minor manual edits), which shows that $\mathtt{NL_2CM}$ is highly capable of dealing with real user questions. Second, *100% of the sentences that did not fail due to other reasons (black-boxes, OASSIS-QL) were correctly processed by the new modules of* $\mathtt{NL_2CM}$*, and in particular, 100% of the IXs were identified.* Last, by our inspection of the questions that could not be captured by the query language semantics (annotated "bad OASSIS-QL"), it turns out that they all happened to involve *temporal semantics*, such as in Q.6 from Example 6.1. The relative abundance of questions of this sort suggests that it might be useful to add temporal constructs to the language.

As for the non-individual questions, these mostly seem to stem from an occasional confusion of the users between questions they *wanted to ask the crowd* with ones that have *explicit individual parts* (as they were guided to ask). For example, consider Q.1 from Example 6.1 which is not individual, but in practice can be answered with the help of the crowd. Including Q.1, 15 out of 21 non-individual questions were nonetheless correctly translated (to queries with an empty SATISFYING clause). Other questions were wrongly identified as individual due to parsing errors (4 questions) and $\mathtt{NL_2CM}$ false positives (2 questions).

We have also analyzed the impact of user interaction. This includes the fetching of missing values and verifying the detected IXs (as described in Section 5.2). Instructing the system to ignore identified IXs was done only in 5.2% of the sentences, and in about half of these cases, the user wrongly dismissed correct IXs. This suggests that *IX verification by the user is ineffective.* In contrast, users frequently changed the default LIMIT and THRESHOLD (70% and 71.2%, resp.). This is expected, since these values do not appear in the original questions. Finally, due to the automated identification of the *question focus* (see Section 5.2), users changed the SELECT clause variables in only 28.9% of the questions. In these questions the user had to correct the question focus identified by FREyA, or decided to add

Preprint

desired variables to the output. In summary, asking the user for missing values was proven beneficial.

6.3 Yahoo! Answers Experiment

In this experiment, we have arbitrarily chosen the first 500 questions from the Yahoo! Answers repositories [34]. Of the 500 questions, 26% asked for individual information. Note that this significant portion of real-life questions could not be handled by previous translation systems. 5% were individual descriptive questions, and the rest were fed into $\texttt{NL}_2\texttt{CM}$ to be translated (some after a small syntactic refinement, to ensure Stanford Parser analyzes them correctly).

The distribution of annotations assigned to the individual, non-descriptive questions is depicted in Figure 9b. Most notably, the translation quality was significantly lower than that of the real user experiment. This indicates that simple user instructions help avoiding many translation problems, mostly relating to NL parsing problems, that are hard to avoid by other means. Altogether, 72% of the Yahoo! Answers questions were translated correctly (out of which, 37% required minor edits), which demonstrates the ability of $\texttt{NL}_2\texttt{CM}$ to handle various questions, ranging from travel and product recommendations to technological and health-related issues. *If we consider only the new modules, 93.8% of the questions (that did not fail due to other reasons) were handled correctly.* For comparison, consider the results of the baseline algorithm that detects IXs based on matching the question terms to the ontology (Figure 9c). Only 13.2% of the questions in total were correctly parsed by this algorithm. Out of the questions that were not parsed correctly by the baseline, 95% were due to false negatives, and 15% were false positives (with overlaps). This is expected, since we use a general ontology that supports a wide variety of questions, but thus also matches many individual terms.

6.4 Queries Analysis

In addition to the qualitative evaluation, we have analyzed the properties of the translated queries in order to gain a better understanding of the system's challenges. Figure 9d displays the distribution of individuality sources within the individual questions of the user experiment (the Yahoo! experiment exhibited similar trends). In these questions, syntactic individuality always appeared along with participant individuality ("should I", "should we"), hence the chart displays the frequency of their combination. The "Multiple" slice represents sentences which contain multiple types of individuality. These results also allow us to compare our IX detector with the baseline approach based on opinion mining tools: if we use, e.g., SentiWordNet [29] to identify individual terms, 34% of the IXs are not identified,

Preprint

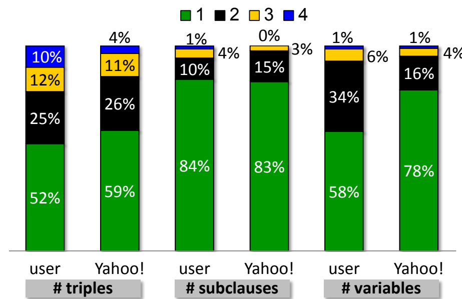

Figure 11: Statistics about correctly translated queries.

because they involve habits and recommendations rather than opinions. The results also show that each of the individuality types that we have defined is indeed frequently found in individual questions. Finally, the results reaffirm that IX Detection is a non-trivial task, since a significant portion of the questions combines multiple individuality types.

Figure 11 displays the distributions of the total number of triples, subclauses and variables in the `SATISFYING` clause, with respect to queries that were correctly translated. These numbers serve as indicators for the complexity of the resulting query (and consequently the original NL question). As the diagram illustrates, the individual part of the questions/requests is mostly very simple, yet there are some queries which have a fairly complex structure, relatively to NL sentences. Interestingly, the queries obtained via our user experiment were slightly more complex by the different metrics we checked. While we cannot know the actual cause for this difference, we conjecture that posing a question to an automated system encourages people to ask more complex questions than they would ask in a forum.

Preprint

7 Related Work

The study of *NL interfaces to databases* (NLIDBs) dates back to the late sixties of the 20th century. Early examples of such systems (e.g. [16, 23]) employed dedicated parsing the NL requests into logical queries over specific data schemas (and thus, were not portable). The logical queries served in some cases as an intermediate stage between NL and standard query languages such as SQL [4]. See [5] for a thorough review of these early approaches. Later NLIDB, e.g., [18, 24, 27, 32], employed more generic approaches for translating NL (or a fragment thereof) to SQL queries, building on machine learning techniques [32], statistical analysis [24], probabilistic models [6] and state-of-the-art NL parsers [18, 19, 27]. In particular, the more recent NaLIR and NaLIX systems [18, 19] interact with the user to refine its interpretation of the NL question and its matching to the schema

of relational databases and XML, respectively. While NL$_2$CM also employs interactive refinement, such interaction is not available in its current NL parsing module (Stanford Parser, which is used as a black-box). In this sense, the techniques of [18, 19] complement those of NL$_2$CM by serving to improve the NL parsing of the input question.

Due to an increasing interest in the Semantic Web, several systems have been recently developed for the *translation of NL requests to SPARQL queries*, e.g., [10, 35]. As mentioned, all of these systems deal only with general knowledge and do not account for individual knowledge. NL$_2$CM employs one such system (FREyA) as a black-box for translating the general query parts. In this context we also mention the recent work of [37] which studies direct evaluation of NL questions (on general information) over RDF graphs, without the intermediate translation-to-query phase. This approach is, however, not suitable to our setting, since it cannot account for the evaluation of individual query parts, for which an ontology is not available.

The use of *crowdsourcing techniques* for data procurement tasks is of great current interest. In addition to crowd mining studies, mentioned throughout this paper [1, 2], recent work studies the crowd-powered evaluation of query operators [26, 33]; data management tasks like data cleaning [9, 31] and schema matching [13, 36]; more high-level processing such as information extraction [15, 17]; and many more. NL interface is mentioned in some of these papers, but only in the context of generating questions to crowd members, in which case the approach is template-based [2, 14]. The *CrowdQ* system [12] involves crowd-powered techniques for identifying patterns in keyword search queries. It would be interesting to study whether these means can also be adapted to our setting, to obtain crowdsourced parsing of user questions that mix general and individual parts.

Some additional NL tools to note here are opinion mining tools such as Stanford Sentiment Analysis [30], and SentiWordNet [29], which are useful for identifying a fragment of individual expressions. Another tool is a plug-in for the GATE architecture [7] that allows defining templates of micro-tasks to the crowd, and may be used for crowd-powered query refinement. It would be interesting to integrate such a tool in our system, in cases when interaction with the user who submitted the question is not possible.

Preprint

8 Conclusion and Future Work

In this paper, we have studied the problem of translating NL questions that involve general and individual knowledge into formal queries, thereby making crowd mining platforms accessible to the public. We have described the novel modules of a translation framework, which detect, decompose, translate and recompose the individual query parts with the general ones. Our approach proves effective in our experimental study, for wide-ranging user questions.

Our study highlights some intriguing future research directions. First, it would be interesting to replace or wrap Stanford Parser to allow interactive parsing refinement, and assess the effect on the output query quality versus the user effort incurred. For the IX Detector, future research may study machine learning techniques for automatically suggesting detection patterns. It would also be interesting to examine whether our approach could be adapted for analyzing NL answers collected from the crowd, forums or social platforms. Finally, our experiments reveal frequent structures in real NL questions. For structures that are not yet supported, e.g., temporal relations in `OASSIS-QL`, adequate new constructs should be developed.

References

[1] Y. Amsterdamer, S. B. Davidson, A. Kukliansky, T. Milo, S. Novgorodov, and A. Somech. Managing general and individual knowledge in crowd mining applications. In *CIDR*, 2015.

[2] Y. Amsterdamer, S. B. Davidson, T. Milo, S. Novgorodov, and A. Somech. OASSIS: query driven crowd mining. In *SIGMOD*, 2014.

[3] Y. Amsterdamer, A. Kukliansky, and T. Milo. NL_2CM: A natural language interface to crowd mining. In *SIGMOD*, 2015.

[4] I. Androutsopoulos, G. Ritchie, and P. Thanisch. Masque/sql - an effcient and portable natural language query interface for relational databases. In *IEA/AIE*, 1993.

[5] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases - an introduction. *NL Eng.*, 1(1), 1995.

[6] J. Berant and P. Liang. Semantic parsing via paraphrasing. In *ACL*, 2014.

[7] K. Bontcheva, I. Roberts, L. Derczynski, and D. P. Rout. The GATE crowdsourcing plugin: Crowdsourcing annotated corpora made easy. In *EACL*, 2014.

[8] D. M. Cer, M. D. Marneffe, D. Jurafsky, and C. D. Manning. Parsing to stanford dependencies: Trade-offs between speed and accuracy. In *LREC*, 2010.

[9] A. Chalamalla, I. F. Ilyas, M. Ouzzani, and P. Papotti. Descriptive and prescriptive data cleaning. In *SIGMOD*, 2014.

[10] D. Damljanovic, M. Agatonovic, H. Cunningham, and K. Bontcheva. Improving habitability of natural language interfaces for querying ontologies with feedback and clarification dialogues. *J. Web Sem.*, 19, 2013.

Preprint

[11] Dbpedia. http://dbpedia.org/About.

[12] G. Demartini, B. Trushkowsky, T. Kraska, and M. J. Franklin. CrowdQ: Crowdsourced query understanding. In *CIDR*, 2013.

[13] J. Fan, M. Lu, B. C. Ooi, W. Tan, and M. Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *ICDE*, 2014.

[14] M. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*, 2011.

[15] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*, 2014.

[16] G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum. Developing a natural language interface to complex data. *ACM Trans. Database Syst.*, 3(2), 1978.

[17] S. K. Kondreddi, P. Triantafillou, and G. Weikum. Combining information extraction and human computing for crowdsourced knowledge acquisition. In *ICDE*, 2014.

[18] F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *PVLDB*, 8(1), 2014.

[19] Y. Li, H. Yang, and H. V. Jagadish. NaLIX: A generic natural language search environment for XML data. *ACM Trans. DB Syst.*, 32(4), 2007.

[20] Linkedgeodata. http://linkedgeodata.org/About.

[21] M. D. Marneffe, B. Maccartney, and C. D. Manning. Generating typed dependency parses from phrase structure parses. In *LREC*, 2006.

[22] A. F. T. Martins, N. A. Smith, E. P. Xing, M. A. T. Figueiredo, and P. M. Q. Aguiar. TurboParsers: Dependency parsing by approximate variational inference. In *(EMNLP*, 2010.

[23] F. Meng and W. W. Chu. The Lunar sciences natural language information system: Final report. Technical Report BBN Report No. 2378, Bolt Beranek and Newman Inc., Cambridge, 1972.

[24] F. Meng and W. W. Chu. Database query formation from natural language using semantic modeling and statistical keyword meaning disambiguation. Technical Report 990003, CD Dept., UCLA, 1999.

[25] J. Nivre, J. Hall, and J. Nilsson. MaltParser: A data-driven parser-generator for dependency parsing. In *LREC*, 2006.

Preprint

[26] A. G. Parameswaran, S. Boyd, H. Garcia-Molina, A. Gupta, N. Polyzotis, and J. Widom. Optimal crowd-powered rating and filtering algorithms. *PVLDB*, 7(9), 2014.

[27] A. Popescu, O. Etzioni, and H. A. Kautz. Towards a theory of natural language interfaces to databases. In *IUI*, 2003.

[28] QALD-1 open challenge. `http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/1/qald-1-challenge.pdf`.

[29] SentiWordNet. `http://sentiwordnet.isti.cnr.it/`.

[30] Stanford Sentiment Analysis tool. `http://nlp.stanford.edu/sentiment`.

[31] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The Data Tamer system. In *CIDR*, 2013.

[32] L. R. Tang and R. J. Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *EMCL*, 2001.

[33] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, 2013.

[34] Yahoo! webscope dataset ydata-yanswers-all-questions-v1_0. `http://research.yahoo.com/Academic_Relations`.

[35] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum. Deep answers for naturally asked questions on the web of data. In *WWW*, 2012.

[36] C. Zhang, Z. Zhao, L. Chen, H. V. Jagadish, and C. Cao. Crowd-Matcher: crowd-assisted schema matching. In *SIGMOD*, 2014.

[37] L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao. Natural language question answering over RDF: a graph data driven approach. In *SIGMOD*, 2014.

Preprint

A IX Patterns

To complete the picture about the IX detection patterns, presented in Section 4.2, we next provide the full list of patterns according to the type of individuality they detect. For brevity, we use a few "macros" within these patterns.

A.1 Lexical Individuality

```
[] amod $x.
filter($x in V_lexical || $x starts_with "must-")
```

Lexically individual adjective: line 1 selects a vertex-edge-vertex triple of a term and the adjective that describes it. The adjective term is assigned to `$x`; and `[]` signifies we are not interested in retrieving the governing vertex, which may be general (otherwise it will be retrieved by another pattern). The `filter` clause in line 2 serves to define constraints over the matched triples. In this case, it returns only triples where the adjective is found in a vocabulary of lexical individuality terms, `V_lexical`, or starts with "must-" (e.g., "must-see").

For instance, given the dependency graph from Figure 3, and assuming "interesting"∈`V_lexical`, this pattern would select the edge ("places" "interesting") and the vertex corresponding to "interesting" in Figure 3. "places" would be matched to `[]`, but would not be selected.

Preprint

```
$x $p $y.
filter($y in V_lexical &&
       $p in {"acomp", "advmod"})
```

Lexically individual adverb/adjectival complement: similar to the previous case, but when the adjective/adverb describes a verb (e.g., "look nice"). In this case, the verb will serve as a part of the IX.

```
$x [] []
filter(POS($x) in VRB_TAGS &&
       $x in V_lexical)
```

Lexically individual verb: this pattern captures verbs like "love" and "prefer". The macro for verb POS tags is defined as follows.

```
VRB_TAGS = {"VB" , "VBD" , "VBG" , "VBN", "VBP", "VBZ"}
```

A.2 Participant Individuality

```
$x aux to.
filter(!exists($y nsubj $x) &&
       !exists($y xsubj $x))
```

Infinitive verbs: infinitive verbs are typically used with a missing subject that refers to an individual participant ("Find a place in Buffalo **to park** a trailer").

Since the definition of the individual participant is independent from the manner in which it relates to the verb/adjective (i.e., whether it is a subject, object, etc.), we use the following recursive macro for this definition. It is sufficient to consider recursion of a very shallow depth (one or two recursive calls cover the vast majority of practical cases), so this macro can be rewritten as a non-recursive one.

```
1 INDV($x) :=
2  ($x in V_participant ||
3   exists($x $p $y.
4          filter(INDV($y)) &&
5                 ($p starts_with "prep_" ||
6                  $p in {"appos","nn","poss","vmod","infmod"}
7     ))))
```

Individual participant macro. At the basic case (line 2), a participant is considered individual if it appears in `V_individual`. This small vocabulary consists of the terms: "I", "my", "me", "you", "your", "we", "our", "us", "someone", "somebody", "anyone", "anybody", "everyone", "everybody", "people", "one", "crowd". Otherwise, a participant is considered individual if it described by another noun which is individual, e.g., "window" is not individual by itself, but e.g., "our window", "my brother's car window" and "window of your car" are considered individual.

```
1 $x $p $y.
2 filter(INDIV($y) &&
3         ($p starts_with "prep_" ||
4          $p in {"agent","dobj","iobj","nsubj","nsubjpass",
5                 "xsubj"}) &&
6        !(exists($z aux $x)))
```

Individual participant pattern. The pattern above captures individual participants in all of the roles they may play with respect to a verb/adjective: an adjunct (line 3) or a subject or object (line 4-5). Line 6 ensures that `$x` is not a verb auxiliary.

A.3 Syntactic Individuality

```
1 $x aux $y.
2 filter($y in V_syntactic)
```

Syntactically individual auxiliaries. The pattern above captures verb auxiliaries with modal meaning, such as "should" and "must", which cause their governing verb to be a part of an IX.

Preprint

```
$x xcomp $y.
filter($x in V_syntactic)
```

Syntactically individual verbs. A restricted group of terms behaves similarly to auxiliaries (e.g., “need to”, “have to”), and are captured by the pattern above.

In total, `V_syntactic` consists of: “should”, “must”, “need”, “needs”, “needed”, “have”, “has”, “had”, “ought”, “shall”.

Preprint