Finding a Lattice of Needles in a Haystack: Forming a Query from a Set of Items of Interest

Boris A.Galitsky

Knowledge-Trail Inc San Jose CA USA
bgalitsky@hotmail.com

Abstract. We introduce a new type of query, a lattice query, which is intended to assist a user of a search engine in query formulation. An associated methodology of search is proposed, where instead of submitting an exact query, the user provides a set of text samples which are the basis of generalization. The lattice query system automatically forms a search query from this generalization and verifies the relevancy of search results to the provided set of samples. Lattice queries are also designed to assist in building templates for information extraction tasks: instead of specifying certain keywords or linguistic patterns, a developer can give a list of samples and leave generalization task to the system. Lattice queries are formed from individual sentences and from paragraphs of text as well. An open source contribution of lattice query search component as a part of OpenNLP is described.

Keywords: search engine query, generalizing from samples, interactive search

1 Introduction

Today, the significant portion of information is obtained via search engines. Horizontal web search engines and well as specialized vertical search engines such as product search and health recommendations are the essential sources of information in the respective domains. Modern open source big data search and exploration systems like Solr and Elasticsearch are broadly used for access and analysis of big data. However, intelligence features such as search relevance and adequate analysis, retrieval and exploration of large quantities of natural language texts are still lacking. It is still had to find information in a horizontal or vertical domain unless precise search keywords are known to the user [1,2,3].

Frequently, novice users of search engines experience difficulties formulating their queries, especially when these queries are long. It is often hard for user who is new to a domain to pick proper keywords. Even for advanced users exploring data via querying, including web queries, it is frequently difficult to estimate proper generality/specificity of a query being formulated. Lattice querying makes it easier for a broad range of user and data exploration tasks to formulate the query: given a few examples, it formulates the query automatically.

In this work we intend to merge the efficiency of distributed computing framework with the intelligence features of data exploration provided by NLP technologies. We

introduce the technique of lattice querying which automatically forms the query from the set of text samples provided by a user by generalizing them in the level of parse trees. Also the system produces search results by matching parse trees of this query with that of candidate answers. Lattice queries allow increase in big data exploration efficiency since they form multiple "hypotheses" concerning user intent and explore data from multiple angles (generalizations).

Exploring data, mostly keyword query and phrase query are popular, as well as natural language-like ones. Users of search engines appreciate more and more 'fuzzy match' queries, which help to explore new areas where the knowledge of exact keywords is lacking. Using synonyms, taxonomies, ontologies and query expansions helps to substitute user keywords with the domain-specific ones to find what the system believes users are looking for [7].

Nowadays, search engines ranging from open source to enterprise offer a broad range of queries with string character-based similarity. They include Boolean queries, span queries which restrict the distances between keywords in a document, regular expressions queries which allow a range of characters at certain positions, fuzzy match queries and more-like-this which allow substitution of certain characters based on string distances. Other kinds of queries allow expressing constraints in a particular dimension, such as geo-shape query.

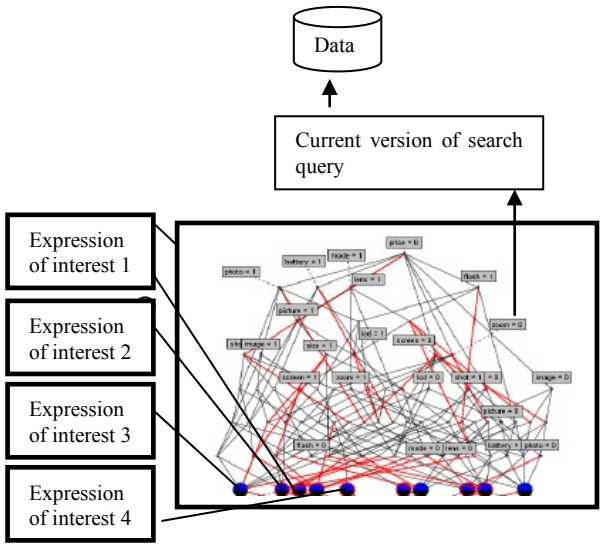


Fig. 1: The idea of lattice query

The idea of lattice query is illustrated in Fig. 1. Instead of a user formulating a query exploring a dataset, he or she proves a few samples (expressions of interest) so that the system builds the lattice of all generalizations for these samples. The system then formulates a query for each lattice node.

Proceeding from a keyword query to regexp or fuzzy one allows making search more general, flexible, assists in exploration of a new domain, as set of

document with unknown vocabulary. What can be a further step in this direction? We introduce lattice queries, based on natural language expressions which are generalized into an actual query. A lattice query contains rich linguistic information, which is derived by generalization of sample expressions (phrases or sentences) specified by the user. Instead of getting search results similar to a given expression (done by 'more like this' query), we first build the commonality expression between all or subsets of the given sample expressions, and then use it as a query. A lattice query includes words as well as attributes such as entity types and verb attributes.

2 Simple lattice queries

Let us start with an employee search example. Let us imagine a company looking for the following individuals:

A junior sale engineer expert travels to customers on site.
A junior design expert goes to customer companies.
A junior software engineer rushes to customer sites.

Given the above set of samples, we need to form a job-search query which would give us candidates somewhat similar to what we are looking for. A trivial approach would be to just turn each sample into a query and attempt to find an exact match. However most of times it would not work, so such queries need to release some constraints. How to determine which constraints need to be dropped and which keywords are most important?

To do that, we apply generalization to the set of these samples. For the entities and attributes, we form the least general generalization. The seniority of the job (adjective) 'junior' will stay. The job activity (noun phrase) varies, so we generalize them into <job-activity>. The higher-level reference to the job is 'expert' and is common for all three cases, so stays. The verb for job responsibility varies, so we use <action>, which can be further specified as <moving_action>, using verb-focused ontologies like VerbNet. To generalize the last noun phrase, we obtain the generalization <customer, NP>.

junior <any job activity> expert <action> customer-NP.

This is a lattice query, which is expected to be run against job descriptions and find the cases which are supposed to be most desired, according to the set of samples.

In terms of parse trees of the potential sentences to be matched with the lattice query, we rewrite it as

JJ-junior NP- NN-expert VP-* NN-customer NP-**

The lattice query read as *find me a junior something expert doing-something-with customer of-something.*

Now we show how this template can applied to accept/reject a candidate answer Cisco junior sale representative expert flew to customers data centers.

We represent the lattice query as a conjunction of noun phrases (NP) and verb phrases (VP) set:

[[NP [DT-a JJ-junior NN- NN-*], NP [NN*-customers]], [VP [VB-* TO-to NN*-customers]]]*

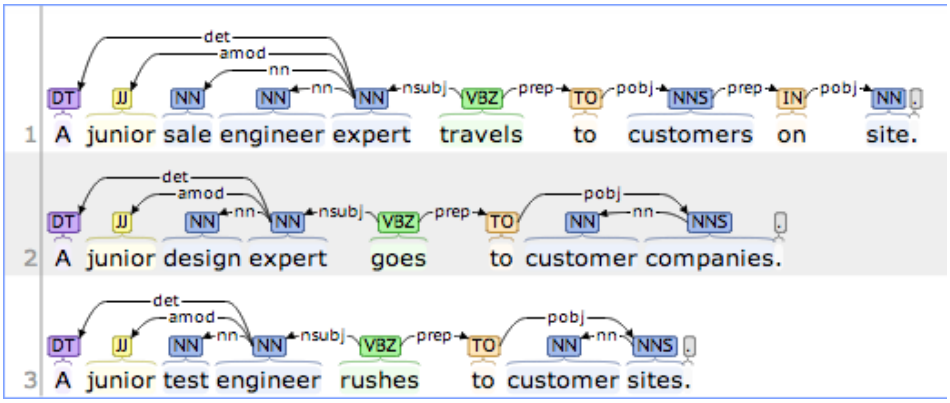
The first NP covers the beginning of the lattice query above, and the second NP covers the end. VP covers the second half of the lattice query starting from *doing-something...*

The generalization between the lattice query and a candidate answer is

[[NP [JJ-junior NN- NN-*], NP [NN*-customers]], [VP [VB-* TO-to NN*-customers]]]*

One can see that the NP part is partially satisfied (the article a does not occur in the candidate answer) and VP part is fully satisfied.

Here are parse trees for three samples



Generalizing these three, we obtain the lattice query to run against a dataset:

[[NP [DT-a JJ-junior NN-* NN-*], NP [NN*-customers]], [VP [VB-* TO-to NN*-customers]]]

One can see that using lattice queries, one can be very sensitive in selecting search results. Searching for a token followed by a word with certain POS instead of just a single token gives a control over false-positive rate. Automated derivation of such constraint allows user to focus on cases instead of making efforts to generate a query which would keep expected search results in and unwanted out.

Definition: a lattice query Q is satisfied by a sentence S, if Q^S = S.

In practice a weak satisfaction is acceptable, where Q^S <= S, but there are constraints on the parts of the lattice query:

- A number of parts in in Q^S should be the same as in Q;
- All words (not POS-* placeholders) from Q should also be in Q^S.

3 More complex lattice queries

Text samples to form a lattice query can be typed, but also can be taken from text already written by someone. To expand the dimensionality of content exploration, samples can be paragraph-size texts.

Let us consider an example of a safety-related exploration task, where a researcher attempts to find a potential reason for an accident. Let us have the following texts as incidents descriptions. These descriptions should be generalized into a lattice query to be run against a corpus of texts for the purpose of finding a root cause of a situation being described.

Crossing the snow slope was dangerous. They informed in the blog that an ice axe should be used. However, I am reporting that crossing the snow field in the late afternoon I had to use crampons.

I could not cross the snow creek since it was dangerous. This was because the previous hiker reported that ice axe should be used in late afternoon. To inform the fellow hikers, I had to use crampons going across the show field in the late afternoon.

As a result of generalization [5, 6, 8] from two above cases, we will obtain a set of expressions for various ways of formulating commonalities between these cases. We will use the following snapshot of a corpus of text to illustrate how a lattice query is matched with a paragraph:

I had to use crampons to cross snow slopes without an ice axe in late afternoon this spring. However in summer I do not feel it was dangerous crossing the snow.

We link two phrases in different sentences since they are connected by a rhetoric relation based on *However* …

```
  rel: <sent=1-word=1..inform> ===> <sent=2-word=4..report>
  From [<1>NP'They':PRP]
  TO    [<4>NP'am':VBP,    NP'reporting':VBG,    <8>NP'the':DT,
<9>NP'snow':NN, <10>NP'field':NN, <11>NP'in':IN, <12>NP'the':DT,
<13>NP'late':JJ,       <14>NP'afternoon':NN,       <15>NP'I':PRP,
<16>NP'had':VBD,          <17>NP'to':TO,          <18>NP'use':VB,
<19>NP'crampons':NNS]
```

We are also linking phrases of different sentences based on communicative actions:

```
  rel: <sent=1-word=6..report> ===> <sent=2-word=1..inform>
  From [<4>NP'the':DT, <5>NP'previous':JJ, <6>NP'hiker':NN]
  TO     [<1>NP'To':TO,     <2>NP'inform':VB,     <3>NP'the':DT,
<4>NP'fellow':JJ, <5>NP'hikers':NNS]
```

As a result of generalizing two paragraphs, we obtain the lattice query:

```
  [[NP [NN-ice NN-axe ], NP [DT-the NN-snow NN-* ], NP [PRP-i
], NP [NNS-crampons ], NP [DT-the TO-to VB-* ], NP [VB-* DT-the
NN-* NN-field IN-in DT-the JJ-late NN-afternoon (TIME) ]], [VP
[VB-was JJ-dangerous ], VP [VB-* IN-* DT-the NN-* VB-* ], VP
[VB-* IN-* DT-the IN-that NN-ice NN-axe MD-should VB-be VB-used
], VP [VB-* NN-* VB-use ], VP [DT-the IN-in ], VP [VB-reporting
IN-in JJ-late NN-afternoon (TIME) ], VP [VB-* NN*-* NN-* NN*-*
], VP [VB-crossing DT-the NN-snow NN-* IN-* ], VP [DT-the NN-*
```

```
NN-field IN-in DT-the JJ-late NN-afternoon (TIME) ], VP [VB-had
TO-to VB-use NNS-crampons ]]]
```

Notice that potential safety-related "issues" are *ice-axe, snow, crampons, being at a … field during later afternoon, being dangerous, necessity to use ice-axe, crossing the snow*, and others. These issues occur in both samples, so that are of a potential interest. Now we can run the formed lattice query against the corpus and observe which issues extracted above are confirmed. A simple way to look at it is as a Boolean OR query: find me the conditions from the list which is satisfied by the corpus. The generalization for the lattice query and the paragraph above turns out to be satisfactory:

```
  [[NP [NN-ice NN-axe ], NP [NN-snow NN*-* ], NP [DT-the NN-
snow ], NP [PRP-i ], NP [NNS-crampons ], NP [NN-* NN-* IN-in JJ-
late NN-afternoon (TIME) ]], [VP [VB-was JJ-dangerous ], VP [VB-
* VB-use ], VP [VB-* NN*-* IN-* ], VP [VB-crossing NN-snow NN*-*
IN-* ], VP [VB-crossing DT-the NN-snow ], VP [VB-had TO-to VB-
use NNS-crampons ], VP [TO-to VB-* NN*-* ]]] => matched
```

Hence we got the confirmation from the corpus that the above hypotheses, encoded into this lattice query, are true. Notice that forming a data exploration queries from the original paragraphs would contain too many keywords and would produce too much marginally relevant results.

4 Evaluation of Performance of Lattice Queries

We conduct evaluation for complex information extraction tasks such as identifying communicative actions and detecting emotional states. Also, we perform evaluation for the rhetoric relation domain: this task is necessary to build a set of parse trees for a paragraph, linking its parse trees. We draw the comparison between information extraction based on the means available within Elasticsearch and Solr framework:

- keyword Boolean queries,
- span queries where the distance between keywords in text is constrained, and
- lattice query-based information extraction.

The corpus is based on he set of customer complains, where both communicative actions and emotions are frequent and essential for complaint analysis tasks. Evaluation was conducted by quality assurance personnel.

We observe in Table 1 that the information extraction F-measure for Keywords and Regular expressions is both 64% for querying indexed data and string search (although the former is about 50 times faster). Relying on span queries gives just 2% increase in F-measure, whereas using lattice queries delivers further 10% improvement.

In this work we introduced a new type of query for search engine framework, the lattice query, which is intended to facilitate the process of an abstract data exploration. Instead of having a user formulate a query, one or more instances are automatically formed from sample expressions. To derive a lattice query, as well as measure relevance of a question to an answer, an operation of syntactic generalization [8, 9] is

used. It finds a maximal common sub-trees between the parse trees for the sample text fragments, and also it finds the maximum common sub-trees between the parse trees for the lattice query and that of the candidate answers. In the latter case, the size of the common sub-trees is a measure of relevance for a given candidate search results.

Method / Task	Keywords and Regexps – finding in string			Keywords and Regexp Queries – Lucene index			Span and 'Like' Queries – Lucene index			Lattice queries First Lucene index then verification by ^		
	P/R		Speed	P/R		Speed	P/R		Speed	P/R		Speed
Extracting communicative actions	64	71	1	63	72	0.02	68	70	0.05	82	75	15.1
Extracting emotional state	62	70	1.2	59	70	0.02	64	68	0.05	80	74	18.2
Extracting rhetoric relation	56	65	1.5	56	66	0.02	59	70	0.05	77	70	25.4

Table 1: Evaluation of lattice query-based information extraction tasks.

We now proceed to an information extraction example in the security domain. One needs to identify text which contains some information of personal interest, such as social security numbers or driver's license numbers, as well as names and addresses of individuals. The requirement is to identify a reference to a person, her activity and a certain document with an id number. Rather than attempting to come up with a rule, a developer of this system specifies the training samples:

"John Doe send her california license 4567456"
"Mary Smith hid her US social number 666-66-6666"
"Jennifer Poppins got her identification 8765"
"Andrew Chen lost his Oregon driver license 731234"

The rules obtained from this samples cover following cases:

"Judith Jain received her washington license 4567456"
"Mary Jones send her Canada prisoner id number 666666666"
"Mary Jones send her Canada prisoner id number 666666666"
"Mary Stewart hid her Mexico cook id number 666666666"
"Robin mentioned her Peru fisher id 2345"

“Peter Doe hid his Bolivia set id number 666666666"
"Robin mentioned her best Peru fisher man id 2345"

But leaves the following cases out:

"Spain hid her Canada driver id number 666666666"
"John Poppins hid her prisoner id 666666666"
"Microsoft announced its Windows Azure release number 666666666"
"John Poppins hid her Google id 666666666"

It should be obvious for the reader the negative set included cases not related to a possible leakage of personal information.

In our evaluation we compared the conventional information extraction approach where extraction rules are expressed using keywords and regular expressions, with the one where rules are frame queries. We observed that frame queries improve both precision and recall of information extraction by producing more sensitive rules, compared to sample expressions which would serve as extraction rules otherwise. An importance of the lattice queries in data exploration is that only the most important keywords are submitted for web search, and neither single document nor keyword overlap deliver such the set of keywords.

References

1. Borgida ER, DL McGuinness, Asking Queries about Frames. Proceedings of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning 1996, 340—349.
2. Bill MacCartney, Michel Galley, and Christopher D. Manning, A phrase-based alignment model for natural language inference. The Conference on Empirical Methods in Natural Language Processing (EMNLP-08), Honolulu, HI, October 2008.
3. Galitsky, B. *Natural Language Question Answering System: Technique of Semantic Headers*. Advanced Knowledge International, Australia (2003).
4. Galitsky, B., Josep Lluis de la Rosa, Gábor Dobrocsi. *Inferring the semantic properties of sentences by mining syntactic parse trees*. Data & Knowledge Engineering. Volume 81-82, November (2012) 21-45.
5. Galitsky, B., Daniel Usikov, Sergei O. Kuznetsov: *Parse Thicket Representations for Answering Multi-sentence questions*. 20th International Conference on Conceptual Structures, ICCS 2013 (2013).
6. Galitsky, B. Machine Learning of Syntactic Parse Trees for Search and Classification of Text. Engineering Application of AI, http://dx.doi.org/10.1016/j.engappai.2012.09.017, (2012).
7. Galitsky, B. Transfer learning of syntactic structures for building taxonomies for search engines. Engineering Applications of Artificial Intelligence. Volume 26 Issue 10, November, 2013, Pages 2504-2515.
8. Galitsky B, Ilvovsky D, Kuznetsov SO, Strok F. Matching sets of parse trees for answering multi-sentence questions. Recent Advances in Natural Language Processing. 2013. doi:http://www.aclweb.org/anthology/R13-1037.
9. Galitsky, B., Learning parse structure of paragraphs and its applications in search. Engineering Applications of Artificial Intelligence 01/2014; 32:160–184.