Synonymy Extraction From Semantic Networks Using String and Graph Kernel Methods

Tim vor der Brück[1] **and Yu-Fang Helena Wang**[2]

Abstract. Synonyms are a highly relevant information source for natural language processing. Automatic synonym extraction methods have in common that they are either applied on the surface representation of the text or on a syntactical structure derived from it. In this paper, however, we present a semantic synonym extraction approach that operates directly on semantic networks (SNs), which were derived from text by a deep syntactico-semantic analysis.

Synonymy hypotheses are extracted from the SNs by graph matching. These hypotheses are then validated by a support vector machine (SVM) employing a combined graph and string kernel. Our method was compared to several other approaches and the evaluation has shown that our results are considerably superior.

1 Introduction

In natural language (NL) the same objects or concepts can be referred to with manifold names and in ever-changing ways. It is obvious that a good coverage of synonyms is useful for a wide range of NL applications. In voice search, hot applications are music selection in a local MP3 media library or the navigation to a point of interest. The reference string for speech recognition is provided by the respective database entry, usually the official name. However, that name is often unwieldy, e.g., ‘Five Guys Famous Burgers and Fries’ will often be spoken as ‘Five Guys’; the category ‘Place of worship’ will more simply be referred to as ‘church’.

In this work, we present a purely semantic synonymy extraction method based on semantic networks. This method is hybrid and employs patterns as well as kernel functions and combines the advantages of both methods with each other.

2 Related Work

A lot of research on synonymy extraction has emerged from the meanwhile classical vector space model, in which each word in a corpus is represented as a vector so that all words span a vector space [18, 10]. One base assumption is that words are similar if their associated vectors are located close to one another.

The word vectors represent the textual context, which is given by neighboring words or words that are related to the investigated word by a certain grammatical relation (head-modifier in [21] or predicate-argument structures in [13]).

[1] Universität Frankfurt, vorderbr@em.uni-frankfurt.de
[2] FernUniversität in Hagen, yh.wang@web.de

The element v_i of the vector for a word w determines the frequency of another word u_i appearing in the context of w. The semantic similarity can be estimated by applying a similarity measure on these vectors. Well-known similarity measures are cosine, p-norm, Kullback-Leibler divergence, or information radius [15, 18]. Some approaches use only binary values for vector elements, i.e., $v_i = 1$ if the number of occurrences is above a given threshold. Common binary-valued similarity measures are Matching Coefficient, Dice Coefficient, Jaccard/Tanimoto coefficient, Overlap coefficient, or Cosine [18]. Other methods for estimating semantic similarity consider iterated co-occurrences [4], co-occurrence networks [3] or use an ontology [14, 17, 16].

Yu’s method for synonym extraction comes from a completely different angle. She introduced a syntagmatic, pattern-based approach [25] in which certain key expressions are used to identify patterns. The following patterns are employed:

- $\textsc{syno}(a1, a2) \leftarrow a1$ ... synonym ... $a2$
- $\textsc{syno}(a1, a2) \leftarrow a1$ ... called ... $a2$
- $\textsc{syno}(a1, a2) \leftarrow a1$ ... known as ... $a2$

where $a1$ and $a2$ are the nearest noun phrases around the respective key expression. The method was extended later [26]. The drawback here is that if arbitrary non-noun tokens are allowed between the key expression and $a1$, $a2$, the patterns are easily rendered invalid, consider e.g., the word ‘not’ between $a1$ and ‘known as’ or embedded subclauses between $a1$ and $a2$. However, if the patterns are considered as closed expressions without token insertion, they are often too specific and cannot be applied.

Although rather rarely used for the extraction of lexical relations like synonymy, kernel functions are often applied for semantic relation extraction. This is often done by comparing the paths in a dependency tree [27, 1, 5] or the dependency subtrees containing the two relation candidate components [2, 5].

3 MultiNet

MultiNet is the underlying SN formalism for our text mining approach [12]. In contrast to ontology-like SNs, such as WordNet [8] or GermaNet [9], which contain lexical and semantic relations between synsets (sets of synonyms), MultiNet is designed to comprehensively represent the semantics of a natural language expression, e.g., a sentence. In MultiNet, an SN consists of a set of nodes and edges. Nodes represent the concepts (word readings), and edges represent the relations between the concepts or functions, where concepts can be arguments or results. Examples of MultiNet relations are given below:

- *ALTN1: Function generating alternative pluralities of entities (or/and)
- *ALTN2: Function generating alternative pluralities of entities (exclusive or)
- ATTCH: Attachment of an object to another object
- SUB: Relation of conceptual subordination (hyponymy or instance relation)
- SUBR: Relation of conceptual subordination for relations
- SYNO: Synonymy relation

An example SN is shown in Figure 1. Nodes can be of two kinds: lexicalized nodes are associated to entries in the semantic lexicon, while nodes that represent complex situations or individual objects are not associated with single lexical entries. The latter nodes are just assigned a unique ID. To each concept, one or more ontological sorts, and a set of binary-valued semantic features is assigned. The ontological sorts (currently more than 40) form a taxonomy. In contrast to other taxonomies, ontological sorts are not necessarily lexicalized, i.e., they need not denote lexical entries. The following enumeration shows a small selection of ontological sorts, which are derived from *object* (*object* is directly derived from the root node and stands for a nominal concept.):

- Concrete objects: e.g., *milk*, *honey*
  - Discrete objects: e.g., *chair*
  - Substances: e.g., *milk*, *honey*
- Abstract objects: e.g., *race*, *robbery*

Semantic features denote certain semantic properties for objects. Such properties can either be present (+), not present (-) or underspecified. The concept *bee.1.1* is e.g. characterized by: discrete object, ANIMAL +, ANIMATE +, ARTIF -, HUMAN -, SPATIAL +, etc. Note that the suffix *.1.1* denotes the reading numbered .1.1 of the word *bee*. See [12] for details on the numbering scheme.

The SNs are constructed by the deep linguistic parser WOCADI[3] [11] for German text analysis. For parsing, WOCADI employs a word class functional analysis instead of a grammar. Aside from the SNs, WOCADI also creates a dependency tree and a so-called *token list*, which contains the tokens enriched with semantic and morphological information. Note that in some cases, the WOCADI parse may fail, particular if the sentence contains grammar errors, complex syntactic constructions or metonymy, which can violate semantic constraints.

4 Synonym Extraction

As we have seen, after running the WOCADI parser on a corpus (here the German[4] Wikipedia), its knowledge is available as a collection of SNs and token lists. In order to exploit the SNs and detect synonymy relations between concepts in the SNs and token lists, inference rules are employed, which we call *extraction rules*. Each extraction rule is given by a premise and a conclusion. In the case of synonym extraction, the conclusion is simply an underspecified synonymy relation, expressed as SYNO(*a1*, *a2*).

The premise part of a deep extraction rule formulates logical conditions over nodes and edges of an SN, and takes itself the form of an SN. That premise SN is tested to match against an instance SN like a template[5]. If a match is established, the instantiated conclusion (bindings are obtained from the matching premise) is extracted as synonymy hypothesis.

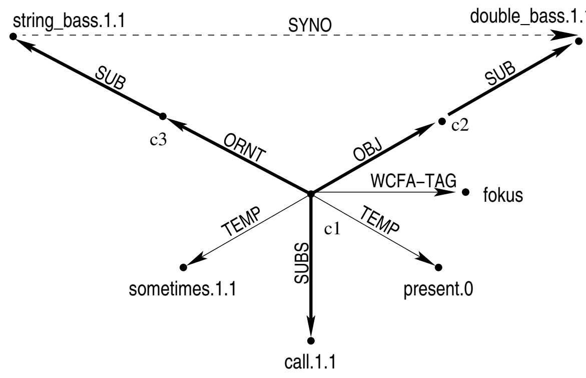

Figure 1: Extraction rule *DS1* (see Table 1) applied to the SN representing the sentence: 'A double bass is sometimes also called string bass.' Arcs matched with the rule premise are printed in bold. The dashed line denotes the inferred relation.

An example of an extraction rule application is illustrated in Figure 1. The SN[6] shown in this figure was created by the parser for the sentence 'A double bass is sometimes also called string bass.'. The node *c*1 represents the whole situation. SUBS(*c*1, *call.1.1*) denotes the fact that this specific situation is about calling, i.e., *c*1 is a troponym of *call.1.1*, which is expressed by the SUBS(*c*1, *call.1.1*) relation. The two objects (*double bass* and *string bass*) participating in this situation are connected by the relations OBJ (neutral object) and ORNT (orientation). The TEMP edge specifies the temporal embedding of this situation (*present.0* and *sometimes.1.1*).

In addition to deep extraction rules, we also employ shallow rules. In contrast to deep extraction rules, shallow rules are also applicable if a sentence is not parsable. The premise of such a rule is based on a regular expression of feature structures. The feature structures are tested to unify with the feature structures of the token list as provided by the parser. An entry for a single token contains (among others) the surface word form, the concept, the lemma and the grammatical category. The variables *a1* and *a2* of the conclusion may be matched to any concept of ontological type *object* (grammatical category: noun phrase). Some example shallow extraction rules are given below[7]:

- SYNO(*a1*, *a2*) $\leftarrow$ *a1*, in short *a2*
- SYNO(*a1*, *a2*) $\leftarrow$ *a1* '(' synonymous: *a2*
- SYNO(*a1*, *a2*) $\leftarrow$ *a1* is [cat art] abbreviation (of $\vee$ for) ?[cat art] *a2*

This section describes several advantages of deep extraction rules in contrast to shallow extraction rules. Consider the

[3] WOCADI is the abbreviation for **wo**rd **c**lass **di**sambiguating parser.

[4] Note that for better readability, all examples and extraction rules are translated from German to English.

[5] We call 'instance SN' the SN which resulted from the application of the WOCADI parser to surface text.

[6] Note that strictly speaking, the sentence is incorrectly considered as non-generic by the parser, which is expressed by the two relations SUB(*c2*, *double_bass.1.1*) and SUB(*c3*, *string_bass.1.1*).

[7] The full list of shallow extraction rules is given at `www.vdb1.de/syno.html`

DSi	Definition	Example
1	$\textsc{syno}(a1, a2) \leftarrow \textsc{arg1}(e, f)\wedge$ $\textsc{attch}(g, f) \wedge \textsc{subr}(e, sub.0)\wedge$ $\textsc{sub}(f, synonym.1.1)\wedge$ $\textsc{sub}(g, a1) \wedge \textsc{arg2}(e, h)\wedge$ $\textsc{sub}(h, a2) \wedge \textsc{sub}(h, f)\wedge$	Person is a synonym of human being.
2	$\textsc{syno}(a1, a2) \leftarrow \textsc{ornt}(e, h)\wedge$ $\textsc{sub}(h, a1)\wedge$ $\textsc{obj}(e, f) \wedge \textsc{sub}(f, a2)\wedge$ $\textsc{subs}(e, call.1.1)$	A laptop is also called notebook.
3	$\textsc{syno}(a1, a2) \leftarrow$ $PREC_{*\textsc{altn1}}(d, e)\wedge$ $\textsc{sub}(d, a1) \wedge \textsc{sub}(e, a2)$	A dog, hound, or canine denotes a certain type of animal.
4	$\textsc{syno}(a1, a2) \leftarrow \textsc{assoc}(e, f)\wedge$ $\textsc{sub}(f, a2) \wedge \textsc{subr}(e, prop.0)\wedge$ $\textsc{arg1}(e, g) \wedge \textsc{sub}(g, a1)\wedge$ $\textsc{arg2}(e, synonymous.1.1)$	laptop is synonymous to notebook.

Table 1: Deep extraction rules for synonym extraction. $PREC_f(x, y)$ denotes that x and y are arguments of function f and x directly precedes y in the argument list.

sentences: 'Notebook ist ein Synoynm zu Laptop.' and 'Notebook ist ein Synonym von Laptop' (Both sentences mean: 'Notebook is a synonym of Laptop'.) To extract the synonymy relation $\textsc{syno}(notebook.1.1, laptop.1.1)$ two different surface rules are required. Putting them together, the following disjunction can be used:

$$\textsc{syno}(a1, a2) \leftarrow a1 \text{ is a synonym (for} \vee \text{of)} \; a2 \qquad (1)$$

Also, two different dependency trees are created for the two sentences (tested by applying the Stanford Dependency Parser [19]), which also lead to two different extraction rules for the use of a dependency tree representation. This is not necessary if deep semantic rules are employed, since both surface representations are mapped to the same SN. The synonymy relation $\textsc{syno}(notebook.1.1, laptop.1.1)$ can be extracted from both sentences just by applying extraction rule *DS1*. Another example is given by the sentence pair:
1. 'The eelpout, which is also called Lycodichthys antarcticus, lived at the coasts of North Europe.'
2. 'The eelpout is also called Lycodichthys antarcticus.'
Again, the synonymy relation $\textsc{syno}(eelpout.1.1, lycodichthys\_antarcticus.1.1)$ can be extracted from both sentences by a single extraction rule (this time $DS2$), while two rules would be necessary for a dependency tree or surface representation.

In both cases, the recall was increased by the higher generality of deep extraction rules. Also, the use of a deep semantic representation in form of SNs can improve precision. Consider the two sentences:
'1. Mr. Schulz will travel to Cologne by either train or car.'
'2. An automobile or car denotes a machine with the following properties:... '
In the first sentence, the word 'or' is an exclusive-or, that means 'Mr. Schulz' will only take either one of the two means of transport but not both together. In the second sentence, the 'or' is meant in the sense that both concepts denote a car, i.e., the two concepts do not exclude each other. Therefore, the deep extraction rule ($*\textsc{altn1}$ combines concepts by an inclusive *or*) *DS3* extracts the synonymy relation $\textsc{syno}(automobile.1.1, car.1.1)$ but not the incorrect hypothesis $\textsc{syno}(train.1.1, car.1.1)$, while a shallow extraction rule would also create the incorrect hypothesis.

An example where the shallow pattern-based approach of Yu [25] (described in Section 2) would be misleading is: 'Aspirin (chemical formula is given in Fig.3), which was discovered by the French chemist C. F. Gerhardt and is also called acetylsalicylic acid, is a famous medicine against headache'.
In this case, even three noun phrases ('chemical formula', 'Fig. 3', 'French chemist C. F. Gerhardt') are located between the two synonyms, which would lead to the incorrect hypothesis $\textsc{syno}(french\_chemist\_c.\_f.\_gerhardt, acetylsalicylic\_acid)$. In contrast, the deep approach would actually extract the correct hypothesis $\textsc{syno}(aspirin.1.1, acetylsalicylic\_acid.1.1)$ by means of the extraction rule *DS2*.

5 Validation of Synonymy Hypotheses

Not all hypotheses generated by the extraction rules are correct. Therefore a two-step mechanism is devised consisting of a semantic-oriented filtering and an SVM-based validation.
Semantic-oriented Filtering: In MultiNet, a concept can be a meaning molecule[12]. In this case it consists of several facets. These facets are assigned different ontological sorts and semantic features. For instance, one facet of *school.1.1* denotes the group of people representing a school (i.e., teachers and students), another facet denotes the institution and a third the building. In order for two concepts to be synonymous, semantic features and ontological sorts have to coincide for all facets and the number of facets have to be identical.

For example:
- *house.1.1* (artif:+, animal:-) cannot be synonymous to *ape.1.1* (artif:-, animal:+)
- *house.1.1* (discrete object) cannot be synonymous to *water.1.1* (substance)
- *school.1.1* (3 facets) cannot be synonymous to *building.1.1* (1 facet)

If this condition is not fulfilled, the investigated concept pair is not stored in the hypotheses database.
SVM-based Validation: Synonymy hypotheses that pass the filter described in Section 5 are not necessarily correct. For this reason, a set of features is calculated, which are used to derive a confidence score by means of a support vector machine (SVM) [24].

Usually, the support vector optimization problem is solved by using the dual representation. For this, a similarity measure is required to compare the instances of the training corpus with each other. Traditionally, this similarity measure was given by the scalar product. In newer approaches this product was generalized to a kernel function, which makes it no longer necessary to create feature vectors for the given instances. Instead, the kernel function can directly compare the instances, which allows it to use SVM for classifying data that have no natural vector representations, such as trees, graphs, or data that have vector representations of different lengths like strings. We used a string and graph kernel as well as ordinary feature values.

Graph Kernel: A graph kernel is used to estimate the similarity of two graphs. In this scenario, the SNs are compared where the relation hypotheses were extracted from. The graph kernel used here estimates the similarity by counting the number of weighted common walks in the SNs. This procedure is based on the fact that graphs with many common walks are very similar. A walk is a generalization of a path. In contrast to a path, the same edge can be visited twice in one common walk. We allow common walks against the arc directions too. Otherwise, common walks could rarely connect both synonym nodes with each other. In the following, we give the employed graph kernel, which is a generalization of the graph kernel of vor der Brück and Helbig [23] to multi-label arcs that is itself based on the graph kernel of Gärtner et al[7].

The first step in this approach consists in determining the product graph $PG = (PV, PE)$ of the two so-called *factor graphs* (the SNs here). Each node of the product graph consists of a pair of nodes of the two factor graphs. The first component is a node of the first graph, the second component a node of the second graph and the labels of both nodes should be identical. In our SN scenario, a label la is '*anon(ymous)*' for a non-lexicalized node or the concept name otherwise.

$$(v_1, v_2) \in PV_{(G_1,G_2)} :\Leftrightarrow v_1 \in V_1 \wedge v_2 \in V_2 \wedge la(v_1) = la(v_2)$$

with $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. There is an arc between two node pairs in the product graph iff there exists an arc between the first components of the two pairs in the first factor graph and an arc between the second components of the second factor graph and both labels are identical.

For the SNs considered here this procedure is too limited since in MultiNet two nodes can be connected by several arcs in the same direction. An equivalent representation would be that only one arc is allowed to connect two nodes, but an arc is assigned a set of labels.

$$\begin{aligned} ((v_1, v_2), (w_1, w_2)) \in PE(G_1, G_2) :\Leftrightarrow \\ (v_1, w_1) \in E_1 \wedge (v_2, w_2) \in E_2 \wedge \\ la(v_1, w_1) \cap la(v_2, w_2) \neq \emptyset \end{aligned} \tag{2}$$

One arc in the product graph is a common walk of length one in the two factor graphs. The adjacency matrix is defined as:

$$\begin{aligned} &A_{PG} = (a_{ij})_{i=1\ldots n, j=1\ldots n} \text{ with} \\ &a_{ij} := \begin{cases} 1, (u_i, u_j) \in PE \vee (u_j, u_i) \in PE \\ 0, \text{otherwise} \end{cases} \end{aligned}$$

$(u_i, u_j) \in PE \vee (u_j, u_i) \in PE$ denotes the fact that the common walks can also be followed against the arc directions. This is a variation of the original method of Gärtner et al. An entry i, j of the adjacency matrix is one (1) if there exists a common walk of length one between node i and j. Analogously, there exist l common walks of length m between the associated node components i and j, iff $A^m_{PG}(i, j) = l$. The similarity between two graphs G_1, G_2 is then estimated by:

$$K(G_1, G_2) = \iota^\top (\sum_{k=0}^{\infty} \lambda^k A^k_{PG})\iota \tag{3}$$

where

- $A^0_{PG} = I$ the identity matrix
- $\iota := (1 \ldots 1)^\top$ is a vector which consists only of ones and is used to sum up the matrix components $\sum_{k=0}^{\infty} \lambda^k A^k_{PG}$.
- $\iota^\top$ denotes the transposed vector to ι.
- $\lambda < 1$ is a constant which is chosen in such a way that the sum converges. Since $\lambda < 1$, the influence of long common walks is less than for short ones. However, since a long common walk always implies many short common walks, the total influence of long common walks is still larger than for short ones.

Furthermore, the node label with two components of the synonymy hypotheses are replaced by fixed variables (*a1* and *a2*), which makes two graphs with different hypotheses better comparable.

There is one drawback of the method as described above. All common walks are treated the same, regardless of whether a common walk actually passes the synonym candidate nodes. Therefore, the original approach of Gärtner et al. is modified in such a way that the following two kernels are determined:

- the graph kernel of the common walks passing at least one of the synonym nodes
- the graph kernel of the common walks passing both of the synonym nodes

Actually, it is not possible to determine the number of common walks that pass (not necessarily start or end at) certain nodes with this method directly. However, the opposite is quite easy. The number of common walks that do not pass certain nodes can be determined just by removing all connections from and to such nodes from the adjacency matrix. Afterwards, the ordinary common walk algorithm can be applied.

Thus, we can determine the graph kernel of common walks that pass at least one of *a1* and *a2* by subtracting the graph kernel of all common walks that pass neither one from the graph kernel of all common walks.

$$\underset{a1 \vee a2}{K}(G_1, G_2) = K(G_1, G_2) - \underset{\neg(a1 \vee a2)}{K}(G_1, G_2) \tag{4}$$

The graph kernel for the common walks that pass *a1* as well as *a2* can be determined similarly:

$$\begin{aligned} \underset{a1 \wedge a2}{K}(G_1, G_2) = &K(G_1, G_2) - \underset{\neg a1}{K}(G_1, G_2) - \\ &\underset{\neg a2}{K}(G_1, G_2) + \underset{\neg(a1 \vee a2)}{K}(G_1, G_2) \end{aligned} \tag{5}$$

Note that $K_{\neg(a1 \vee a2)}(G_1, G_2)$ was added to the right side of Equation 5 since otherwise the graph kernel for the common walks that pass neither *a1* nor *a2* would be subtracted twice.

String Kernel: The graph kernel function can only be applied if the parse was successful for the sentences where the hypothesis candidates were extracted from. This is not always the case. Therefore, we apply a string-kernel in addition. The string kernel is applied on the concepts of the token list as returned by WOCADI. If the parse was not successful, a disambiguation of the readings is usually not done. In this case, the reading is selected that appears most often in the corpus. To improve the generality, the synonym candidates are replaced by fixed variables (here *a1* and *a2*). Parenthesis expressions are removed if neither of the two synonym candidates appear inside them. This is done to increase the performance as well as to improve the similarity value, since such expressions are usually not used to express the synonymy relation in any way.

We selected the common subsequence string kernel, which counts the weighted number of common subsequences of two strings (here concept lists) w and v [22, 412–414].

$$K_n(w,v) = \sum_{u \in C^n} \sum_{\mathbf{a}:u=w[\mathbf{a}]} \sum_{\mathbf{b}:u=v[\mathbf{b}]} \lambda^{l_w(a)+l_v(b)} \qquad (6)$$

with:

- λ: a weighting factor with $\lambda < 1$
- $\mathbf{a} \in \mathbb{N}^n$: a vector consisting of sorted indices that reference components (here concepts) in w
- $\mathbf{b} \in \mathbb{N}^n$: a vector consisting of sorted indices that reference components in v
- $l_s : \mathbb{N}^n \to \mathbb{N}$, $l_s(a) := a[n] - a[1]$: function that returns the length of the covered interval in the given string
- C: here set of all concepts

Gaps in the matching process are allowed but are penalized by increasing the exponent of λ. This exponent is not given by the length of the subsequence but instead by the length of the covered area in the two concept lists. In this way, a small subsequence can still lead to a large exponent if there are many or large gaps in the two concept lists containing components that cannot be matched. The total string kernel is given by: $K(w,v) = \sum_{i=1}^{\min\{|w|,|v|\}} K_i(w,v)$

The Graph kernel, string kernel and a radial basis function, which is applied on a set of features, are combined by a weighted sum. The optimal weight setting is determined by a grid search over the interval [0,1]. We used features to test if one concept label could be an abbreviation of the others, if both concepts can occur in similar semantic network contexts, if both contexts were written similar and if the relation hypotheses were extracted by both deep and shallow patterns. Additionally one binary feature is used for every pattern which is set to one if the hypotheses was extracted by this feature and to zero otherwise.

6 Results

	PNS	PS	Σ
NS	297	1428	1725
S	10	167	177
Σ	307	1595	1902

Table 2: Confusion matrix for the semantic-oriented filter (see Sect. 5). PS=Predicted synonym, PNS=Predicted non-synonym, NS=non-synonym, S=synonym.

	GSK −		GK +		GSK +	
	PNS	PS	PNS	PS	PNS	PS
NS	324	376	507	193	637	63
S	67	633	104	596	113	587
Σ	391	1009	611	789	750	650

Table 3: Confusion matrix for the validation of synonyms. GSK −=without graph / string-kernel, GK +=with graph kernel, GSK +=with graph / string kernel.

Our synonym extraction algorithm (called SemQuire) was applied on the entire German Wikipedia corpus of 2009. The entire Wikipedia was parsed by the deep analyzer WOCADI [11] and converted into SNs and token lists. The rule application approach described in the previous sections were applied on this output. In total, 265 938 synonymy hypotheses were stored in the database, 19 269 of them only originating from deep patterns.

Measure	Filter	GSK −	GK +	GSK +
Accuracy	0.244	0.684	0.788	0.874
F-measure	0.189	0.741	0.801	0.870
Precision	0.105	0.627	0.755	0.903
Recall	0.944	0.904	0.851	0.839

Table 4: Accuracy, F-measure, precision, and recall for the validation of synonyms.

The semantic-oriented filter (see Sect. 5) was tested on 1902 randomly selected hypotheses, which were annotated by human annotators with either 1 (synonymy relation actually present) or 0 (synonymy relation not present), by a 10-fold cross-validation. The confusion matrix is given in Table 2, the evaluation measures in Table 4. Precision is the relative frequency with which a predicted synonym is actually one, while accuracy is the relative frequency with which the decision (synonym/non-synonym) is correct. The evaluation shows that the recall is very high, which means the number of false negatives is very small. This demonstrates that indeed this classifier is very useful as a filter. Currently, the name and abbreviation lexicons of HaGenLex are not yet used, which would result in higher precision values.

The SVM-based validation was tested on 1 500 annotated hypotheses (accuracy: 0.844, F-measure: 0.637, precision: 0.701, recall: 0.583) by a 10-fold cross-validation. Note that this recall only reflects the validation but not the extraction of the hypotheses. We also compared our approach with several ontology-based semantic similarity measures on the same evaluation set. These measures exploit hyponymy and already known synonymy relations. The ontology consists of GermaNet, Wiktionary and HaGenLex. Since instance relations are considered as hyponymy relations in GermaNet and Wiktionary, these measures can be applied to proper names (such as 'Germany' or 'UK') as well. The coverage of names by these resources is rather poor, which leads to quite low F-measures (Leackock-Chodorov: 0.0243, Lin: 0.0227, Resnik: 0.0391 [16, 17, 20]). Context based measures perform better. The information radius reaches an F-measure of 0.136. For classification of the similarity measures described above we used a threshold. All similarity values below this threshold are mapped to 0 (hypothesis incorrect), all above to 1 (hypothesis correct). The threshold is chosen in such a way that the F-measure is maximized. Furthermore, we re-implemented the approach of Yu [25] (the reimplementation of the method described in [26] is not finished) and applied it on the Wikipedia, too, where all patterns were translated into German. In total, this approach extracted 11 638 hypotheses, with an estimated precision of 0.347 (determined on an annotated subset), which leads to estimated 4 038 correct hypotheses. In contrast, the estimated number of correct hypotheses of SemQuire is 36 165. Also the precision of SemQuire (0.701) is much superior to that of Yu's method (0.347).

Note that the F-measure is highly dependent on the applied rules, i.e., the use of unreliable but often applicable rules leads to a rather low (validation) F-measure, whereas the exclusive use of reliable but less applicable rules results in a high F-measure. Therefore, we conducted a second experiment where the amount of positive and negative examples are equal (700 positive and 700 negative), which avoids such variations in F-measure. The resulting confusion matrix is given in Table 3 and accuracy, F-measure, precision, and validation recall in Table 4. Three different experiments were done: employing only shallow features (GSK −), employing shallow features and graph kernel (GK +), and employing shallow features as well as graph and string kernel (GSK +). The evaluation results were considerably superior to the baseline (accuracy: 0.5), which is just the approach to opt for hypothesis correctness in all cases. Furthermore, the use of a graph and string kernel leads to a significant improvement in the evaluation measures (e.g., increase of 0.13 for F-measure). The increase of accuracy and precision is significant with a level of 1%.

The weights of features, string kernel and graph kernel, as determined by the grid search, which were used to combine the kernels, took the following values: Features: 0.0, Graph kernel: 1.0, String kernel: 1.0. These weights show that if graph and string kernels are employed, the use of the features is actually not necessary. However, this does not mean the features are completely useless since they are much faster to compute. For the selected test and training set, a full parse was available for 548 of the hypotheses, only a chunk parse for 607 hypotheses, and the parse failed for the remaining 245. This shows the potential of the graph kernel method. Although a full parse was available in the minority of cases, the graph kernel was assigned a weight identical to the string kernel.

7 Discussion and Outlook

An approach called SemQuire was introduced for extracting synonyms employing a deep semantic representation. Instead of using only the surface representation of sentences, the patterns are defined on a semantic level and are applied on SNs.

In contrast to a shallow representation, the semantic patterns have the advantage of a greater generality, which reduces the number of patterns. Also, by usage of graph and string kernel, evaluation results were improved significantly. Furthermore, SemQuire obtained quite superior results in contrast to several context and ontology-based semantic similarity measures and the approach of Yu.

For future work, we plan to learn patterns and additional semantic features automatically. Furthermore, graph kernels based on common walks as proposed in this paper usually suffer from tottering, which can be avoided if the same arc is prevented from being visited two times in a row [6]. Also the use of the extracted synonymy relations in real-world applications is of interest.

ACKNOWLEDGEMENTS

Especially, we want to thank Prof. Helbig for his support for this work.

REFERENCES

[1] C. Bunescu et al., ‘A shortest path dependency kernel for relation extraction’, in *Proc. of HLT/EMNLP*, pp. 724–731, Vancouver, Canada, (2005).

[2] A. Culotta and J. Sorenson, ‘Dependency tree kernels for relation extraction’, in *Proc. of ACL*, pp. 423–429, Barcelona, Spain, (2004).

[3] P. Edmonds, ‘Choosing the word most typical in context using a lexical co-occurrence network’, in *Proc. of ACL*, pp. 507–509, Madrid, Spain, (1997).

[4] C. Biemann et al., ‘Automatic acquisition of paradigmatic relations using iterated co-occurrences’, in *Proc. of LREC*, Lisbon, Portugal, (2004).

[5] F. Reichartz et al., ‘Dependency tree kernels for relation extraction from natural language text’, in *Proc. of ECML*, pp. 270–285, Bled, Slovenia, (2009).

[6] P. Mahé et al, ‘Extensions of marginalized graph kernels’, in *Proceedings of ICML*, Banff, Canada, (2004).

[7] T. Gärtner et al., ‘On graph kernels: Hardness results and efficient alternatives’, in *Proc. of COLT*, pp. 129–143, Washington, DC, (2003).

[8] *WordNet An Electronic Lexical Database*, ed., C. Fellbaum, MIT Press, Cambridge, Massachusetts, 1998.

[9] B. Hamp and H. Feldweg, ‘GermaNet - a lexical-semantic net for german’, in *Proc. of the ACL workshop Automatic IE and Building of Lexical Semantic Resources for NLP Applications*, pp. 9–15, Madrid, Spain, (1997).

[10] Z. Harris, *Mathematical Structures of Language*, J. Wiley & Sons, New York, 1968.

[11] S. Hartrumpf, *Hybrid Disambiguation in Natural Language Analysis*, Ph.D. dissertation, FernUniversität in Hagen, Fachbereich Informatik, Hagen, Germany, 2002.

[12] H. Helbig, *Knowledge Representation and the Semantics of Natural Language*, Springer, Heidelberg, Germany, 2006.

[13] D. Hindle, ‘Noun classification from predicate-argument structures’, in *Proc. of ACL*, pp. 268–275, Pittsburgh, Pennsylvania, (1990).

[14] J. Jiang and D. Conrath, ‘Semantic similarity based on corpus statistics and lexical taxonomy’, in *Proc. of ROCLING*, pp. 19–33, Taipei, Taiwan, (1997).

[15] S. Kullback and R. Leibler, ‘On information and sufficiency’, *Annals of Mathematical Statistics*, **22**(1), 79–86, (1951).

[16] C. Leacock and M. Chodorow, ‘Combining local context and WordNet similarity for word sense identification’, in *WordNet. An Electronic Lexical Database*, 265–283, MIT Press, Cambridge, Massachusetts, (1998).

[17] D. Lin, ‘Principle-based parsing without overgeneration’, in *Proc. of the Workshop of Computational Terminology*, pp. 57–64, Montreal, Canada, (1993).

[18] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*, MIT Press, Cambridge, Massachusetts, 1999.

[19] M.de Marneffe and C. D. Manning, *Stanford Typed Dependencies Manual*, 2008. online at: `http://nlp.stanford.edu/software/dependencies_manual.pdf`.

[20] P. Resnik, ‘Using information content to evaluate semantic similarity in a taxonomy’, in *Proc. of IJCAI*, pp. 448–453, Montréal, Canada, (1995).

[21] G. Ruge, ‘Automatic detection of thesaurus relations for information retrieval applications’, in *LNCS 1337*, 499–506, Springer, Heidelberg, Germany, (1997).

[22] B. Schölkopf and A. Smola, *Learning with Kernels - Support Vector Machines, Regularization, Optimization and Beyond*, MIT Press, Cambridge, Massachusetts, 2002.

[23] H. Helbig T.vor der Brück, ‘Validating meronymy hypotheses with SVMs and graph kernels’, in *Proc. of ICMLA*, pp. 243–250, Washington, DC, (2010). IEEE Press.

[24] V. Vapnik, *Statistical Learning Theory*, John Wiley & Sons, New York, New York, 1998.

[25] H. Yu, ‘Automatic extraction from scientific abstracts of synonyms for proteins and genes’, in *Proc. of AMIA*, (2001).

[26] H. Yu et al., ‘Automatic extraction of gene and protein synonyms from medline and journal articles’, in *Proc. of AMIA*, (2002).

[27] S. Zhao and R. Grishman, ‘Extracting relations with integrated information using kernel methods’, in *Proc. of ACL*, pp. 419–426, Ann Arbor, Michigan, (2005).