Hybrid Reasoning Over Large Knowledge Bases Using On-The-Fly Knowledge Extraction

Giorgos Stoilos[*]$^{[0000-0002-9633-347X]}$, Damir Juric, Szymon Wartak, Claudia Schulz$^{[0000-0002-2569-7065]}$, and Mohammad Khodadadi

Babylon Health, London, SW3 3DD, UK
{firstname.lastname}@babylonhealth.com

Abstract. The success of logic-based methods for comparing entities heavily depends on the axioms that have been described for them in the Knowledge Base (KB). Due to the incompleteness of even large and well engineered KBs, such methods suffer from low recall when applied in real-world use cases. To address this, we designed a reasoning framework that combines logic-based subsumption with statistical methods for on-the-fly knowledge extraction. Statistical methods extract additional (missing) axioms for the compared entities with the goal of tackling the incompleteness of KBs and thus improving recall. Although this can be beneficial, it can also introduce noise (false positives or false negatives). Hence, our framework uses heuristics to assess whether knowledge extraction is likely to be advantageous and only activates the statistical components if this is the case. We instantiate our framework by combining lightweight logic-based reasoning implemented on top of existing triple-stores with an axiom extraction method that is based on the labels of concepts. Our work was motivated by industrial use cases over which we evaluate our instantiated framework, showing that it outperforms approaches that are only based on textual information. Besides the best combination of precision and recall, our implementation is also scalable and is currently used in an industrial production environment.

Keywords: Large Medical Ontologies · Axiom Extraction from Text · Hybrid Reasoning

1 Introduction

Large Knowledge Bases (KBs) have started to play a key role in applications like dialogue systems [29], healthcare [4], and recommendation systems [20]. KBs describe the entities of the domain at hand and their relationships. This enables semantic interpretation of information expressed in terms of knowledge from a KB, and thus allows the exchange of information between different systems or components expressing information using a shared KB.

To capture real-world meaning, systems may not only use atomic concepts from the KB but also combine concepts to express more complex entities of the

[*] Since submission Giorgos Stoilos is a member of Huawei Technologies, R&D, UK

domain. For example, in a biomedical application using SNOMED CT to represent medical data, a user profile may contain the concept $C_1 := \mathsf{RecentInjury} \sqcap \exists \mathsf{findingSite.Head}$, capturing the condition "recent injury in the head" associated with the patient. Yet, in a symptom-checking service the same condition may be represented as $C_2 := \mathsf{HeadInjury} \sqcap \exists \mathsf{occurred.Recent}$. To enable proper information exchange between these services, *subsumption reasoning* over the stored knowledge is crucial to correctly match equivalent information expressed in different ways, i.e. to prove that $\mathcal{K} \models C_1 \sqsubseteq C_2$ and $\mathcal{K} \models C_2 \sqsubseteq C_1$ for a KB $\mathcal{K}$.

In theory, ontology reasoners like ELK [15] can be used for this kind of subsumption checking, however, in practice this is problematic for at least two reasons. First, even large and well-engineered KBs suffer from *incompleteness* [21, 10], leading to very low recall of equivalent and subsumed concepts. Indeed, for the above example and for $\mathcal{K}$ the SNOMED CT, we have $\mathcal{K} \not\models C_1 \sqsubseteq C_2$ and $\mathcal{K} \not\models C_2 \sqsubseteq C_1$ since $\mathcal{K}$ is missing an axiom of the form $ax := \mathsf{RecentInjury} \equiv \mathsf{Injury} \sqcap \exists \mathsf{occurred.Recent}$. SNOMED CT contains many such ill-defined concepts [21] like SevereDepression, which is not defined in terms of concepts Severe and Depression, or CardiacMuscleThickness, not defined in terms of CardiacMuscle and Thick. The same can be observed for other KBs like DBpedia [1], where the category ItalianRenaissancePainters is not connected to concepts Painter or ItalianRenaissance. Second, such reasoners are not designed for reasoning over very large industrial KBs, like DBpedia and Freebase, which are usually stored in triple-stores or graph databases.

To address the above issues, we designed a *hybrid reasoning framework* for subsumption checking that couples logic-based reasoning with on-the-fly knowledge extraction. Knowledge extraction is used to enrich the concepts being compared with intended but missing axioms and hence increase recall of subsumption checking. Since knowledge extraction can produce false positives, i.e. lead to subsumptions that are not intended, the framework incorporates heuristics to decide when to apply knowledge extraction.

We give a concrete instantiation of our framework, where the knowledge extraction component is realised using Natural Language Processing techniques that construct (complex) concepts from the *labels* of the compared concepts. For instance, in our running example, the label of concept RecentInjury in SNOMED CT is "Recent Injury". From that, our knowledge extraction component constructs the more detailed concept $\mathsf{Injury} \sqcap \exists \mathsf{occurred.Recent}$, thus recovering the missing axiom ax and eventually enabling to prove the subsumption since $\mathcal{K} \cup \{ax\} \models C_1 \sqsubseteq C_2$. Furthermore, to allow reasoning over large industrial KBs we realise logic-based reasoning via a lightweight approximate reasoner implemented on top of existing triple-stores. Finally, we present concrete instantiations of the heuristics controlling the knowledge extraction component.

Importantly, our knowledge extraction method not only forms a crucial part of the hybrid reasoning framework, but it can also be applied as a stand-alone method for constructing complex concepts from short phrases and can thus be useful, e.g., for keyword-based and entity-centric query answering on top of KBs [23, 14, 13]. This furthermore enables us to apply our hybrid reasoner to textual inputs (short phrases), which can be transformed into concepts using the stand-alone knowledge extraction component, followed by the normal application of our hybrid reasoner.

Our work was motivated by several industrial use cases in Babylon Health[1], a digital health care provider offering services such as AI-based symptom-checking and triaging. Data in all services are encoded using (complex) concepts built from a large medical KB that is stored in a triple-store. It is desirable that different services exchange and compare concepts for the purposes of interoperability, ensuring the delivery of intelligent services to end-users. For example, if new evidence for a "head injury" is encoded in the user profile in the form of concept C_1, other services like triaging or symptom-checking need to be able to interpret C_1 and compare it to concepts used by these services, e.g. C_2, to provide advice to the patient on how to proceed. We tested our implementation in two use cases within this industrial healthcare setting. Our results show the advantage of our knowledge extraction method used internally in our hybrid subsumption checking reasoner compared to pure logic-based reasoning, as well as the high quality of concepts constructed by the extraction method. Furthermore, results reveal that our hybrid reasoner provides the best combination of precision and recall compared to approaches that are purely statistical or purely rule-based.

2 Preliminaries

The concept extraction method we design will construct concepts expressed in the Description Logic (DL) $\mathcal{EL}$ [2], that is, concepts that are defined by the grammar $C := \top \mid A \mid C_1 \sqcap C_2 \mid \exists R.C$ where A is an *atomic concept* and R is an *atomic property*. In addition to the standard DL notation, we assume that every atomic concept A (resp. property R) has an associated label denoted by $\mathscr{L}(A)$ (resp. $\mathscr{L}(R)$). $\mathcal{EL}$-concepts can also be written as $\bigsqcap_i A_i \sqcap \bigsqcap_j \exists R_j.C_j$ where each A_i is an atomic concept or $\top$ and each C_j is again an $\mathcal{EL}$-concept. We also consider concepts like $\exists R.B$ to be of the form $\top \sqcap \exists R.B$.

$\mathcal{EL}$ is expressive enough to capture many medical KBs like SNOMED CT. In fact, as noted in [8], the SNOMED CT KB is *primitive*, that is, it can be rewritten as a set of subsumption axioms of the form $A \sqsubseteq D$ where A is atomic. This is because full definitions of the form $A \equiv D$ are acyclic and therefore can be recursively unfolded into axioms of the form $B \sqsubseteq \exists R.A$ to obtain $B \sqsubseteq \exists R.D$ and then discard $A \equiv D$.

The reasoning problem investigated in our work is *subsumption checking* between two concepts C and D of the above form with respect to a KB $\mathcal{K}$—that is, whether $\mathcal{K} \models C \sqsubseteq D$. However, acknowledging the fact that KBs are incomplete [21, 10] the problem we study is whether a set of "missing" axioms $\mathcal{K}'$ can be extracted such that $\mathcal{K} \cup \mathcal{K}' \models C \sqsubseteq D$. In order not to modify the KB during subsumption checking, the axioms in $\mathcal{K}'$ can be immediately unfolded into C and D, obtaining new concepts C' and D'. We thus reformulate the original axiom addition problem as a concept rewriting problem and check if $\mathcal{K} \models C' \sqsubseteq D'$.

[1] https://www.babylonhealth.com/

3 A Framework for Hybrid Reasoning

In theory, given two concepts standard reasoning techniques can be used to compare them w.r.t. subsumption. However, the success of this task heavily depends on the knowledge described in the KB for every concept and, as already pointed out in the literature [21], a lot of relevant knowledge is usually missing.

Example 1. Consider the SNOMED CT KB $\mathcal{K}$ and the phrase "recent head injury", which different services may represent in the following different ways:
$C_1 := \mathsf{RecentInjury} \sqcap \exists \mathsf{findingSite}.\mathsf{Head}$
$C_2 := \mathsf{Injury} \sqcap \exists \mathsf{findingSite}.\mathsf{Head} \sqcap \exists \mathsf{occurred}.\mathsf{Recently}$
It can be verified using any DL reasoner that $\mathcal{K} \not\models C_1 \sqsubseteq C_2$ and $\mathcal{K} \not\models C_2 \sqsubseteq C_1$ since $\mathcal{K}$ is missing an axiom of the form $\mathsf{RecentInjury} \equiv \mathsf{Injury} \sqcap \exists \mathsf{occurred}.\mathsf{Recently}$.

The above example highlights the need for a method that tries to construct such missing but intended axioms for any two concepts to enable higher recall of traditional subsumption checking algorithms. Looking again at the example, we note that the missing knowledge is in fact encoded in the *label* of the concept $\mathsf{RecentInjury}$, which is "recent injury". More precisely, using this label, our goal is to construct an axiom $ax := \mathsf{RecentInjury} \equiv \mathsf{Injury} \sqcap \exists \mathsf{occurred}.\mathsf{Recently}$ and then directly unfold it into C_1 for subsumption checking, yielding $C_1' := \mathsf{Injury} \sqcap \exists \mathsf{occurred}.\mathsf{Recently} \sqcap \exists \mathsf{findingSite}.\mathsf{Head}$. We would then have $\mathcal{K} \models C_1' \sqsubseteq C_2$ and $\mathcal{K} \models C_2 \sqsubseteq C_1'$, implying the equivalence of C_1 and C_2 (given the missing axiom).

Although axiom extraction and concept unfolding can improve recall of subsumption checking, they may also introduce false positives (i.e. subsumption of concepts that should be unrelated) or even false negatives (i.e. failure of proving subsumption for concepts that are subsumed without unfolding).

Example 2. Assume that for the concepts in Example 1, we also use the label of concept Injury to unfold C_2, which is "Traumatic AND/OR non-traumatic injury". A concept extraction method is likely to construct an axiom $ax' := \mathsf{Injury} \equiv \mathsf{Injury} \sqcap \exists \mathsf{assocWith}.\ \mathsf{Traumatic} \sqcap \exists \mathsf{assocWith}.\mathsf{NonTraumatic}$ from this. Then, for C_2' the unfolding of ax' in C_2, we will have $\mathcal{K} \not\models C_1' \sqsubseteq C_2'$.

Based on the above, we designed a *hybrid reasoning framework*, given in Algorithm 1, which combines logic-based subsumption checking with on-the-fly statistical knowledge extraction. Given a candidate subsumption $C \sqsubseteq D$, the algorithm first attempts to use standard logic-based reasoning (line 1) to prove subsumption over a KB $\mathcal{K}$. If subsumption cannot be proven, the knowledge extraction method may be activated, subject to some applicability conditions. The *first condition* (sim) assesses if the similarity of the compared concepts C and D is sufficiently high (line 2). Intuitively, the higher the similarity, the more likely it is for the axiom extraction step to lead to true positive results.

If the first condition is satisfied, the question is which of the two concepts should be extended with additional knowledge. Due to the monotonicity of the OWL semantics, adding axioms for C would be valuable since failure to prove subsumption implies that D likely has "more constraints" (conjuncts) than C.

Algorithm 1 $\mathsf{isSubsumed}_{\mathcal{K}}(C, D)$

```
    Input: Concepts of the form $C := \bigsqcap_i A_i \sqcap \bigsqcap_j \exists R_j.E_j$ and $D := \bigsqcap_k B_k \sqcap \bigsqcap_l \exists S_l.F_l$
 1: if $\mathcal{K} \models C \sqsubseteq D$ then return true
 2: if $\mathsf{sim}(C, D) < \sigma_1$ then return false
 3: $C_{ext} := C$
 4: if $|\mathsf{diff}(C, D)| > \sigma_2$ then
 5:     $C_{ext} := \bigsqcap_i \mathsf{constructConcepts}(A_i, \mathcal{K}) \sqcap \bigsqcap_j \exists R_j.E_j$
 6:     if $\mathcal{K} \models C_{ext} \sqsubseteq D$ then return true
 7: end if
 8: $D_{ext} := \bigsqcap_k \mathsf{constructConcepts}(B_k, \mathcal{K}) \sqcap \bigsqcap_l \exists S_l.F_l$
 9: if $\mathcal{K} \models C_{ext} \sqsubseteq D_{ext}$ then return true
10: return false
```

These additional axioms could then make up for the missing constraints. The *second applicability condition* (diff) thus checks whether D has more constraints than C. If so, the knowledge extraction component is activated to unfold C with an axiom obtained by processing its main atomic concepts (line 5). If the unfolded C is still not subsumed by D, D is also unfolded (line 8).

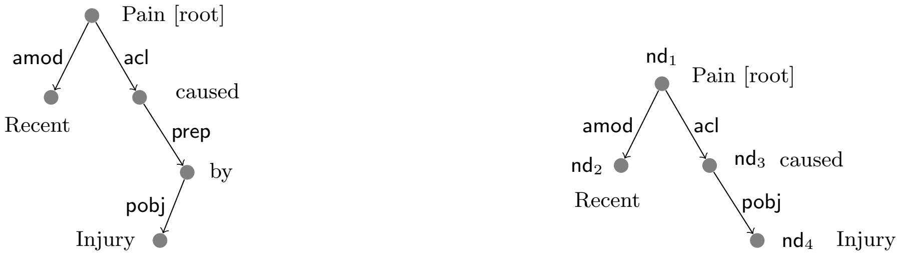
There are various things to note about our hybrid reasoning framework. First, extraction is only applied on the main atomic concepts (i.e. the A_i and B_k) as these represent the gist of a complex concept. Applying knowledge extraction on more parts of a complex concept would induce further fuzziness, thus increasing the probability for false positives and false negatives. For the same reason, our hybrid reasoner applies knowledge extraction *on-the-fly* instead of extracting all possible axioms from concept labels before reasoning, which would lead to the unrestricted usage of extracted axioms. For small KBs, extracting axioms upfront could allow manual validation, thus providing an advantage over on-the-fly extraction. However for large-scale industrial KBs as investigated here, this is infeasible since there is no way to reliably validate thousands of axioms. Thus, on-the-fly extraction is more beneficial for such KBs.

The exact implementation used for logic-based subsumption checking and knowledge extraction is up to the user and depends on the application at hand. In the following sections, we present concrete instantiations of all parts, targeted at knowledge extraction from *concept labels* and reasoning over *triple-stores*.

4 Extracting Concepts From Text

Our concept extraction method (i.e., the implementation of constructConcepts in Algorithm 1) will be based on the *labels* of concepts. Labels have been used in the past for tasks like ontology enrichment [9, 21] as they constitute a good source of additional knowledge that is usually in abundance in ontologies.

The problem of constructing concepts from their labels can be broken down into two parts, as further detailed in the following sections: 1) parsing the label to correctly identify its parts, and 2) linking each part to atomic concepts in the KB and selecting appropriate KB properties to piece them together.



(a) Dependency Tree (b) Normalised Tree

Fig. 1. Dependency and normalised trees of "recent pain caused by injury".

4.1 Parsing the Phrase Structure

Concept labels usually follow a specific pattern, that is, they contain a *central entity*, frequently narrowed down by *modifiers*. For example, in the SNOMED CT concept label "pain in left leg" the central entity is "pain", which is further detailed by "leg", which in turn is narrowed down via modifier "left". Examples of concept labels in other KBs are "Italian Renaissance painters" or "Thriller Movie". Such phrases may include linguistic constructs like prepositions, e.g., "Pain **in** Leg" versus "Leg Pain", and can in some rare cases contain verbs, e.g., "central sleep apnea **caused by** high altitude". Finally, KBs seldom contain concepts representing conjunctions or disjunctions of atomic entities, hence their labels rarely include coordinating conjunctions like "pain in arm **and** chest" and never contain non-$\mathcal{EL}$ constructors like "at least" and "for all".

The above observations motivate the use of dependency parsing [16], which precisely attempts to capture the main word in a phrase (called *root*) and then recursively append the rest of the modifiers to it. At an abstract level, the *dependency tree* of a phrase *txt* can be characterised as a tree where each node nd is labelled with a corresponding word from *txt*, denoted $\mathsf{nd}.\ell$, and each edge $\langle \mathsf{nd}_1, \mathsf{nd}_2 \rangle$ is labelled with a *dependency relation* $\langle \mathsf{nd}_1, \mathsf{nd}_2 \rangle.\ell$ that denotes the linguistic relation between the nodes $\mathsf{nd}_1.\ell$ and $\mathsf{nd}_2.\ell$.

Figure 1a depicts the dependency tree of the phrase "recent pain caused by injury" obtained using ClearNLP [6]. Since a dependency parser constructs a node for every word in the phrase, compound concepts like "heart attack" are split into separate nodes. However, grouping them is beneficial for better interpreting meaning [27]. Moreover, since prepositions do not carry concept meaning, we prune prepositions and other functional words. In summary, the dependency tree is post-processed as follows:

($\Diamond$) All paths $\langle \mathsf{nd}_1, \mathsf{nd}_2 \rangle, ..., \langle \mathsf{nd}_{n-1}, \mathsf{nd}_n \rangle$ with $n \geq 2$ such that each edge is labelled with dependency relation $\mathsf{compound}$ are collapsed to one node nd_1 with new label $\mathsf{nd}_1.\ell \oplus$ " " $\oplus ... \oplus$ " " $\oplus \mathsf{nd}_n.\ell$ where $\oplus$ denotes string concatenation.

($\sharp$) All paths $\langle \mathsf{nd}_1, \mathsf{nd}_2 \rangle, ..., \langle \mathsf{nd}_{n-1}, \mathsf{nd}_n \rangle$ with $n \geq 2$ such that each edge is labelled with either prep or case are collapsed to one node nd_1.

Due to (♯) the dependency tree in Figure 1a is reduced to the tree in Figure 1b. This post-processing can be easily realised using a tree traversal algorithm.

4.2 Building Concepts

After parsing the phrase to identify the important parts, atomic concepts from the KB need to be linked to the tree nodes, a problem referred to as entity linking [12]. Furthermore, a KB property between them needs to be established in order to construct a (complex) concept.

Many approaches have been proposed for entity linking, most of them involving supervised machine learning methods [24]. We use a simple, yet effective and scalable, information retrieval approach, which does not require a large training dataset. More precisely, given the label $\mathsf{nd}.\ell$ of a node in the tree, *ElasticSearch* is used to search the KB for the concept or property with the most similar label, denoted by $\mathsf{linkC}(\ell)$ and $\mathsf{linkR}(\ell)$, respectively. If no matching concept can be found, $\mathsf{linkC}(\ell) = \bot$, and if no matching property can be found, $\mathsf{linkR}(\ell) = \mathsf{assocWith}$, using the most common and versatile property.

A concept corresponding to a normalised dependency tree is defined recursively as given in Definition 1. We distinguish two strategies for obtaining a property between two concepts. If the tree contains three adjacent nodes nd_i, nd_j, and nd_k such that nd_j is a verb, linkR links the verb to a property in the KB to connect the concepts represented by nodes nd_i and nd_k; otherwise, a property is "mined" using two adjacent nodes representing concepts.

Definition 1. *For a node* nd_i *in some dependency tree, the concept corresponding to* nd_i *is* $\bot$ *if* $\mathsf{linkC}(\mathsf{nd}_i.\ell) = \bot$*, otherwise it is recursively defined as follows:*

$$\begin{aligned}\mathsf{linkC}(\mathsf{nd}_i.\ell) = {} & \textstyle\bigsqcap_{\langle \mathsf{nd}_i,\mathsf{nd}_j\rangle.\ell=\mathsf{acl},\langle \mathsf{nd}_j,\mathsf{nd}_k\rangle} \exists \mathsf{linkR}(\mathsf{nd}_j.\ell).C_k \sqcap \\ & \textstyle\bigsqcap_{\langle \mathsf{nd}_i,\mathsf{nd}_j\rangle.\ell\neq\mathsf{acl}} \exists \mathsf{mine}(\mathsf{nd}_i.\ell, \mathsf{nd}_j.\ell).C_j\end{aligned}$$

where C_k *and* C_j *are the concepts corresponding to the sub-trees rooted at nodes* nd_k *and* nd_j*, respectively.*

Function mine is based on domain and range restrictions of properties. Consider for example nodes nd_1 and nd_2 in Figure 1b and assume that they have been linked to concepts Pain and Recent from SNOMED CT, respectively. Pain is a sub-concept of ClinicalFinding and Recent is a sub-concept of TemporalConcept. According to the SNOMED CT documentation, property temporalContext has these two concepts as a domain and range, respectively, so we can create concept $\mathsf{Pain} \sqcap \exists \mathsf{temporalContext}.\mathsf{Recent}$. If no property can be mined this way, mine returns assocWith. Modern KBs often come with such domain and range restrictions (e.g., DBpedia contains almost 2,000). If no such axioms exist then properties between concepts can be mined using, e.g., statistical approaches [19].

Our implementation of constructConcepts returns the concept corresponding to the root of the dependency tree and traverses it using a depth-first algorithm. Applied on the dependency tree of Figure 1b, exploiting the domain and range restrictions in SNOMED CT and the existence of the verb in the tree, our method constructs concept $\mathsf{Pain} \sqcap \exists \mathsf{temporalContext.Recent} \sqcap \exists \mathsf{causedBy.Injury}$.

If a dependency tree contains coordinating conjunctions, we split the dependency tree into multiple trees [31], construct a concept for each according to Definition 1 and connect them with conjunction to form an overall concept.

Even though the above method was initially developed as part of our hybrid reasoning framework (i.e. method constructConcepts in Algorithm 1), it can be used as a stand-alone approach for concept extraction from any phrase that follows a structure similar to concept labels. It can thus enable the usage of our hybrid reasoner with *textual input queries* to verify if these queries express the same information: first concepts are constructed from the queries using the stand-alone knowledge extraction component and subsequently the hybrid reasoner is applied as usual. This pipeline approach is further discussed in Section 6. In the rest of this paper, we refer to our concept extraction method as *concept builder.*

5 Practical Hybrid Reasoning over Large KBs

Our hybrid reasoning approach presented in Section 3 uses logic-based reasoning to check subsumption between two given concepts (lines 1, 6, and 9 in Algorithm 1), which is implemented in SPARQL.

Unfortunately, the concepts we are dealing with may involve a number of conjuncts, existential quantifiers, or even be nested, while SPARQL cannot check subsumption recursively. To address these issues, we use a form of structural subsumption [3], which reduces subsumption checking to subsumption between the atomic elements of a (potentially complex) concept.

Definition 2. *Let* $C := \bigsqcap_i A_i \sqcap \bigsqcap_{j=1}^m \exists R_j.E_j$ *and* $D := \bigsqcap_k B_k \sqcap \bigsqcap_{l=1}^n \exists S_l.F_l$ *be two concepts and let* $\mathcal{K}$ *be a KB. We say that* C *is* structurally subsumed *by* D*, denoted by* $C \sqsubseteq_s D$*, if and only if the following hold:*

1. *For every* B_k *some* A_i *exists such that* $\mathcal{K} \models A_i \sqsubseteq B_k$*.*
2. *for every* $l \in [1, n]$ *either of the following holds:*
   (a) *there exists* $j \in [1, m]$ *s.t.* $\mathcal{K} \models R_j \sqsubseteq S_l$ *and* $\mathcal{K} \models E_j \sqsubseteq_s F_l$
   (b) *some* $A_i \sqsubseteq \exists T.G \in \mathcal{K}$ *exists s.t.* $\mathcal{K} \models T \sqsubseteq S_l$ *and* $\mathcal{K} \models G \sqsubseteq_s F_l$*.*

If some conjunct $\exists S_l.F_l$ of D does not subsume any conjunct in C, condition 2b performs an *expansion* on one of the main concepts A_i of C, using the KB to check if A_i is subsumed by some other concept that is subsumed by $\exists S_l.F_l$. Definition 2 can be easily implemented as SPARQL queries over triple-stores.

Example 3. Consider a KB $\mathcal{K}$ containing the following axioms:
$\mathsf{FootPain} \sqsubseteq \exists \mathsf{findingSite.Foot}, \quad \mathsf{FootPain} \sqsubseteq \mathsf{Pain}, \mathsf{Foot} \sqsubseteq \mathsf{Limb}$
Let $C_1 = \mathsf{FootPain}$ and $C_2 = \mathsf{Pain} \sqcap \exists \mathsf{findingSite.Limb}$ and assume we want to check whether $\mathcal{K} \models C_1 \sqsubseteq C_2$, which is indeed the case. According to Definition 2, C_1 is structurally subsumed by C_2 since $\mathcal{K} \models \mathsf{FootPain} \sqsubseteq \mathsf{Pain}$ and $\mathsf{FootPain} \sqsubseteq \exists \mathsf{findingSite.Foot} \in \mathcal{K}$ such that $\mathcal{K} \models \mathsf{Foot} \sqsubseteq_s \mathsf{Limb}$ (condition 2b of Definition 2).

As discussed in Section 3, concept extraction and unfolding should only be applied in our hybrid reasoning framework if we have reason to believe that it will be helpful for proving a correct subsumption. Since our concept extraction method uses labels of concepts, we also base the instantiation of the applicability condition sim on the *label* similarity of the compared concepts C and D.

Definition 3. *Let concepts C and D of the same form as in Definition 2 be the input of Algorithm 1. Let $\mathsf{str\text{-}sim}$ be some string similarity algorithm and let $\oplus$ denote string concatenation. We instantiate $\mathsf{sim}(C, D)$ in line 2 of Algorithm 1 as $\mathsf{str\text{-}sim}(\oplus_i \mathscr{L}(A_i) \oplus \oplus_{j=1}^{m} \mathscr{L}(E_j), \oplus_k \mathscr{L}(B_k) \oplus \oplus_{l=1}^{n} \mathscr{L}(F_l))$.*

For example, for concepts $C = \mathsf{FootPain}$ and $D_1 = \mathsf{Pain} \sqcap \exists \mathsf{findingSite.Head}$, the similarity score is expected to be lower compared to the one for C and $D_2 = \mathsf{Pain} \sqcap \mathsf{findingSite.Foot}$. Using an appropriate threshold σ_1, we can avoid applying concept builder on C when we compare it to D_1 but do apply it when we compare it to D_2, so as to extract conjunct $\exists \mathsf{findingSite.Foot}$ from label "Foot Pain" of C, which appears in D_2 and thus lets us prove subsumption. We apply Levenshtein distance for $\mathsf{str\text{-}sim}$ and find that setting σ_2 to half of the shorter string's length works well.

For the instantiation of the second applicability condition diff, we check whether the reason that subsumption failed was that some conjuncts in D do not subsume any conjuncts in C.

Definition 4. *Let C, D be concepts of the same form as in Definition 2 and let $\mathcal{K}$ be a KB. Function $\mathsf{diff}(C, D)$ returns all $\exists S_l.F_l$ in D such that both of the following hold:*

1. *$\forall j \in [1, m]$ either $\mathcal{K} \not\models R_j \sqsubseteq S_l$ or $\mathcal{K} \not\models E_j \sqsubseteq_s F_l$, and*
2. *no $A_i \sqsubseteq \exists T.G \in \mathcal{K}$ exists such that $\mathcal{K} \models T \sqsubseteq S_l$ and $\mathcal{K} \models G \sqsubseteq_s F_l$*

If the set returned has more than σ_2 elements (for our purpose $\sigma_2 = 0$ works well), then Algorithm 1 will try to extract conjuncts from C that are subsumed by those in D using concept builder in line 5. An empty set expresses that the reason for non-subsumption of C and D is that A is not subsumed by B, in which case B rather than A should be unfolded to be able to prove that A is a more specific concept than B.

Example 4. Consider Example 1, where
$C_1 := \mathsf{RecentInjury} \sqcap \exists \mathsf{findingSite.Head}$
$C_2 := \mathsf{Injury} \sqcap \exists \mathsf{findingSite.Head} \sqcap \exists \mathsf{occurred.Recently}$
Then, $\mathsf{diff}_{\mathcal{K}}(C_1, C_2) = \{\exists \mathsf{occurred.Recently}\}$, which does not subsume any conjunct of similar form in C_1 and there is no such conjunct in the KB for $\mathsf{RecentInjury}$ either. Hence, Algorithm 1 will extract a concept from the label of C_1 as desired (if $\sigma_2 = 0$).

In contrast, $\mathsf{diff}_{\mathcal{K}}(C_2, C_1)$ returns $\emptyset$ since the only conjunct of C_1 also exists in C_2 ($\exists \mathsf{findingSite.Head}$). Thus, when calling $\mathsf{isSubsumed}_{\mathcal{K}}(C_2, C_1)$ the algorithm will skip the block in lines 4–7 and will only apply concept builder on C_1.

Table 1. Examples of concepts constructed by concept builder for SNOMED labels

partially correct concept	
Blood in Urine	$\mathsf{Blood} \sqcap \exists \mathsf{assocWith}.\mathsf{Urine}$ $\mathsf{Blood} \sqcap \exists \mathsf{findingSite}.\mathsf{Urine}$ (doctor concept)
wrong concept	
Prune belly syndrome	$\mathsf{Prune} \sqcap \exists \mathsf{assocWith}.\mathsf{Syndrome} \sqcap \exists \mathsf{findingSite}.\mathsf{Belly}$ $\mathsf{PruneBellySyndrome}$ (doctor concept)

6 Evaluation

Our work was motivated by the need to compare medical knowledge encoded in different services in Babylon Health, such as an AI-based symptom-checking chatbot, triaging, drug prescriptions, and telemedicine. Medical information in all services is encoded using (complex) concepts built from a *large medical KB* [4], curated from sources like SNOMED CT and NCI and stored in GraphDB [26].

Even though the same KB is used, different services (and even humans) may encode the same information in different ways. For example, concepts representing symptoms in the triaging system were manually created by doctors, encoding "*swelling of ear*" as an atomic KB concept $\mathsf{SwollenEar}$. However, during a patient interaction with the chatbot the same phrase may be automatically encoded as $\mathsf{Swelling} \sqcap \exists \mathsf{findingSite}.\mathsf{Ear}$. To allow interoperability of services, a system able to identify these two concepts as being equivalent needs to be in place. We deploy our hybrid reasoner for this task.

In the following, we evaluate the performance of our hybrid reasoner on two different use cases that require subsumption reasoning. First, however, we evaluate the performance of concept builder in isolation, as it is an integral part of our hybrid reasoner and can be used as a stand-alone concept extraction tool.

6.1 Performance of Concept Builder

We randomly sampled 200 SNOMED CT concept labels containing at least two words and applied concept builder to extract (complex) concepts linked to the KB. Three doctors then evaluated the quality of concepts constructed by concept builder by classifying each as *correct*, *partially correct*, or *wrong*. Rare disagreements between doctors' judgements were resolved by discussions.

For 19 labels, concept builder failed to extract a concept from the SNOMED label as it was unable to link some of the words in the label to KB concepts. For the remaining 181 SNOMED labels concept builder exhibits good performance, with 60% of extracted concepts evaluated as correct by doctors, 29% as partially correct, and only 11% as wrong. As illustrated in Table 1, the most common reasons for concepts being wrong or only partially correct are an incorrect choice of property (first example in the table) and insufficient tree pruning, resulting in too many atomic concepts in the constructed concept (second example in the table). Overall, we consider the performance of our concept builder sufficiently good to be applied as the knowledge extraction component in our hybrid reasoner and as a stand-alone tool for constructing concepts from short phrases.

Table 2. Number of equivalences and subsumptions proven by hybrid reasoner between complex /atomic doctors' concepts and supposedly equivalent concept builder concepts.

Type of Reasoning	**complex**					**atomic**				
	equiv	subs	none	error	timeout	equiv	subs	none	error	timeout
logic	0	0	0	0	0	169	8	55	0	0
+ expansion	29	0	0	1	0	68	0	0	0	17
+ builder	36	4	0	0	0	411	68	67	2	2
+ expansion/builder	24	4	100	5	0	83	17	0	0	6
total	89	8	100	6	0	731	93	122	2	25

6.2 Use Case 1: Constructing the Symptom-Checking Network

The first dataset to evaluate our hybrid reasoner is based on a symptom-checking engine, consisting of a network of 1176 nodes representing symptoms, diseases, and risk factors interconnected with probabilities. A group of doctors created this network manually by associating each medical entity node they wanted the network to contain with an atomic or complex concept based on the KB. For example, the doctors created a node for "Aching Epigastric Pain" and associated it with the concept $\mathsf{EpigastricPain} \sqcap \exists \mathsf{hasSensation}.\mathsf{AchingSensation}$. Besides the concept, the group of doctors also stored the name of the medical entity they represented.

To test how well our hybrid reasoner can identify equivalent concepts, we apply our concept builder on the name associated with each node to automatically obtain a second concept representation D using the KB. Since D is based on the same entity name as concept C constructed by doctors, C and D should be equivalent. We use the hybrid reasoner to check this, i.e. if $C \sqsubseteq D$ and $D \sqsubseteq C$, giving an insight into how well the hybrid reasoner can identify equivalences.

Our results are summarised in Table 2, distinguishing between atomic and complex concepts created by doctors. The columns "equiv" and "subs" denote cases where both or only one of the above subsumption calls to the reasoner succeeded. Column "none" counts concepts for which our hybrid reasoner could not prove subsumption, "error" denotes cases in which concept builder failed to construct a concept for the given phrase, and "timout" cases in which the reasoner exceeded the time-out limit of five seconds. Rows distinguish the type of reasoning performed by the reasoner while trying to prove the two subsumption directions, in particular, simple *logic*-based reasoning, i.e. no expansion and no concept extraction (line 1 in Algorithm 1 without using 2b in Definition 2), logic-based reasoning with concept *expansion* (line 1 in Algorithm 1 using 2b in Definition 2), and logic-based reasoning (and concept expansion) with the application of concept *builder* on KB labels (Algorithm 1 past line 1). The last row, *expansion/builder*, denotes cases where one subsumption direction required expansion (without concept builder) and the other direction also required the usage of concept builder.

Table 3. Examples and reasons why our hybrid reasoner could not prove equivalence between two concepts (top concept – concept builder; bottom concept – doctor).

Semantic Equivalence	
Poor Hygiene	$\mathsf{Hygiene} \sqcap \exists \mathsf{associatedWith}.\mathsf{Poor}$ $\mathsf{NeglectOfPersonalHygiene}$
Highly Complex Concept	
Analgesia overusee headache	$\mathsf{Headache} \sqcap \exists \mathsf{hasDef}.\mathsf{Analgesia} \sqcap \exists \mathsf{assocWith}.\mathsf{RepetitiveStrainInjury}$ $\mathsf{AnalgesicOveruseHeadache}$

Table 2 shows that our hybrid reasoner exhibits good performance: in 70% of the cases it can prove the equivalence of concepts and in another 9% it can prove that one is subsumed by the other. Concept expansion has a significant impact in increasing the inference power of determining subsumption. Furthermore, applying concept builder for on-the-fly knowledge extraction from concept labels leads to another huge improvement. Note that for complex concepts, pure logic-based reasoning is unable to prove any equivalences, illustrating the usefulness of our hybrid reasoner.

Further analysis revealed that logic-based reasoning is only able to prove equivalence if the two given concepts C and D are in fact the same atomic concept A, i.e. $C = A$ and $D = A$. In contrast, in all cases in which our hybrid reasoner applied knowledge extraction and expansion to prove equivalence, at least one of the given concepts was a complex concept, making the reasoning much more challenging. Note that for any complex concept by a doctor, concept builder constructed a different complex concept, making subsumption checking extremely difficult. Despite this complexity, our hybrid reasoner is able to prove equivalence in half of these highly challenging cases.

The main reasons why our reasoner failed to prove subsumption between concepts was that the two concepts were semantically equivalent but expressed using completely different concepts and that one concept was way more complex than the other, as illustrated in Table 3. Overall, this use case shows the advantage of our hybrid subsumption reasoning method, which applies knowledge extraction to tackle the inherent incompleteness in KBs, as compared to pure logic-based reasoning.

6.3 Use Case 2: Understanding User Queries for Symptom-Checking

The symptom-checking engine from the first use case is accessed by end-users through a chatbot. Given a user query such as "My stomach hurts", the correct node from the network needs to be activated to initiate symptom-checking.

Before the development of our hybrid reasoner, two different methods were tested to map user queries to nodes in the network. First, the *GATE* text annotation system [7], which applies string matching and hand-crafted rules, was used to identify names of symptom nodes in a query. Second, the Google universal *sentence embedder* [5] (based on Transformer) was used to embed both the

Table 4. Performance of different methods on our query–symptom dataset.

	GATE	**GATE**$_{\textbf{emb}}$	**emb**	**ConBReas**	**ConBReas**$_{\textbf{emb}}$
Precision	1.00	0.86	0.72	0.96	0.84
Recall	0.53	0.74	0.72	0.79	0.88

user query and all symptom nodes in the network and then the node with the closest vector compared to the query vector (using cosine similarity) was chosen.

Our hybrid reasoner can be used to map queries to nodes as follows: first, concept builder extracts a concept C from user text, then the hybrid reasoner finds a "closest" matching symptom by checking which node in the network is associated with a concept D that subsumes C, i.e. for which D it holds that $C \sqsubseteq D$. We refer to this pipeline as ConBReas(Concept Builder–Reasoner).

To systematically compare the performance of the three methods, doctors created a dataset of 1878 mock user queries[2] and matched each with a node from the network. For example, doctors matched the query "*I can't focus*" to the node associated with concept PoorConcentration. In addition to comparing the performance of GATE and the embedder to ConBReas, we also experiment with using the embedder as a fallback in GATE and ConBReas, that is, if either of these approaches fails to return a node from the network then the embedder is used; we call these settings GATE$_{\text{emb}}$ and ConBReas$_{\text{emb}}$, respectively.

Performance results of the different methods are given in Table 4. As expected, GATE offers the lowest recall but highest precision, since it performs an almost exact lexical match between the words in a query and the possible symptom labels. The combination of GATE with an embedder increases its recall, but decreases precision. The embedder provides better recall but the precision is too low for a medical application. ConBReas outperforms the embedder in terms of both precision and recall. Compared to GATE, ConBReas' precision is only slightly lower, while its recall is much higher. ConBReas$_{\text{emb}}$ obtains the best trade-off between precision and recall among all methods, and is thus used in production to link user queries to the symptom checking engine.

For future improvement, we performed an error analysis and found that most of ConBReas' mistakes are due to concept construction from the user query, and in particular the entity linking step if sophisticated common-sense reasoning is needed to link the correct concept. Table 5 gives some examples of false positives (wrong concept constructed) and false negatives (no concept constructed).

7 Related Work and Conclusions

To the best of our knowledge, our hybrid reasoning framework is the first approach applying statistical methods as part of logic-based subsumption checking. Conversely, Movshovitz-Attias et al. [18] train a subsumption classifier and include logic-based features using a KB, in particular the overlap of concepts'

[2] for privacy reasons we do not use real user queries

Table 5. Examples of false positives (FP) and false negatives (FN) of ConBReas.

User Query	ConBReas	Correct Symptom	Type
I don't sleep well	SleepingWell	RestlessSleep	FP
I'm hungry all the time	$\mathsf{Always} \sqcap \exists \mathsf{assocWith.Hungry}$	IncreasedAppetite	FP
I can't speak properly	—	DifficultySpeaking	FN
I'm getting skinnier	—	WeightLoss	FN

properties in Biperpedia. Like us they exploit dependency trees, used in terms of a feature expressing whether any paths in the dependency trees of the two concepts match. In contrast to our hybrid reasoner, the results of their classifier are not easily explainable and the classifier needs sufficient training data, whereas our reasoner can be applied without training and can be easily interpreted.

The problem of learning DL axioms from unstructured text has received considerable attention [11, 22, 8, 28]. The structure of text targeted by these works is substantially different to the one usually found in concept labels. These works deal with verb phrases which often have some definitory character, e.g. from text "*Enzymes* **are** *proteins* that **catalyse** *chemical reactions*" an axiom of the form $\mathsf{Enzyme} \sqsubseteq \mathsf{Protein} \sqcap \exists \mathsf{catalyse.ChemicalReaction}$ can be extracted. In contrast, concept labels – as used by us – are usually short phrases without verbs, so properties between entities need to be inferred as they cannot be extracted from the text. The work by Romacker [25] is close to ours as they also use dependency parsing and then map the dependency tree to DL concepts. However, again full sentences are targeted using a form of reification. For our running example (Figure 1a) their approach would create $\mathsf{Provoking} \sqcap \mathsf{provokingAgent.Injury} \sqcap \exists \mathsf{provokingPatient.Pain}$, whereas ours produces simpler and more concise concepts enabling easier subsumption checking.

Concept labels have been previously proven useful for extracting additional information from KBs [9, 21]. Fernandez-Breis et al. [9] use manually created patterns to "decompose" concept labels, whereas Pacheco et al. [21] use all subwords in a concept label and do not consider any dependency relations between them. Our approach is less rigid as it construct concepts dynamically based on the structure of the dependency tree.

The structure of concept labels found in KBs resembles keyword-based and entity-oriented natural language queries [23, 13, 17]. Compared to Pound et al. [23] we can extract arbitrary (even nested) existentially quantified concepts and do not require training of application-specific template classifiers. Furthermore, we do not require that relations between concepts are expressed by words in the query [13]. Similar to us, Lei et al. [17] derive KB properties to connect concepts from words in the query or, if not present, from the KB structure. However, they construct a tree from the query which needs to be a sub-tree of the KB, whereas our complex concepts are not tied to the KB structure in this way. Xu et al. [30] use dependency trees for relation extraction, however they require task-specific training data, whereas we apply pre-trained dependency parsers.

The comparison with related work highlights the novelty of our hybrid reasoning framework in integrating knowledge extraction from concept labels using NLP methods into logic-based subsumption checking. We have provided an instantiation of our hybrid reasoning framework paying extra care on design choices in order to provide a highly efficient system that can operate in a production environment. Our evaluation on two industrial use cases showed that our approach provides the best combination of precision and recall compared to purely statistical or purely rule-based approaches and our reasoner is thus currently employed in an industrial production environment.

Although our use case and implementation regards the medical domain, all methods are domain independent and can be adapted easily to new domains. For example, concept builder can be used on phrases like "Italian Painters" over DBpedia to construct concept $\mathsf{Painter} \sqcap \exists \mathsf{birthPlace.Italy}$ or on "Thriller Movies" over the Movie KB[3] to construct concept $\mathsf{Movie} \sqcap \exists \mathsf{belongsToGenre.Thriller}$. Future work will target better compounding methods in concept builder to avoid the construction of over-complicated concepts as well as the integration of commonsense reasoning methods to infer equivalence in difficult cases.

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. In: ISWC/ASWC. pp. 722–735 (2007)
2. Baader, F., Brandt, S., Lutz, C.: Pushing the EL envelope. In: IJCAI. pp. 364–369 (2005)
3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
4. Barisevičius, G., Coste, M., Geleta, D., Juric, D., Khodadadi, M., Stoilos, G., Zaihrayeu, I.: Supporting digital healthcare services using semantic web technologies. In: ISWC. pp. 291–306 (2018)
5. Cer, D., Yang, Y., Kong, S., Hua, N., Limtiaco, N., John, R.S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Sung, Y., Strope, B., Kurzweil, R.: Universal sentence encoder. CoRR **abs/1803.11175** (2018)
6. Choi, J.D., McCallum, A.: Transition-based dependency parsing with selectional branching. In: ACL. pp. 1052–1062 (2013)
7. Cunningham, H., Tablan, V., Roberts, A., Bontcheva, K.: Getting more out of biomedical documents with gate's full lifecycle open source text analytics. PLoS Computational Biology **9**(2) (2013)
8. Distel, F., Ma, Y.: A hybrid approach for learning SNOMED CT definitions from text. In: DL. pp. 156–167 (2013)
9. Fernandez-Breis, J.T., Iannone, L., Palmisano, I., Rector, A.L., Stevens, R.: Enriching the gene ontology via the dissection of labels using the ontology pre-processor language. In: EKAW. pp. 59–73 (2010)
10. Galárraga, L., Razniewski, S., Amarilli, A., Suchanek, F.M.: Predicting completeness in knowledge bases. In: WSDM. pp. 375–383 (2017)

[3] http://www.movieontology.org/

11. Gyawali, B., Shimorina, A., Gardent, C., Cruz-Lara, S., Mahfoudh, M.: Mapping natural language to description logic. In: ESWC. pp. 273–288 (2017)
12. Hachey, B., Radford, W., Nothman, J., Honnibal, M., Curran, J.R.: Evaluating entity linking with wikipedia. Artificial Intelligence **194**, 130–150 (2013)
13. Han, S., Zou, L., Yu, J.X., Zhao, D.: Keyword search on rdf graphs - a query graph assembly approach. In: CIKM. pp. 227–236 (2017)
14. Hou, J., Nayak, R.: A concept-based retrieval method for entity-oriented search. In: AusDM. pp. 99–105 (2013)
15. Kazakov, Y., Krötzsch, M., Simancik, F.: The incredible ELK - from polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies. Journal of Automated Reasoning **53**(1), 1–61 (2014)
16. Kübler, S., McDonald, R.T., Nivre, J.: Dependency Parsing. Synthesis Lectures on Human Language Technologies, Morgan & Claypool Publishers (2009)
17. Lei, C., Özcan, F., Quamar, A., Mittal, A.R., Sen, J., Saha, D., Sankaranarayanan, K.: Ontology-based natural language query interfaces for data exploration. IEEE Data Engineering Bulletin **41**(3), 52–63 (2018)
18. Movshovitz-Attias, D., Whang, S.E., Noy, N.F., Halevy, A.Y.: Discovering subsumption relationships for web-based ontologies. In: WebDB. pp. 62–69 (2015)
19. Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E.: A review of relational machine learning for knowledge graphs. Proceedings of the IEEE **104**(1), 11–33 (2016)
20. Oramas, S., Ostuni, V.C., Noia, T.D., Serra, X., Sciascio, E.D.: Sound and music recommendation with knowledge graphs. ACM TIST **8**(2), 21:1–21:21 (2016)
21. Pacheco, E.J., Stenzhorn, H., Nohama, P., Paetzold, J., Schulz, S.: Detecting underspecification in SNOMED CT concept definitions through natural language processing. In: AMIA (2009)
22. Petrova, A., Ma, Y., Tsatsaronis, G., Kissa, M., Distel, F., Baader, F., Schroeder, M.: Formalizing biomedical concepts from textual definitions. Journal of Biomedical Semantics **6**(1), 22 (2015). https://doi.org/10.1186/s13326-015-0015-3
23. Pound, J., Hudek, A.K., Ilyas, I.F., Weddell, G.: Interpreting keyword queries over web knowledge bases. In: CIKM. pp. 305–314 (2012)
24. Raiman, J., Raiman, O.: Deeptype: Multilingual entity linking by neural type system evolution. In: AAAI. pp. 5406–5413 (2018)
25. Romacker, M., Markert, K., Hahn, U.: Lean semantic interpretation. In: IJCAI. pp. 868–875 (1999)
26. Stoilos, G., Geleta, D., Shamdasani, J., Khodadadi, M.: A novel approach and practical algorithms for ontology integration. In: ISWC (2018)
27. Stuckenschmidt, H., Ponzetto, S.P., Meilicke, C.: Detecting meaningful compounds in complex class labels. In: EKAW. pp. 621–635 (2016)
28. Völker, J., Hitzler, P., Cimiano, P.: Acquisition of OWL DL axioms from lexical resources. In: ESWC. pp. 670–685 (2007)
29. Wessel, M., Acharya, G., Carpenter, J., Yin, M.: OntoVPA - an ontology-based dialogue management system for virtual personal assistants. In: IWSDS. pp. 219–233 (2017)
30. Xu, H., Hu, C., Shen, G.: Discovery of dependency tree patterns for relation extraction. In: PACLIC. pp. 851–858 (2009)
31. Zhang, W., Liu, S., Yu, C., Sun, C., Liu, F., Meng, W.: Recognition and classification of noun phrases in queries for effective retrieval. In: CIKM. pp. 711–720 (2007)