A language-independent method for the extraction of RDF verbalization templates

Basil Ell
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
basil.ell@kit.edu

Andreas Harth
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
harth@kit.edu

Abstract

With the rise of the Semantic Web more and more data become available encoded using the Semantic Web standard RDF. RDF is faced towards machines: designed to be easily processable by machines it is difficult to be understood by casual users. Transforming RDF data into human-comprehensible text would facilitate non-experts to assess this information. In this paper we present a language-independent method for extracting RDF verbalization templates from a parallel corpus of text and data. Our method is based on distant-supervised simultaneous multi-relation learning and frequent maximal subgraph pattern mining. We demonstrate the feasibility of our method on a parallel corpus of Wikipedia articles and DBpedia data for English and German.

1 Introduction

Natural Language Generation (NLG) systems require resources such as templates (in case of template-based NLG) or rules (in case of rule-based NLG). Be it template-based or rule-based systems, these resources limit the variability and the domain-specificity of the generated natural language output and manual creation of these resources is tedious work.

We propose a language-independent approach that induces verbalization templates for RDF graphs from example data. The approach is language-independent since it does not rely on pre-existing language resources such as parsers, grammars or dictionaries.

Input is a corpus of parallel text and data consisting of a set of documents D and an RDF graph G, where D and G are related via a set of entities E where an entity can be described by a document in D and described by data in G. Output is a set of templates. Templates consist of a graph pattern that can be applied to query the graph and of a sentence pattern that is a slotted sentence into which parts of the query result are inserted. A template enables verbalization of a subgraph of G as a complete sentence.

An example is shown in Fig. 1.[1] The graph pattern GP can be transformed into a SPARQL query Q_{GP}. Querying the data graph G results in the graph G_{GP}. G_{GP} can be verbalized as an English (German) sentence S_{en} (S_{de}) using the sentence pattern SP_{en} (SP_{de}).

The approach employs the distant supervision principle (Craven and Kumlien, 1999; Bunescu and Mooney, 2007; Carlson et al., 2009; Mintz et al., 2009; Welty et al., 2010; Hoffmann et al., 2011; Surdeanu et al., 2012) from relation extraction: training data is generated automatically by aligning a database of facts with text; therefore, no hand-labeled data is required. We apply simultaneous multi-relation learning (Carlson et al., 2009) for text-data alignment and frequent maximal subgraph pattern mining to observe commonalities among RDF graph patterns.

Besides the general idea to allow for non-experts to assess information encoded in RDF, we envision application of these verbalization templates in three scenarios:

(1) In query interfaces to semantic databases, casual users - usually not capable of writing formal queries – specify their information needs using keywords (Lei et al., 2006; Thomas et al., 2007; Wang et al., 2008), questions in free-text or using a controlled language (Kaufmann et al., 2006; Cimiano et al., 2008; Wendt et al., 2012; Damljanovic et al., 2012), or forms (Hunter and Odat, 2011; Mendes

[1] Further examples and the evaluation material can be found on our website at http://km.aifb.kit.edu/sites/bridge-patterns/INLG2014

Proceedings of the 8th International Natural Language Generation Conference, pages 26–34,
Philadelphia, Pennsylvania, 19-21 June 2014.

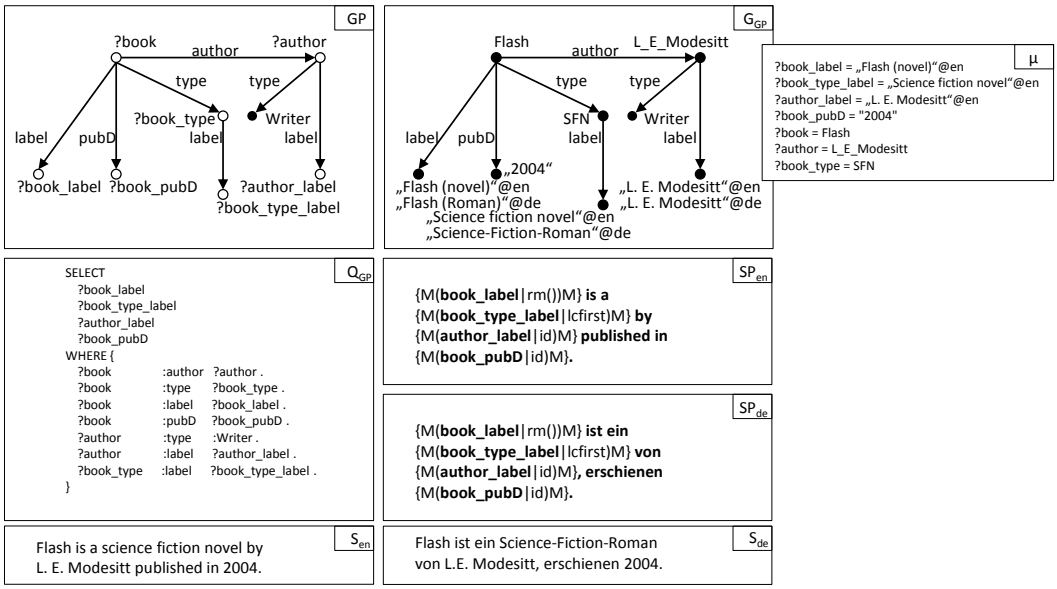

Figure 1: A template consists of a graph pattern GP and a sentence pattern SP. The graph pattern GP can be transformed into a SPARQL query Q_{GP}. A result of querying the data graph is the RDF graph G_{GP} with the list of solution mappings μ. This graph can be verbalized as an English sentence S_{en} using the English sentence pattern SP_{en} or as a German sentence S_{de} using the German sentence pattern SP_{de}. The modifiers, e.g. `lcfirst`, are explained in Table 1.

et al., 2008). The system queries an RDF database according to its interpretation of the input. Query results could be verbalized.

(2) Since the introduction of the Google Knowledge Graph,[2] when searching for an entity such as the city of Karlsruhe via Google, besides the search results shown on the left a table is displayed on the right which provides a short description of the entity taken from Wikipedia. While these descriptions are decoupled from data in the knowledge graph they could be generated automatically.

(3) The collaboratively-edited knowledge base Wikidata provides machine-readable data which can be used, e.g., by the Wikipedia. The Wikidata browsing interface *reasonator* currently explores the use of template-based NLG in order to provide human-readable descriptions of its entities.[3] Since the templates are created manually, currently only for few types of entities these verbalizations can be provided.

[2] `http://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html` (accessed 2014-03-20)

[3] See for example the page about Johann Sebastian Bach: `http://tools.wmflabs.org/reasonator/?q=Q1339` (accessed 2014-03-20)

1.1 Main contributions

We present an approach to induce RDF verbalization templates from a parallel text-data corpus. (1) The approach is distant-supervised, since it does not require labeled data but instead automatically aligns a database of facts with text by performing simultaneous multi-relation learning. (2) Hypotheses about a sentence's content are represented as an RDF graph pattern. Hypotheses graphs are reduced via frequent maximal subgraph pattern mining. (3) We introduce RDF verbalization templates consisting of a sentence pattern which includes modifiers and a graph pattern of unrestricted size. (4) Our approach does not use language resources such as parsers or dictionaries and is thus language independent and (5) does not depend on a certain ontology or domain. (6) The feasibility of the approach is validated for English and German given a large dataset which resulted in the induction of a large number of templates that are general in terms of enabling verbalization of numerous subgraphs in our dataset.

1.2 Definitions

- A **template** is a tuple (sp, gp) where sp is a sentence pattern and gp is a graph pattern. We denote the set of variables within a sentence pattern sp as $Var_{SP}(sp)$ and the

set of variables within a graph pattern gp as $Var_{GP}(gp)$. A template (sp, gp) is *safe* if the set of variables within the sentence pattern is a subset of the set of variables within the graph pattern: $Var_{SP}(sp) \subseteq Var_{GP}(gp)$.

- A **sentence pattern** (SP) is a string that consists of terminals, variables, and modifiers. Within an SP a (var., mod.) tuple (v, m) is denoted as {M(v|m)M}. {M(and)M} serve as delimiters of a (var., mod.) tuple.
- A **graph pattern** is a set of triples patterns (s, p, o) where $s \in \mathcal{U} \cup \mathcal{V}$, $p \in \mathcal{U} \cup \mathcal{V}$, and $o \in \mathcal{U} \cup \mathcal{L} \cup \mathcal{V}$. $\mathcal{U}$ is a set of identifiers, $\mathcal{L}$ is a set of literals, and $\mathcal{V}$ is a set of variables.
- A **modifier** $m \in M$ is a function applicable to the value of a variable v - denoted by $m(v)$.

1.3 Template-based NLG

A template can be applied for Natural Language Generation as follows. Given an RDF data graph G and a template (sp, gp), a SPARQL SELECT query can be created: SELECT PV WHERE { gp' }. The list of projection variables PV is the list of variables $v \in Var_{SP}(sp)$. gp' is constructed by adding each triple pattern to gp'. An example of a query (Q_{GP}) created from a graph pattern (GP) is shown in Fig. 1.

Executing a query results in a solution sequence[4] which is a list of solution mappings $\mu: V \rightarrow \mathcal{T}$ from a set of variables V to a set of RDF terms $\mathcal{T} = \mathcal{U} \cup \mathcal{L}$. See Fig. 1 for an example of a solution mapping (μ).

For each non-terminal in sp representing a variable-modifier tuple (v, m), the modifier m is applied on $\mu(v)$ resulting in $m(\mu(v))$. Finally, the tuple (v, m), expressed as {M(v|m)M}, is replaced in sp with $m(\mu(v))$. After replacing each such tuple the sentence creation is complete.

2 Parallel corpus

Our approach requires a parallel corpus of text and data and consists of texts that describe entities in natural language and a data graph that semantically describes entities.

Formally, the parallel corpus consists of a set of entities E, a set of documents D, and an RDF data graph G. An entity can be described by a document in a certain language. The relation $document \subseteq E \times L \times D$ relates an (entity, language) tuple to a set of documents. $document(e, l)$ denotes the (potentially empty) set of documents that describe an entity $e \in E$ in language $l \in L$.

G is a set of triples (s, p, o) where $s, p \in \mathcal{U}$, and $o \in \mathcal{U} \cup \mathcal{L}$.

Each literal has a datatype, returned by the function $datatype: \mathcal{L} \rightarrow T$. Literals of type *string* can be language-tagged. The function $l_l: \mathcal{L} \rightarrow L$ returns the language $l_l(r) \in L$ for a literal $r \in \mathcal{L}$ if it exists. An entity can have a human-readable form which is a language-tagged literal. The relation $\lambda \subseteq E \times L \times \mathcal{L}$ relates an entity $e \in E$ to a (possibly empty) set of literals $\lambda(e, l) \subseteq \mathcal{L}$ in language $l \in L$. The property p_λ relates an entity with its label.

Each entity $e \in E$ occurs in the data graph. This means that G contains a triple (s, p, o) where $s = e$ or $o = e$.

3 Approach

Our approach consists of six steps:

1. For each entity $e \in E$ we collect sentences from documents about the entity that mention the entity.
2. For each sentence we align the sentence and the data graph by iteratively exploring the vicinity of the entity within the graph. This leads to a set of identified entities: entities that are believed to be mentioned within the sentence; and a set of observations. The subgraph of G that consists of all triples that contain an identified entity serves as an hypothesis graph: no fact that is not expressed in the subgraph is expressed in the sentence.
3. Each (sentence, identified entities, graph) triple is abstracted by replacing identified literals in the sentence and in the graph with variables and by replacing identified entities in the graph with variables. This step can lead to multiple distinct (sentence pattern, graph pattern) tuples for each sentence. This abstraction enables comparing different sentences that share the same sentence pattern after abstraction.
4. A set of (sentence pattern, graph patterns) tuples is built for each sentence pattern.
5. For each (sentence pattern, graph patterns) tuple the set of graph patterns is analyzed regarding their commonalities. This is realized via the frequent and maximal subgraph pat-

[4] We adopt the terminology from the SPARQL 1.1 Query Language documentation available at http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

tern extraction (fmSpan) algorithm which results in a set of graph patterns that are subgraph patterns to the input graph patterns.

6. Given the output of this algorithm and the abstracted sentences the templates are created.

3.1 Sentence collection

Given a language name l, a set of entities E, a set of documents D and an ordered list of modifiers M, for each entity $e \in E$ for each document $d \in document(e, l)$ (which describes e in language l) the document is split into a set of sentences. For each sentence that has an acceptable length (measured as the number of characters), for each label $x \in \lambda(e, l)$ and for each string modifier $m \in M_{string}$, we store the (sentence, entity, left, right, λ, matched) tuple if the modified label $m(x)$ matches the sentence. See Alg. 1.

Algorithm 1 Collect example sentences

```
1:  procedure COLLECT EXAMPLE SENTENCES(l, E, D, M)
2:   for each e ∈ E do
3:    for each d ∈ document(e, l) ∈ D do
4:     for each s ∈ sentences(d, l_min, l_max) do
5:      for each x ∈ λ(e, l) do
6:       for each m ∈ M_string do
7:        if applicable(m, x) then
8:         (left, right, x') = MatchesLabel(s, x, m, "str")
9:         if (left, right, x') ≠ ∅ then
10:         Output (s, e, left, right, x, x', m)
11:         Continue with next sentence
```

Algorithm 2 MatchesLabel

```
1:  procedure MATCHESLABEL(s, x, m, t)
2:   if t = "str" ∨ t ≠ "integer" then
3:    if length(x) ≥ 4 then
4:     if s matches (\W|^)\w{l0, l1}m(x)\w{r0, r1}(\W|$) then
5:      return (left, right, matched)
6:   else if t = integer then
7:    if s matches (\D|^)m(x)(\D|$) then
8:     return (left, right, matched)
9:   return ∅
```

In Alg. 2, $\backslash W$ denotes a non-word character (such as a blank), $\backslash D$ denotes a non-digit, $\backslash w$ denotes a word character[5] (such as "x"), $\backslash w\{a, b\}$ denotes a sequence of at least a word-characters and not more than b word characters, l_0 and l_1 are the minimum and maximum number of word characters that may appear on the left side of the modified string $m(x)$ between this string and a non-word character or the beginning of the sentence ($\hat{}$). r_0 and r_1 are the corresponding numbers regarding the right side of the modified string. \$ denotes the end of the sentence. In case of a match, the string that is matched by $\backslash w\{l_0, l_1\}$ is stored as *left*, the string that is matched by $\backslash w\{r_0, r_1\}$ is stored as *right* and the part that matches $m(x)$ is stored as *matched*. Note that *matched* can be different from $m(x)$ since the application of the modifier m to the string x can result in a regular expression that contains information for the matcher specifying that a part of the string needs to be matched case-insensitively.

Allowing a certain number of characters to be added to the left and to the right of a string has the intention to match even though prefixes and postfixes are added. For example, this allows to match "German" within its plural form "Germans" or to match "magic" within magician.

3.2 Sentence and data alignment

The sentence and the data graph are aligned by iteratively exploring the vicinity of an entity within the graph. This exploration is described by Alg. 3 which builds a set of observations *obs*. A member of this set is a 7-tuple where the first three members form a triple (entity, property, literal value), followed by the strings matched to the left and the right, the matched string and the modifier. Moreover, the algorithm creates a graph $graph \subseteq G$ consisting of all triples that contain an identified entity. Here, an entity e is identified if a modifier $m \in M$ exists such that an $x \in \lambda(e, l)$ exists such that the sentence matches $m(x)$. Output is the original sentence, the set of identified entities, the set of observations, and the subgraph.

Algorithm 3 Data Collection

```
1:  procedure COLLECTDATA(s, e, lang, left, right, x, x', m)
2:   identified = ∅; todo = (e); done = ∅;
3:   graph = ∅; obs = {(e, p_l, x, left, right, x', m)}
4:   while todo ≠ ∅ do
5:    e ← todo.first
6:    todo ← todo\{e}; done ← done ∪ {e}
7:    for each (e, p, o) ∈ G do
8:     graph ← graph ∪ {(e, p, o)}
9:     if o is a literal then
10:     (l, r, o', m) = CL(s, o)
11:     if (l, r, o', m) ≠ ∅ then
12:      obs ← obs ∪ {(e, p, o, l, r, o', m)}
13:      if o = λ(e, lang) then
14:       identified ← identified ∪ {e}
15:     else if o is a URI then
16:      if o ∉ done ∧ o ∉ todo then
17:       todo.add(o)
18:    for each (e2, p, e) ∈ G do
19:     graph ← graph ∪ {(e2, p, e)}
20:     if e2 ∉ done ∧ o ∉ todo then
21:      todo.add(e2)
22:   Output (sentence, identified, obs, graph)
```

3.3 Sentence and graph abstraction

In the previous step, ambiguities may exist. For example, given two triples (e_1, p_1, v) and

[5]Note that word- and non-word characters are language-specific and may be defined for each language individually.

(e_2, p_2, v) where v is a literal value and the entities e_1 and e_2 are identified, if the value v is found in the sentence, then it cannot be resolved whether the sentence expresses the fact (e_1, p_1, v) or (e_2, p_2, v). Therefore, for each situation where a value appears in two distinct contexts or where values overlap (two values overlap if the intervals of their character positions overlap), the sentence and graph pattern is copied and on each copy another abstraction is performed thus leading to multiple abstractions per sentence. Alg. 4 iteratively creates all sentence abstractions given a language name l, a sentence S, and a set of observations obs. The function `poss` evaluates the set of observations that are still valid. The function `apply` replaces a string in a sentence with variable and modifier. Thereby, the *left* and *right* parts are added to the modifier. For an observation (e, p, o, l, r, o', m) the string concatenation $l{+}o'{+}r$ is replaced with $\{$`M(`v_i`|m')M`$\}$. For each observation a new variable v_i is introduced. m' is a modifier to which the modifiers `+l(l)` and `+r(r)` are appended which denote that certain strings are added to the left and the right of the literal. The graph is abstracted by replacing the triple (e, p, o) with the triple pattern (e, p, v_1). After completely abstracting a sentence pattern, each identified entity is replaced by a variable; triples that do not contain any variable are removed.

Algorithm 4 Sentence abstraction

```
1:  procedure AbsSentence(l, S, obs)
2:    P ← poss(l, S, obs)
3:    if P = ∅ then
4:      Output(S)
5:    else
6:      O ← overlap(S, P)
7:      for each p ∈ P do
8:        if p ∉ O then
9:          S ← apply(S, p)
10:     if O = ∅ then
11:       Output(S)
12:     else
13:       for each p ∈ O do
14:         S' ← apply(S, p)
15:         AbsSentence(l, S', P)
```

3.4 Grouping

Given a set of (sp, gp) tuples, for each language we build groups of tuples where in each group the sentence patterns are pairwise equivalent when ignoring modifiers. Sentence patterns sp_i and sp_j are equivalent if either they are identical ($sp_i{=}sp_j$), or if an injective function $m{:}Var_{SP}(sp_i){\rightarrow}Var_{SP}(sp_j)$ exists such that when each variable v in sp_i is replaced with $m(v)$, the resulting string sp_i' is identical to sp_j.

For each group the set of graph patterns is used as input for the algorithm presented in the following section.

3.5 Frequent maximal subgraph pattern extraction

Before we describe the *fmSpan* algorithm (fmSpan: Frequent Maximal Subgraph PAttern extractioN) we need to introduce our notation:

Two graph patterns gp_i and gp_j are *equivalent* ($gp_i{=}gp_j$) if an injective function $m{:}Var_{GP}(gp_i){\rightarrow}Var_{GP}(gp_j)$ exists such that when each variable v in gp_i is replaced with $m(v)$, the resulting graph pattern gp_i' is identical to gp_j. A graph pattern gp_i is *subgraph pattern* to another graph pattern gp_j, denoted by $gp_i{\subseteq^p}gp_j$, if an injective function $m{:}Var_{GP}(gp_i){\rightarrow}Var_{GP}(gp_j)$ exists such that when each variable v in gp_i is replaced with $m(v)$, resulting in gp_i', each triple pattern in gp_i' is also a triple pattern in gp_j. Given a set of graph patterns $GP{=}\{gp_1, ..., gp_n\}$ and given a graph pattern x, the *coverage* of x regarding GP is the number of graphs in GP to which x is a subgraph pattern: $c(x, GP) := |\{gp_i \in GP | x \subseteq^p gp_i\}|$.

Given a set of graph patterns $I{=}\{gp_1, ..., gp_n\}$, from the set of all subgraph patterns $P{=}2^{gp_1} \cup ... \cup 2^{gp_n}$ a set of graph patterns $K{=}\{gp_i, ..., gp_j\}{\subseteq}P$ is selected where:

1. for each $gp_k \in K$:
   (a) $c(gp_k, I) \geq min\_coverage$
   (b) $\neg\exists gp_l \in P : gp_k \neq gp_l \wedge gp_k \subseteq^p gp_l \wedge c(gp_l, I) \geq min\_coverage$
2. $\neg\exists gp_l \in P : c(gp_l, I) \geq min\_coverage \wedge (\neg\exists gp_m \in P : \neg(gp_m, gp_l) \wedge c(gp_m, I) \geq min\_coverage) \wedge gp_l \notin K$

This means that each member of K is sufficiently frequent (1a) and maximal (2b) and that every maximal graph pattern is contained in K (2).

3.6 Template creation

For each (sentence pattern, graph patterns) tuple the frequent maximal subgraph pattern mining is performed on the group of graph patterns which results in a set K of subgraph patterns. Each $k \in K$ is pruned with Alg. 5. Thereby, if a variable appears in a high number of triples that do not contain any other variable, then these triples are removed. After the pruning each $k \in K$ is then rejected if it is either not safe, not connected, or, when queried against G returns no results.

Algorithm 5 Graph-pattern pruning

1: **procedure** PRUNEGRAPHPATTERN(k)
2: **for each** $v \in Var_{GP}(k)$ **do**
3: $T \leftarrow \{(s,p,o) \in k | (s = v \wedge o \notin Var_{GP}(k)) \vee (o = v \wedge s \notin Var_{GP}(k))\}$
4: **if** $|T| > max_t$ **then**
5: $k \leftarrow k \backslash T$

Datatype	Modifier	Description
xsd:string	id	Does not change the string.
	lcfirst	Sets the first char to lower case if that char is upper case.
	ucfirst	Sets the first char to upper case if that char is lower case.
	case-i	Case-insensitive match
	rm()	If a string ends with a string in round braces, e.g. "Dublin (Ohio)", that part is cut off.
	-1r	Removes the rightmost char.
xsd:gYear	YYYY	Transforms a year value into a four-digit representation.
xsd:integer	integer_id	Does not change the integer.
	enInt_sep	Adds English thousands separators, e.g., *10,000*.
	deInt_sep	Adds German thousands separators, e.g., *10.000*.
xsd:date	enM,_D_Y	Result, e.g., *March, 22 2014*
	enD_M_Y	Result, e.g., *22 March 2014*
	deM_D,_Y	Result, e.g., *März 22, 2014*
	deD_M_Y	Result, e.g., *22. März 2014*

Table 1: List of modifiers per datatype

4 Experiments

We created a multilingual (English, German) parallel text-data corpus using data from DBpedia[6] and documents from the Wikipedia. The graph G consists of $88,708,622$ triples, the set of documents D consists of $4,004,478$ English documents and $716,049$ German documents. The corpus relations and functions are defined as follows:

- $document(e,l) :=$ $\{d | (e, \text{dbo:abstract}, "d"@l) \in G\}$.
- $\lambda(e,l) := \{v | (e, \text{rdfs:label}, "v"@l) \in G\}$
- The $datatype$ of a literal `"r^^t"` is t.
- The language l_l of a literal `"d"@l` is l.

The modifiers we used in the experiment are given in Table 1.[7] Application of date and integer modifiers may also depend on the language of a sentence. On a value a list of modifiers can be applied. The list of string modifier lists is shown in Fig. 2. The table also shows how often each list of modifiers was applied during the abstraction of English and German sentences.

	groups ≥ 5	templates	all groups
en	4569	3816	686,687
de	2130	1250	269,551

Table 3: Number of groups with a cardinality ≥ 5, the number of induced templates and the number of all groups.

We created two sets of entities E_{en} (E_{de}): those for which an English (German) document exist that consists of at least 100 characters. E_{en} and E_{de} contain $3,587,146$ and $613,027$ entities, respectively. For each entity for each document we split the text into sentences using the Perl module *Lingua::Sentence*[8] and discarded sentences that do not end with a full stop, an exclamation mark, or a question mark or that were shorter (longer) than 50 (200) characters. We used the set of string modifiers presented in Fig. 2 to identify entities via occurrence of a modified version of their labels in a sentence. The results are $3,811,992$ ($794,040$) English (German) sentences.

Abstraction resulted in 3,434,108 (530,766) abstracted English (German) sentences where at least two entities are identified per sentence.

The group size histogram is displayed in Fig. 2.[9] The majority (90%) of all groups of English (German) sentences contain between 5 and 164 (5 and 39) sentences.

Table 3 gives for each language the number of groups that contain more than 5 graph patterns, the number of templates we induced, and the number of all groups. Results of the coverage evaluation $cov_t(G)$ are shown as a histogram in Fig. 3. It shows that for the majority of the templates a high number of subgraphs of G can be verbalized, which means that the templates are not fitted to only a small number of subgraphs: e.g. for 221 English templates verbalize between 10^5 and 10^6 subgraphs, each.

[6] `http://wiki.dbpedia.org/Downloads39`
We used the files long_abstracts_en, long_abstracts_en_uris_de, mappingbased_properties_en, raw_infobox_properties_en, article_categories_en, instance_types_en, labels_en, labels_en_uris_de, category_labels_en, and category_labels_en_uris_de.

[7] Modifiers are only applied if their application to a literal modifies that literal. For example, if a string begins with a lower case character, then the *lcfirst* modifier is inapplicable.

[8] `http://search.cpan.org/~achimru/Lingua-Sentence-1.05`

[9] We cut off the long tail.

No	Modifier list	en	de	No	Modifier list	en	de	No	Modifier list	en	de
(1)	id	10,619,509	1,349,922	(9)	-1r	42,754	15,025	(17)	-1r, -1r, lcfirst	8430	90
(2)	lcfirst	141,865	868	(10)	-1r, lcfirst	7513	99	(18)	-1r, -1r, ucfirst	1020	5
(3)	ucfirst	11,018	8	(11)	-1r, ucfirst	875	4	(19)	-1r, -1r, case-i	733	92
(4)	case-i	295,593	16,351	(12)	-1r, case-i	863	50	(20)	rm(), -1r, -1r, lcfirst	0	0
(5)	rm()	2705	762	(13)	rm(), -1r, lcfirst	0	0	(21)	rm(), -1r, -1r, ucfirst	0	0
(6)	rm(), lcfirst	13	0	(14)	rm(), -1r, ucfirst	0	0	(22)	rm(), -1r, -1r, case-i	66	1
(7)	rm(), ucfirst	0	0	(15)	rm(), -1r, case-i	55	6				
(8)	rm(), case-i	50	0	(16)	-1r, -1r	39,113	11,632				

Table 2: List of lists of string modifiers and their number of applications

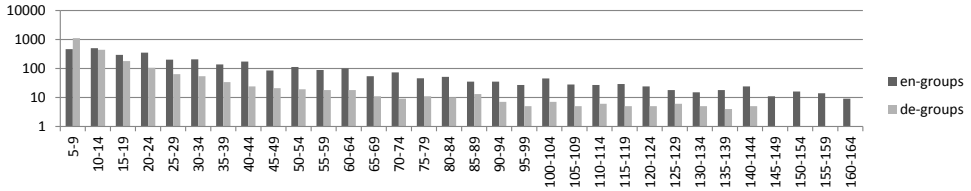

Figure 2: Histogram depicting how often sentence groups occurred with a particular size

5 Evaluation

We evaluate the results from the experiment described in the previous section along the dimensions *coverage*, *accuracy*, *syntactic correctness*, and *understandability* where the latter three are inspired by (Lester and Porter, 1997; Mellish and Dale, 1998; Reiter and Belz, 2009).

Coverage: we define $cov(t, G)$ of a template $t{=}(sp, gp)$ regarding a data graph G as the number of subgraphs of G that can be verbalized with that template i.e. match gp.

Accuracy: is measured in two parts:

1. The extent to which everything that is expressed in gp is also expressed in sp is measured for each triple pattern within the graph pattern on a 4-point scale: (1) *The triple pattern is explicitly expressed*, (2) *The triple pattern is implied*, (3) *The triple pattern is not expressed*, and (4) *Unsure*.
2. The extent to which the sp expresses information that is expr. in gp is measured on a 4-point scale: (1) *Everything is expressed*, (2) *Most things are expressed*, (3) *Some things are expressed*, and (4) *Nothing is expr.*.

Syntactic correctness: the degree to which the verb. is syntactically correct, in particular whether it adheres to English or German grammar: (1) The verb. is completely synt. correct. (2) The verb. is almost synt. correct. (3) The verb. presents some syntactical errors. (4) The verb. is strongly synt. incorrect.

Understandability: Adapted from (Nagao et al., 1985): (1) The meaning of the verb. is clear. (2) The meaning of the verb. is clear, but there are some problems in word usage, and/or style. (3) The basic thrust of the verb. is clear, but the evaluator is not sure of some detailed parts because of word usage problems. (4) The verb. contains many word usage problems, and the evaluator can only guess at the meaning. (5) The verb. cannot be understood at all.

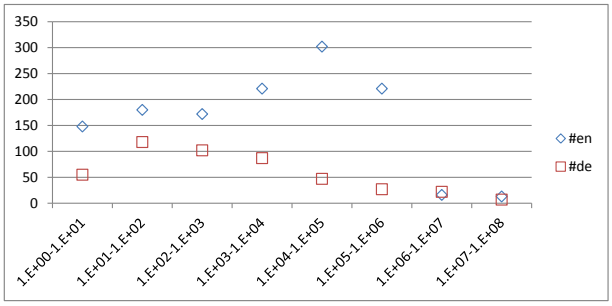

Figure 3: Histogram of the coverage $cov(t, G)$

We evaluated a random sample of 10 English and 10 German templates using a group of 6 evaluators which are experts in the fields of RDF and SPARQL and that are proficient in both English and German. Each template was evaluated by 3 experts, each expert evaluated 10 templates. For each template we retrieved a maximum of 100 subgraphs that matched the graph pattern, randomly selected 10 subgraphs and verbalized them. For each template an evaluator was asked to evaluate accuracy given the graph pattern and given the sentence pattern and, given the list of 10 verbalizations, to evaluate each sentence regarding syntactic correctness and understandability.

$cov(t, G)$ of all 5066 templates is shown in Fig. 3. For example, it shows that there are about 300 templates where each template can be used

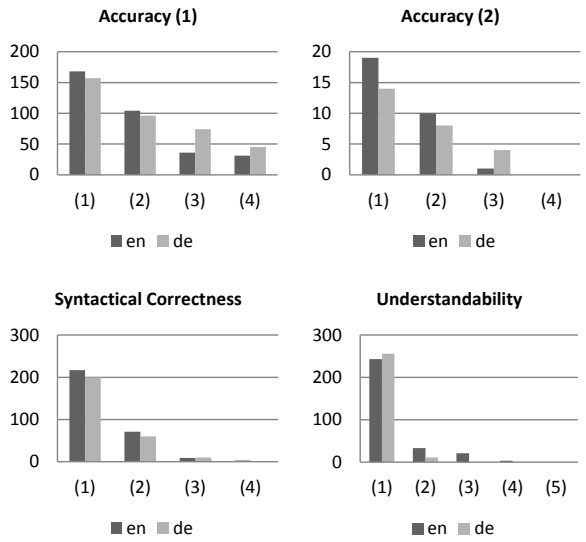

Figure 4: Evaluation results regarding accuracy, syntactical correctness, and understandability

to verbalize between 10^4 and 10^5 subgraphs of G. Results regarding the remaining dimensions are shown in Fig. 4. The values of the x-axes correspond to the scale of the respective dimension. The majority of the triple patterns are either explicitly or implicitly expressed in the sentence pattern. However, some triple patterns are not expressed in the sentence pattern. Syntactical correctness and understandability are mostly high.

6 Related work

(Welty et al., 2010) present *a technique for reading sentences and producing sets of hypothetical relations that the sentence may be expressing*. Given a parallel text-data corpus, entities identified as proper nouns in parsed sentences are replaced with variables. For each (pattern, set of relations) tuple for each sentence that matches this pattern it is counted in how many sentences that match this pattern a certain relation exists between the two entities identified in the sentence. This leads to positive weights assigned to patterns. Negative weights are assigned by applying patterns to sentences, identifying the entities and assigning a negative weight to the relation if the relation expressed by the pattern is not expressed in the data.

In contrast to this approach, our approach 1) does not require to parse input sentences 2) does not only regard relations between proper nouns, 3) constrains candidate entities to the vicinity of already identified entities. Moreover, 4) our approach takes into account the graph of entities identified in a sentence (hypothesis graphs) compared to sets of relations and can thus express multiple relations between entities.

(Duma and Klein, 2013) present an unsupervised approach to NLG template extraction from a parallel text-data corpus. Similar to our approach, text and data are aligned by identifying labels of entities in sentences. The search space is limited by only allowing to match entities that are directly linked to the entity a text is about. Sentences are abstracted by replacing the entity with the name of the property that links the entity with the entity the text is about thus limiting the depth of the graph to 1. Abstracted sentences are parsed and pruned by removing constituents that could not be aligned to the database and by removing constituents of certain classes and then post-processed using manually created rules.

(Gerber and Ngomo, 2011) present an approach to learning natural language representations of predicates from a parallel text-data corpus. For each predicate where a tuple of entities is identified in a sentence, the predicate's natural language representation is the string between the two entities, e.g. `'s acquisition of` for the predicate `subsidiary` and the sentence *Google's acquisition of Youtube comes as online video is really starting to hit its stride.* The main differences to our approach are 1) that we do not focus on learning how a single predicate is expressed but rather how a graph, consisting of multiple related entities, can be expressed in natural language and 2) that a relation between two entities is not only expressed by the string between two entities.

7 Conclusions

We have shown that verbalization templates can be extracted from a parallel text-data corpus in a distant-supervised manner – without the need for pre-existing language resources such as parsers, grammars or dictionaries – and that applying these templates for NLG leads to promising results. The main novelty is the application of frequent maximal subgraph pattern mining for the purpose of analyzing commonalities in sets of hypotheses graphs. Even though the approach is linguistically shallow, verbalizations are already syntactically mostly correct and understandable.

Acknowledgements

The authors acknowledge the support of the European Commission's Seventh Framework Programme FP7-ICT-2011-7 (XLike, Grant 288342).

References

Razvan Bunescu and Raymond Mooney. 2007. Learning to extract relations from the web using minimal supervision. In *Annual meeting-association for Computational Linguistics*, volume 45, pages 576–583.

Andrew Carlson, Justin Betteridge, Estevam R Hruschka Jr, and Tom M Mitchell. 2009. Coupling semi-supervised learning of categories and relations. In *Proceedings of the NAACL HLT 2009 Workshop on Semi-supervised Learning for Natural Language Processing*, pages 1–9. Association for Computational Linguistics.

P. Cimiano, P. Haase, J. Heizmann, M. Mantel, and R. Studer. 2008. Towards portable natural language interfaces to knowledge bases–The case of the ORAKEL system. *Data & Knowledge Engineering*, 65(2):325–354.

Mark Craven and Johan Kumlien. 1999. Constructing biological knowledge bases by extracting information from text sources. In Thomas Lengauer, Reinhard Schneider, Peer Bork, Douglas L. Brutlag, Janice I. Glasgow, Hans-Werner Mewes, and Ralf Zimmer, editors, *ISMB*, pages 77–86. AAAI.

D. Damljanovic, M. Agatonovic, and H. Cunningham. 2012. FREyA: An interactive way of querying Linked Data using natural language. In *The Semantic Web: ESWC 2011 Workshops*, pages 125–138. Springer.

Daniel Duma and Ewan Klein, 2013. *Generating Natural Language from Linked Data: Unsupervised template extraction*, pages 83–94. Association for Computational Linguistics, Potsdam, Germany.

Daniel Gerber and A-C Ngonga Ngomo. 2011. Bootstrapping the linked data web. In *1st Workshop on Web Scale Knowledge Extraction @ International Semantic Web Conference*, volume 2011.

Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S Weld. 2011. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 541–550. Association for Computational Linguistics.

J. Hunter and S. Odat. 2011. Building a Semantic Knowledge-base for Painting Conservators. In *E-Science (e-Science)*, pages 173–180. IEEE.

E. Kaufmann, A. Bernstein, and R. Zumstein. 2006. Querix: A natural language interface to query ontologies based on clarification dialogs. In *International Semantic Web Conference (ISWC 2006)*, pages 980–981.

Yuangui Lei, Victoria Uren, and Enrico Motta. 2006. SemSearch: A Search Engine for the Semantic Web. pages 238–245. Springer.

J.C. Lester and B.W. Porter. 1997. Developing and empirically evaluating robust explanation generators: The KNIGHT experiments. *Comp. Linguistics*, 23(1):65–101.

C. Mellish and R. Dale. 1998. Evaluation in the conext of natural language generation. *Computer Speech and language*, 12(4):349–374.

P.N. Mendes, B. McKnight, A.P. Sheth, and J.C. Kissinger. 2008. TcruziKB: Enabling Complex Queries for Genomic Data Exploration. In *Semantic Computing, 2008*, pages 432–439.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - ACL-IJCNLP 09*, pages 1003–1011.

Makoto Nagao, Jun-ichi Tsujii, and Jun-ichi Nakamura. 1985. The Japanese government project for machine translation. *Computational Linguistics*, 11(2-3):91–110.

E. Reiter and A. Belz. 2009. An investigation into the validity of some metrics for automatically evaluating natural language generation systems. *Computational Linguistics*, 35(4):529–558.

Mihai Surdeanu, Julie Tibshirani, Ramesh Nallapati, and Christopher D Manning. 2012. Multi-instance multi-label learning for relation extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 455–465. Association for Computational Linguistics.

E. Thomas, J.Z. Pan, and D. Sleeman. 2007. ONTOSEARCH2: Searching ontologies semantically. In *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*, pages 70–72.

H. Wang, T. Tran, P. Haase, T. Penin, Q. Liu, L. Fu, and Y. Yu. 2008. SearchWebDB: Searching the Billion Triples. In *Billion Triple Challenge at the International Semantic Web Conference (ISWC 2008)*.

Chris Welty, James Fan, David Gondek, and Andrew Schlaikjer. 2010. Large scale relation detection. In *Proceedings of the NAACL HLT 2010 First International Workshop on Formalisms and Methodology for Learning by Reading*, pages 24–33. Association for Computational Linguistics.

M. Wendt, M. Gerlach, and H. Düwiger. 2012. Linguistic Modeling of Linked Open Data for Question Answering. *Interacting with Linked Data (ILD 2012)*, pages 75–86.