Inflection-Tolerant Ontology-Based Named Entity Recognition for Real-Time Applications

Christian Jilek
German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany
Department of Computer Science, TU Kaiserslautern, Germany
christian.jilek@dfki.de

Markus Schröder
German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany
Department of Computer Science, TU Kaiserslautern, Germany
markus.schroeder@dfki.de

Rudolf Novik
German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany
rudolf.novik@dfki.de

Sven Schwarz
German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany
sven.schwarz@dfki.de

Heiko Maus
German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany
heiko.maus@dfki.de

Andreas Dengel
German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany
Department of Computer Science, TU Kaiserslautern, Germany
andreas.dengel@dfki.de

Abstract

A growing number of applications users daily interact with have to operate in (near) real-time: chatbots, digital companions, knowledge work support systems – just to name a few. To perform the services desired by the user, these systems have to analyze user activity logs or explicit user input extremely fast. In particular, text content (e.g. in form of text snippets) needs to be processed in an information extraction task. Regarding the aforementioned temporal requirements, this has to be accomplished in just a few milliseconds, which limits the number of methods that can be applied. Practically, only very fast methods remain, which on the other hand deliver worse results than slower but more sophisticated Natural Language Processing (NLP) pipelines.

In this paper, we investigate and propose methods for real-time capable Named Entity Recognition (NER). As a first improvement step, we address word variations induced by inflection, for example present in the German language. Our approach is ontology-based and makes use of several language information sources like Wiktionary. We evaluated it using the German Wikipedia (about 9.4B characters), for which the whole NER process took considerably less than an hour. Since precision and recall are higher than with comparably fast methods, we conclude that the quality gap between high speed methods and sophisticated NLP pipelines can be narrowed a bit more without losing real-time capable runtime performance.

2012 ACM Subject Classification Computing methodologies → Information extraction; Computing methodologies → Semantic networks

Keywords and phrases Ontology-based information extraction, Named entity recognition, Inflectional languages, Real-time systems

Digital Object Identifier 10.4230/OASIcs.LDK.2019.11

2nd Conference on Language, Data and Knowledge (LDK 2019).
Editors: Maria Eskevich, Gerard de Melo, Christian Fäth, John P. McCrae, Paul Buitelaar, Christian Chiarcos, Bettina Klimek, and Milan Dojchinovski; Article No. 11; pp. 11:1–11:14

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – DE 420/19-1.

Acknowledgements We thank Sven Hertling, Jörn Hees, Erfan Shamabadi, Oleksii Kotvytskyi and Tim Sprengart for their contributions in this project's early and late phase, respectively.

1 Introduction

The number of application areas, in which users are supported by systems that operate in (near) real-time, grows: chatbots, digital companions, knowledge work support systems – just to name a few. Our targeted scenario involves a system based on Semantic Desktop [15] technology, that semi-automatically re-organizes itself based on user context [10] in order to better support knowledge work and information management activities[1]. We envision an intelligent, proactive assistance parallel to the actual work. Such systems need mechanisms to analyze observed user activities (entering text, browsing a website, reading/writing files, ...) in order to decide on the right support measures and perform them accordingly. The demand for very short reaction times limits the number of methods that can be applied.

In this paper, we focus on Information Extraction (IE) methods, more precisely Named Entity Recognition (NER), that are ontology-based (our system operates on knowledge graphs in the background) and meet the demand for providing meaningful results within only a few milliseconds on users' typical computing devices. By *only a few* we actually mean a small two-digit number of milliseconds. According to Miller (1968) and Card et al. (1991), as cited in [13], 100 ms is "about the limit for having the user feel that the system is reacting instantaneously" and 1000 ms is "about the limit for the user's flow of thought to stay uninterrupted". Our goal is to stay below the first value. In cases, in which this is not possible (e.g. too much data to be processed at once), 1000 ms should be the upper bound of processing time to be tolerated. Since we also need some time for selecting and performing the support measures, the IE task has to be completed within only a fraction of this time span. Such strict temporal requirements usually rule out very sophisticated Natural Language Processing (NLP) pipelines (higher quality solutions but slow), leaving only rather simple (lower quality) but very fast methods often based on pre-defined rules or gazetteers. A gazetteer is conceptually just a list of terms (typically static), that the input text is later scanned for, e.g. the names of persons, organizations or locations. Since our scenario also involves highly inflectional languages like German[2], we additionally have to take slight variations of such terms into account. To vividly illustrate the problem of inflections in NER, we fed the first paragraph of the German Wikipedia article of *Propositional calculus* (German: *Aussagenlogik*) to *DBpedia Spotlight*[3] [11], a well-known and often used recognizer for Wikipedia/DBpedia[4] entities in given text snippets. The results are depicted in Figure 1 (middle section): Twelve entities (in just three sentences; we highlighted them in yellow) are not found, ten of them due to lexical variations induced by inflection. E.g. *Wahrheitswert* (*truth value*) is found, whereas its inflected forms ending with *-e* and *-en* are not. If we lower the confidence to 0.0, there are still some entities missing and false positives come up.

In summary, our goal is to find or implement methods that are fast enough to meet the aforementioned temporal constraints while at the same time achieving better results than standard high speed methods. Recognizing entities despite the just mentioned lexical

[1] for an overview and more details please see `https://comem.ai/`
[2] other inflectional languages: Spanish, Latin, Hebrew, Hindi, Slavic languages, ...
[3] `https://www.dbpedia-spotlight.org/demo/`
[4] `https://wiki.dbpedia.org/`

Figure 1 First paragraph of the German Wikipedia article of *Aussagenlogik* (top) fed to DBpedia Spotlight [11] using confidence values of 0.5 (middle) and 0.0 (bottom). (highlighting we applied: green: existing Wikipedia articles not linked in the original document, yellow: false negatives, red: false positives.)

variations induced by inflection would be a first improvement step. Note that disambiguation as well as recognizing Named Entities (NE) yet unknown to the system (i.e. not available as instances in the knowledge graph) are out of this paper's scope. Since there is a lot of explicitated contextual information available in our system, we intend to address disambiguation in our scenario in a future paper.

The rest of this paper is structured as follows: Section 2 provides an overview of related work in this area. Our approach is described in Section 3 and its evaluation is presented in 4. In Section 5, we conclude this paper and give a an outlook on possible future work.

2 Related Work

We were looking for approaches (more or less) explicitly addressing inflection tolerance or real-time capability, preferably both at the same time:

Concerning real-time capability, Dlugolinsky et al. [8] present an overview of different gazetteer-based approaches, especially referring to various versions included in the GATE (General Architecture for Text Engineering) framework [6]. They distinguish between

character- and token-based variants and state that the latter usually have "longer running time and low processing performance". They thus focus on character-based gazetteers and present several versions [8, 12]. Since some of their implementations are available online, we also included them in our evaluation (see Section 4).

Savary & Piskorski [17] investigated solutions for Polish, also a highly inflectional language. As one subcomponent of their IE platform *SProUT* they filled a gazetteer by "explicitly listing all inflected forms of each entry".

Day & Prukayastha [7] gave an overview of different NER methods especially targeting Indian languages. Their paper presented gazetteer-based and machine learning approaches as well as hybrid solutions.

Al-Jumaily et al. [3] present an NER system for Arabic text mining. They use a token-based approach involving stemming as well as pre- and postfix verification tailored to the Arabic language. Although they aim for real-time applications, they do not give any details about their system's runtime performance.

Al-Rfou & Skiena [4] propose *SpeedRead*, an NER pipeline which they tested to run ten times faster than the *Stanford CoreNLP* pipeline[5]. Unfortunately for us, they only reported runtime performance in terms of tokens per second. In their final results, they say SpeedRead achieves about 153 tokens/sec. Using the word length statistics published by Norvig [14] and assuming an average token length of about five characters, we would end up having 765 char./sec, which is still much too slow for our scenario as we will later see. Even if we assume an average token length of twelve, although more than 90% of all English words are shorter [14], we would still be too slow having 1836 char./sec.

In summary, we found several approaches dealing with either real-time capability or inflection tolerance. One paper even mentioned both, but did not report any concrete speed measures. Nevertheless, doing NER extremely fast is apparently rarely discussed in literature, yet. This may be because usual NER methods operate in only a few seconds, which may be sufficient for many use cases, unfortunately not ours.

We will refer to some of the ideas discussed in this section when presenting our approach in the following.

3 Approach

We focus on the very fast recognition of NEs given as instance labels of ontologies. Moreover, these labels should still be recognized even if they slightly lexically vary as induced by inflection. To achieve this, we exploit knowledge graphs connected or available to our system such as an individual user's Personal Information Model (PIMO) [16] or DBpedia to get more details about the entities, e.g. their specific type. Based on this type, we can then accept different lexical variations per instance according to language information coming from Wiktionary[6], for example. For instance, we should not allow too many variations of person names, whereas we can be more tolerant when dealing with topic, project, organization or location names, especially if they contain adjectives like the *Technical* University of Kaiserslautern or *German* Research Center for *Artificial* Intelligence. As an example, Figure 3 shows all 18 inflected forms of *künstlich* (*artificial*) in German (word `w4` in the figure).

As depicted in Figure 2, we have a hierarchical NE recognizer as the core of our system. It operates on several sub-recognizers, mostly Multi-Layer Finite-State Transducers (MLFST) as described later, each of them having a different focus (configuration). The core recognizer

[5] `https://stanfordnlp.github.io/CoreNLP/`
[6] `https://www.wiktionary.org/`

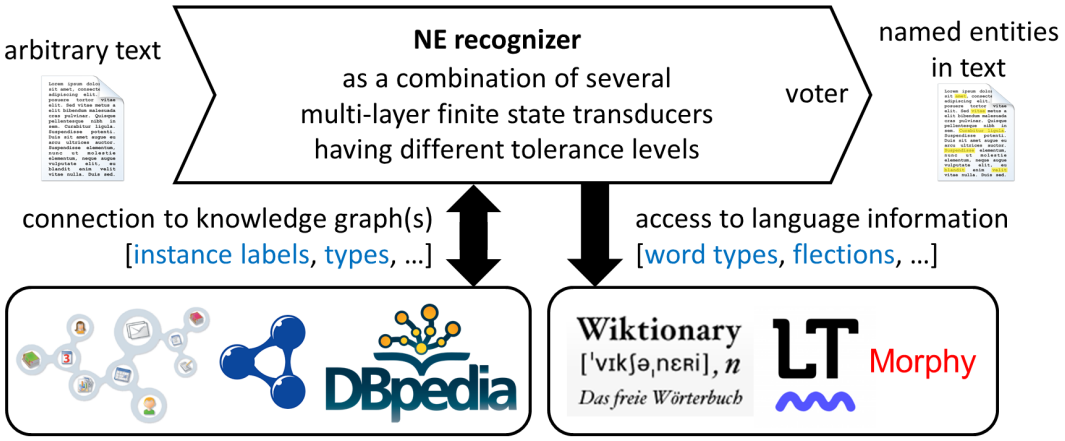

Figure 2 Architecture of our system.

collects their results and decides (votes) which ones to accept. To acquire the entity labels as well as background information, it is connected to knowledge graphs and language information sources as described before. Its individual aspects are discussed in more detail in the following.

3.1 FST-based NER

To meet the aforementioned strict runtime requirements, we basically follow a gazetteer-based approach. The additionally required inflection tolerance is not well compatible with the usually static character of a gazetteer. We thus need enhancements as described in the following.

Our core method is based on the well-known string matching algorithm by Aho & Corasick [2]. It operates on *tries*, i.e. trees whose nodes represent characters, which are traversed synchronously to the processing of each character of the input text. Whenever the traversal ends in an accepting state, there is a string match. Since, in our case, these strings are the labels of NEs, we additionally demand that their ID or URI is returned, which makes the system a Finite-State Transducer (FST). The algorithm basically has linear runtime complexity as discussed later. Our scenario involves a highly dynamic, evolving knowledge graph, in which instances (and especially their labels) can be added, deleted or updated potentially several times per minute. We thus omitted further optimizations like suffix compression in favor of a fast and easy to update FST structure.

3.2 Multi-Layer FST

For runtime performance reasons we decided against sophisticated NLP pipelines (test results and more details in Section 4) and therefore follow the approach of explicitly listing all inflected forms of an entity label as proposed in [17]. Without further ado, this would easily lead to memory performance problems due to a considerable increase of the FST, especially for multi-word terms: The more words such a multi-word term consists of, the more potential combinations exist. Although inflection tolerance is discussed more thoroughly in the paragraph after next, let us just consider a short example here: If we allow each combination of inflected forms of the term *Deutsches Forschungszentrum für Künstliche Intelligenz* (*German Research Center for Artificial Intelligence*, shortly referred to as DFKI), although lots of them are grammatically not correct (as also discussed later), we would end

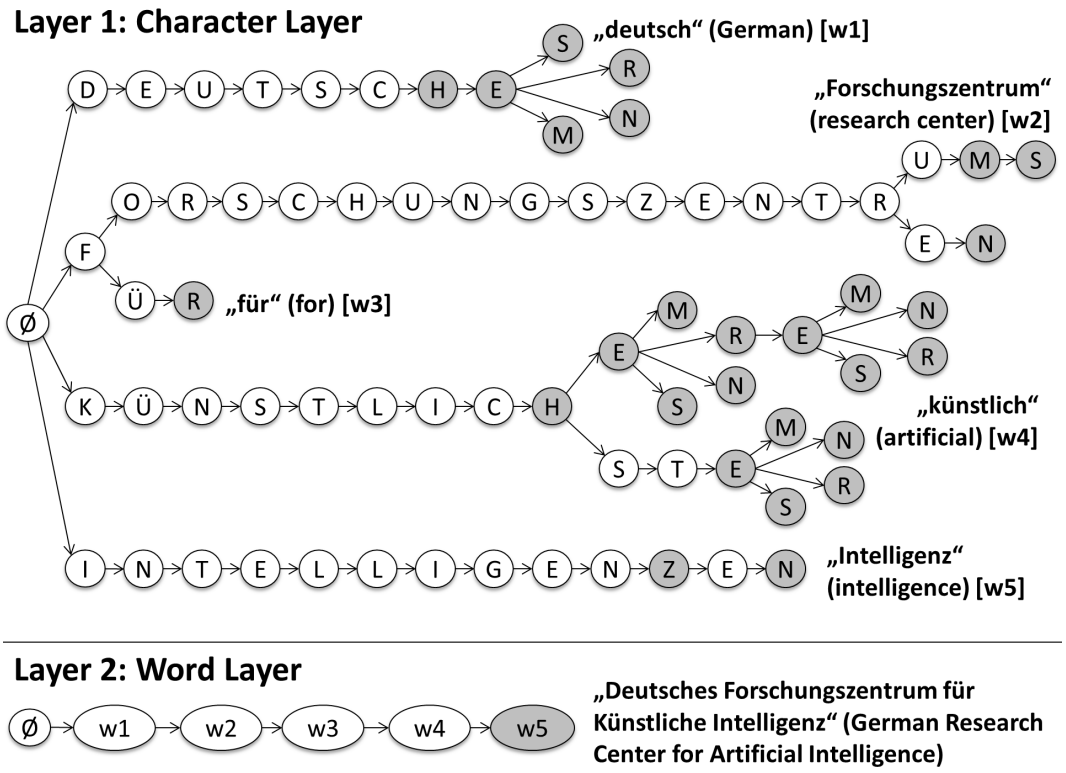

Figure 3 Multi-layer finite transducer consisting of a character and a word layer, and fed with the term *Deutsches Forschungszentrum für Künstliche Intelligenz* (∅: start nodes; w_i: word IDs; gray nodes: accepting states).

up with 576 variations ($= 6 \cdot 3 \cdot 1 \cdot 16 \cdot 2$; see upper part of Figure 3). Inspired by Abney, who proposed the idea of *finite-state cascades* [1], we therefore chose to introduce an additional layer to separate character from word processing, making our system a multi-layer FST as illustrated in Figure 3: Once a word is identified in the first layer (i.e. the FST is in an accepting state; gray node), its ID is passed to the second layer, which checks whether this word may be accepted at this position, either as a single-word or part of a multi-word term. If a term match is detected, its ID/URI is returned. As a consequence, each word and its inflected forms, no matter how often or at which positions (in multi-word terms) they appear, only exist once in the FST, thus preventing it from growing too fast in size.

To avoid backtracking in the word layer, the system processes several options in parallel as shown in Figure 4: Once the character layer recognizes a word, e.g. `w6`, a new word node processor in the second layer is spawned (see upper left part of the figure; purple color). If layer 1 then reports the next word `w7` (highlighted in green), processor 1 goes one step further in the graph now having a traversed path containing both words. Additionally, another processor is spawned, starting directly with `w7`. For this behavior, we use the metaphor of a rake (if you merge all start nodes into a single one, you get the image): Spawning another processor is like adding another tine to the rake. Traversals in the word layer are only possible if the next detected word is a successor of the current one within any term of the FST, which, for example, is not the case when processor 2 tries to handle `w9`, or processor 4 tries to start directly with `w9`. The latter means that there is no term in the

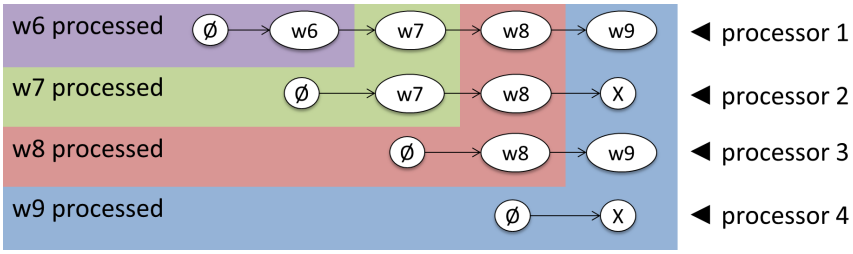

Figure 4 Processing in the word layer: several processors operate in parallel. Their traversed paths are depicted (∅: start nodes; w_i: word IDs; X: failure states).

FST starting with the word `w9`. These two processors are then in a failure state (indicated by "X"). If there was a matching term in their traversed path, it is collected to be later processed by the voter. If that is not the case, the failed processors may be removed from the rake. The second case in which processors are removed, whether they are in a failure state or not, is after an explicit signal from the first layer, e.g. when reaching the end of a file or sentence. Spawning additional processors to evaluate different possibilities in parallel especially originated from the latter. Consider the case of interpreting a dot: It could either indicate the end of a sentence (*"Today, I met my Prof."*), or an abbreviation (*"Prof. Smith was also there."*). Thus, there is a forking in the second layer to evaluate both possibilities separately. In theory, this could lead to endless forking, which is prevented by processors reaching failure states (i.e. given word sequences not matching any term) followed by their removal. The basic steps of our algorithm are given as pseudocode (see Algorithm 1).

3.3 Real-Time Capability

Reading an input text of length n characterwise yields a basic runtime complexity of $\mathcal{O}(n)$. The same is true for processing n characters in the first layer (at most n transitions having a constant amount of operations; no backtracking needed). The processing of a character may lead to the detection of a new word, which then triggers transitions in the word layer. The number of these transitions depends on the number p of processors ("tines in the rake"). p does not depend on n, but on the vocabulary, i.e. all words fed to the FST, especially $w_{\max}$, the maximum number of words in all multi-word terms. Although $p_{\max}$ is constant for a given vocabulary, it may still be very large in worst case[7]. In practical scenarios however, $p \ll p_{\max}$ can be assumed, since the vocabulary is only a tiny fraction of the power set of its words. As a consequence, processors fail very fast due to given word combinations not matching any term in the FST. Considering an additional constant amount of $c > 0$ operations per processor in each transition of the second layer yields an upper limit of $c \cdot p_{\max} \cdot n$. Since n is thus only multiplied with constants, the overall runtime complexity remains $\mathcal{O}(n)$. Although the second layer's overhead is noticeable in practice (as we will see in Section 4), the overall runtime complexity is still linear and benefits our system's applicability in scenarios of real-time processing.

[7] In worst case, a term consisting of $w_{\max}$ words is read, whereas each subterm also exists in the vocabulary. Moreover, for each of these subterms there is an additional variant ending with a dot. This leads to forking after every word and a total amount of $p_{\max} = \sum_{i=1}^{w_{\max}} 2^i$ processors before the first one of them fails and is removed.

Algorithm 1: Basic steps of our MLFST-based NER algorithm in pseudocode.

```
input  : text to process (text)
output : found entities (foundEntities)
foundEntities ← { };
collectedTerms ← { };
c ← first character of text;
w ← c;
while c not equals EOF (end of file or text snippet) do
    if c is whitespace character then
        if w matches in character layer then
            add new word node processor (in word layer);
            for all word node processors p do
                process w with p (may either lead to word match or failure state in p);
            end
        end
        collectedTerms ← collectedTerms ∪ collect matching terms from word layer;
        remove word node processors in failure state (word layer);
        w ← ∅;
    else
        w ← w + c;
    end
    c ← read next character of text (character layer);
end
collectedTerms ← collectedTerms ∪ collect matching terms from word layer;
foundEntities ← do voting on collectedTerms (word layer);
return foundEntities;
```

3.4 Inflection Tolerance

As mentioned before, to accept different lexical variations of terms, e.g. induced by inflection, we utilize information coming from connected ontologies as well as other language information sources. Concerning the latter, we use a lemmatization table extracted from *LanguageTool*[8], an open source proofreading software for several languages, which itself contains binary files of *Morfologik* to look up part-of-speech data. Such entries look as follows:

```
künstlich     künstlich   ADJ:PRD:GRU
künstliche    künstlich   ADJ:AKK:PLU:FEM:GRU:SOL
künstlichem   künstlich   ADJ:DAT:SIN:MAS:GRU:SOL
...
```

They contain the inflected form, its lemma as well as declension information like word class, case, number, gender, etc. We additionally used *Wiktionary*, a free wiki-based dictionary, whose data[9] we extracted using *DKpro JWKTL*[10] [18]. Nevertheless, there were still lots of words not covered by any of these sources, especially compound words like *Forschungszentrum*

[8] `https://github.com/languagetool-org` (uses `https://github.com/morfologik`)
[9] `https://dumps.wikimedia.org/` (dump file of 2016-07-01)
[10] `https://dkpro.github.io/dkpro-jwktl/`

(*research center*). To counter this, we additionally implemented heuristics like longest suffix matching to decompound words and use the inflected forms of the last part (if available). In the case of *Forschungszentrum* not being in our database, the heuristic would first look for the word *orschungszentrum* (fails), then *rschungszentrum* (fails), *schungszentrum* (fails), etc., until finally finding *zentrum* and using its inflected forms, i.e. *Zentrum*, *Zentrums* and *Zentren*. The matching part of the original word is then replaced with these inflected forms as shown in Figure 3. The heuristic additionally expects a parameter indicating the minimum length of the remaining suffix (e.g. five characters) to receive more meaningful results. Our tool is thus able to handle yet unknown words to a certain extent without user interaction. In this regard, let us revisit the aforementioned 576 variations of the term DFKI. As also mentioned, most of them are grammatically not correct. Since we also want to handle yet unknown words, especially compound ones, while keeping the user interaction as low as possible (not asking for feedback), we decided to accept all variations obtained as the Cartesian product of all inflected forms of each of a term's words. We assume that grammatically wrong variants do rarely occur in given texts and if they do, users will agree with the entity being recognized despite the misspelling. Nevertheless, the question remains whether this decision considerably increases the false positive rate. We will address this in Section 4. To avoid actually harmful false positives of incorrectly inflected variants, we exploit additional ontological information like the type of an entity. For example, the name of a person tolerates far less variants than the name of a topic. Basically, we only allow a possessive/genitive case "s" at the end, like stated before. As a consequence, our NE recognizer is actually not just a single MLFST, but a combination of several ones each having a different configuration. Currently, there is one having higher and another one having lower tolerance. The latter, for example, contains person names. There is also an option to especially deal with acronyms: They do not only require exact matches, moreover all characters need to be uppercase. To further avoid non-meaningful variants, we only use adjective and noun information from the lemmatization table, which reduces ambiguities when not having thorough NLP information. This is a compromise we can accept, since labels more often contain nouns and adjectives than verbs.

When processing input text, the different MLFST operate in parallel. In the end, a voter receives, assesses, filters and finally returns their results. Additionally, each MLFST has its own internal voter which assesses all results simultaneously present in a processing rake. In the current implementation, these voters follow a strategy of only keeping the longest match, e.g. if the term *personal information management* is found in the text, the also matching terms of *personal information* and *information management* would be discarded.

4 Evaluation

4.1 Setting

Besides finding out how fast our NE recognizer performs in practice, we were especially interested whether our design decisions (see Section 3) would lead to a considerable increase in false positives. We were thus looking for large amounts of German natural language texts (prose) written by different people to test our approach. The German Wikipedia meets this requirement but lacks ground truth data for the inflected forms present in these texts. We therefore decided to only look at the wikilinks (see Figure 1, top section, blue words) and take them as a silver standard: A human has annotated terms in the text (often in inflected form) with the label of their respective Wikipedia article (typically in basic form). Figure 1 also shows that users themselves decide which terms they annotate: There are lots of entities

(highlighted in green), which are not annotated although there is a Wikipedia article for them. This is especially true for self-references, e.g. the term *Aussagenlogik* is not annotated in "its own" article (i.e. the one about *Aussagenlogik*). Recognizers fed with such terms, would nevertheless find them, which has to be considered when measuring precision.

Regardless of possible shortcuts, annotations are structured as follows: the term appearing in the text and the name of the Wikipedia article it refers to (in the following also shortly called *the link*) are written in double brackets separated by a pipe symbol, e.g. `[[Häuser|Haus]]` (plural form of *house* appears in the text, whereas the article is labeled with the singular form). Since inflection usually just changes one to four characters, the Levenshtein distance (LD) between term and link can help us identifying samples we could use to evaluate our approach. Note that independent term-link-combinations like `[hometown|Eton]` or adjective-noun-combinations like `[entscheidbar|Entscheidbarkeit]`(*decidable/decidability*) are undesirably also covered by such an LD-based heuristic. On the other hand, this evaluation approach offers millions of inflection samples (we ran our tests on 3.9M articles having 50.4M annotations).

We downloaded German Wikipedia dump files[11] and used 3.9M article names as a basis for feeding our recognizers. Disambiguation information in brackets like in *"Berlin (Russland)"* (a village in Russia sharing its name with the German capital) were removed (this raises disambiguation issues as discussed later). We also removed number-, symbol- and single-character-only labels, since they were not relevant for our investigations. As ontological background information we used types[12] coming from DBpedia, which were available for about 0.5M entities. For types like person, city, film, etc., we applied a low tolerance strategy (i.e. possessive/genitive case "s" is the only accepted variation), whereas all other ones were treated with higher tolerance.

4.2 Evaluated Named Entity Recognizers

We evaluated our MLFST approach against three baseline methods. The first and most obvious one, *StemFST*, was also implemented by us and uses the MLFST's character layer combined with the *Lucene*[13] *German Stemmer*, which is based on [5]. The other methods are the previously mentioned ones by Dlugolinsky et al. [8], who made two of their gazetteers available online[14]: one based on hash-map multi-way trees (*HMT*), and the other based on first child-next sibling binary trees (*CST*). Both produce the same results in terms of found NEs, but differ in memory consumption and runtime performance.

After filtering and editing as mentioned in the previous paragraph, we had slightly above 3.3M article names of the German Wikipedia that we fed to all four NE recognizers. HMT and CST take these terms without further changes. StemFST splits each term into words and reassembles it after stemming them. Then it adds the altered term to its FST. MLFST does the same but instead of stemming the words, it looks up (or tries to infer) their inflected forms. Completely filled, the inner high-tolerance MLFST contained 8.5M character nodes and 3.5M word nodes, the low-tolerance part kept 1M and 0.4M nodes, respectively.

[11] `https://dumps.wikimedia.org/` (dump file of 2016-11-01)
[12] `https://downloads.dbpedia.org/3.9/de/instance_types_de.ttl.bz2`
[13] `https://lucene.apache.org/`
[14] `http://ikt.ui.sav.sk/gazetteer/`

4.3 Results

All computations were performed on a notebook having an Intel Core i7-4910MQ 2.9 GHz CPU and 16 GB RAM, running on Windows 7 (64-bit).

HMT only needed 10.4 min for processing 3.9M articles (9.4B characters), while the others needed 31.0 to 47.7 min (see Figure 5). Figure 6 shows that HMT trades memory efficiency for speed, since it is the only recognizer passing the 1 GiB mark by needing 3.5 GiB. The others needed 0.72 to 0.96 GiB.

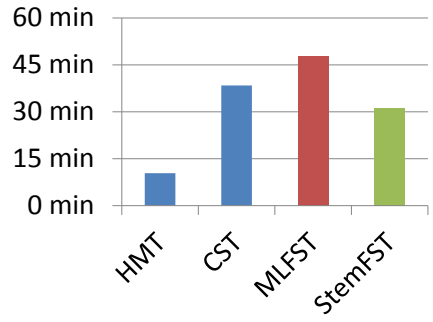

Figure 5 Processing time.

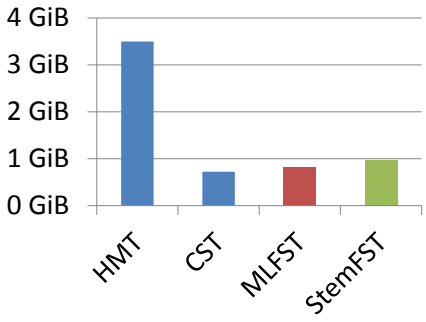

Figure 6 Memory usage.

4.3.1 Recall

Let us next consider recall: All recognizers reached values slightly below or above 70%. Figure 8 additionally shows the results itemized by LD. If term and link match exactly (`LD=0`, which is the case for 69% of all annotations), all recognition rates are above 92%[15]. In LD ranges of `LD=1` to `LD=4` (11% of all annotations), HMT/CST's recall is close to 0%, whereas MLFST still has rates of 79%, 66%, 36% and 8%, respectively. StemFST even has slightly higher rates. Reaching recall near 100% should not be expected, since not all variations are caused by inflection and their number decreases with increasing LD. For higher LD values (`LD>4`, 21% of all annotations), all recognition rates are close to 0%.

4.3.2 Precision

Concerning precision, we already mentioned the problem of how to measure it adequately. We decided to calculate multiple values: P_O measures precision only for terms *overlapping* with annotation positions, because only there we have "ground truth" data. As shown in Figure 7, some found terms (purple highlighting) are not exactly matching the actual annotation (blue word, highlighted in green as the only true positive). If terms are overlapping with the annotation, we count them as a false positive. P_A counts *all* terms not exactly

Figure 7 Example sentence to illustrate the different precision values.

[15] errors in the dump and imperfect parsing caused a slight decrease (100% expected)

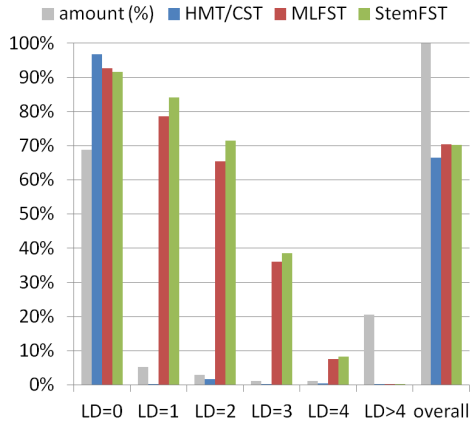

Figure 8 Recall itemized by Levenshtein distance of term and link.

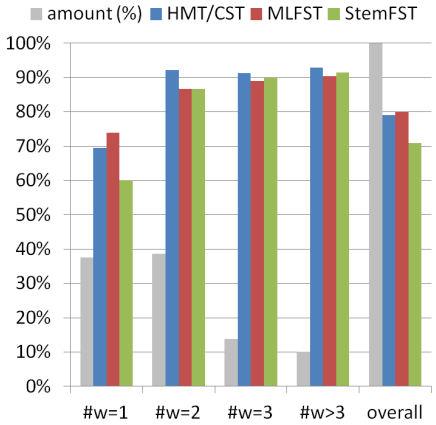

Figure 9 Precision P_O^* itemized by the terms' number of words (#w).

matching as false positives, especially also the non-overlapping ones (red highlighting). Since disambiguation was out of this paper's scope and there are labels belonging to more than 1000 instances (e.g. *Jewish cemetery*), it makes a large difference whether or not we additionally count more than 1000 false positives for each true positive in a text. We thus introduce P_O^* and P_A^*, which count multiple entities having the same label only once. P_O^* is 79% for HMT/CST and 80% for MLFST, while StemFST only reaches 71%. Figure 9 additionally depicts P_O^* itemized by the number of words a term consists of. For multi-word terms, all approaches achieve values between 87% and 92%. There is a remarkable difference for single word terms: Here, stemming seems to be too rough causing terms to lose their specifity and StemFST to lose 14% compared to MLFST, which performs best having 74%. The other overall precision values P_O, P_A and P_A^* are shown in Figure 10. They are far lower than P_O^* due to the aforementioned reasons. However, in a short experiment, in which students annotated some randomly chosen articles manually, we observed values for P_A^* that were similar to P_O^* above. We thus have a slight indication that P_A^* (depicted above) heavily underestimates our algorithm's precision. Finally answering one of our initial research questions: the false positive rate of MLFST is not considerably higher (in some cases even lower) than with the other recognizers.

4.3.3 Runtime Performance

Regarding runtime performance, MLFST and StemFST process between 3281 and 5048 characters per millisecond and are thus comparable to CST as illustrated in Figure 11. HMT is about three times faster at the expense of memory consumption (see Figure 6). All tested recognizers are by orders of magnitude faster than basic NLP pipelines. We tested OpenNLP[16] and CoreNLP using a basic pipeline consisting only of a tokenizer, sentence splitter and part-of-speech tagger. Although no NER-specific analyzers like noun chunkers or type classifiers were added yet, their processing time was already out of our targeted range. Running the basic pipeline on all 3.9M articles would presumably have taken about 18 days in the case of CoreNLP, for example.

[16] `https://opennlp.apache.org/`

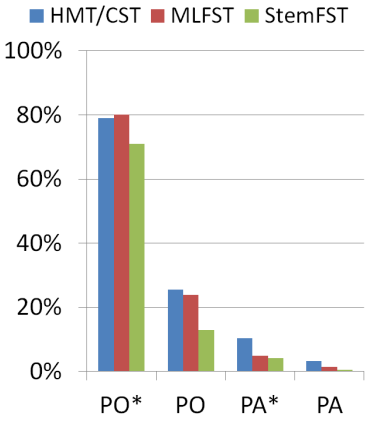

Figure 10 Precision: P^*_O, P_O, P^*_A, P_A.

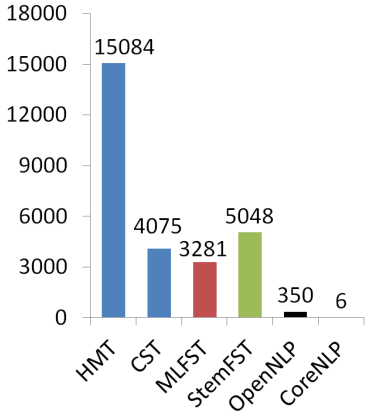

Figure 11 Processed characters per ms.

5 Conclusion & Outlook

In this paper, we presented an ontology-based NER approach that is comparably fast as available high speed methods while outperforming them in the recognition of terms that lexically vary slightly, e.g. induced by inflection. We were thus able to narrow the quality gap to more sophisticated but also much slower NLP pipelines a bit more without losing real-time capable runtime performance.

In the future, we plan to additionally incorporate StemFST into MLFST, since its recall was slightly better for multi-word terms. Additionally, we could add more layers scanning for patterns like phrases that indicate todos or appointments, Hearst patterns [9], etc. There is also much potential for improving the language capabilities of our approach, e.g. improved rules and heuristics (e.g. to infer inflections) or multi-language support. Last but not least, we plan to incorporate disambiguation mechanisms by exploiting the explicated user context available in our system.

References

1 Steven Abney. Partial parsing via finite-state cascades. *Natural Language Engineering*, 2(4):337–344, 1996.

2 Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

3 Harith Al-Jumaily, Paloma Martínez, José L. Martínez-Fernández, and Erik Van der Goot. A real time Named Entity Recognition system for Arabic text mining. *Language Resources and Evaluation*, 46(4):543–563, 2012.

4 Rami Al-Rfou and Steven Skiena. SpeedRead: A Fast Named Entity Recognition Pipeline. *Proceedings 24th International Conference on Computational Linguistics (COLING 2012)*, pages 51–66, 2012.

5 Jörg Caumanns. A fast and simple stemming algorithm for German words. Technical Report TR B 99-16, Center für Digitale Systeme, Freie Universität Berlin, 1999.

6 Hamish Cunningham, Valentin Tablan, Angus Roberts, and Kalina Bontcheva. Getting more out of biomedical documents with GATE's full lifecycle open source text analytics. *PLoS computational biology*, 9(2):e1002854, 2013.

7 Arindam Dey and Bipul Syam Prukayastha. Named Entity Recognition using Gazetteer Method and N-gram Technique for an Inflectional Language: A Hybrid Approach. *International Journal of Computer Applications*, 84(9), 2013.

8 Stefan Dlugolinskỳ, Giang Nguyen, Michal Laclavík, and Martin Šeleng. Character gazetteer for Named Entity Recognition with linear matching complexity. In *3rd World Congress on Information and Communication Technologies (WICT)*, pages 361–365. IEEE, 2013.

9 Marti A. Hearst. Automatic Acquisition of Hyponyms from Large Text Corpora. In *Proceedings of the 14th Conference on Computational Linguistics - Vol. 2*, pages 539–545. Association for Computational Linguistics, 1992.

10 Christian Jilek, Markus Schröder, Sven Schwarz, Heiko Maus, and Andreas Dengel. Context Spaces as the Cornerstone of a Near-Transparent and Self-Reorganizing Semantic Desktop. In *The Semantic Web: ESWC 2018 Satellite Events*, pages 89–94. Springer, 2018.

11 Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. DBpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems (I-Semantics)*, pages 1–8. ACM, 2011.

12 Giang Nguyen, Štefan Dlugolinskỳ, Michal Laclavík, Martin Šeleng, and Viet Tran. Next Improvement Towards Linear Named Entity Recognition Using Character Gazetteers. In *Advanced Computational Methods for Knowledge Engineering*, pages 255–265. Springer, 2014.

13 Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann, 1993.

14 Peter Norvig. English Letter Frequency Counts: Mayzner Revisited or ETAOIN SRHLDCU, 2013. accessed: 2018-08-18. URL: `http://norvig.com/mayzner.html`.

15 Leo Sauermann, Ansgar Bernardi, and Andreas Dengel. Overview and Outlook on the Semantic Desktop. In *Proceedings of the 1st Workshop on the Semantic Desktop at the ISWC 2005 Conference*, pages 74–91. CEUR-WS, 2005.

16 Leo Sauermann, Ludger van Elst, and Andreas Dengel. PIMO – a framework for representing personal information models. In *Proceedings of I-Media '07 and I-Semantics '07*, pages 270–277. Know-Center, Austria, 2007.

17 Agata Savary and Jakub Piskorski. Lexicons and grammars for named entity annotation in the National corpus of Polish. In *18th International Conference Intelligent Information Systems*, pages 141–154, 2010.

18 Torsten Zesch, Christof Müller, and Iryna Gurevych. Extracting Lexical Semantic Knowledge from Wikipedia and Wiktionary. In *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC)*, pages 1646–1652, 2008.