Semantic Web 0 (0) 1
IOS Press

Automatic Ontology Population from French Classified Advertisements

Céline Alec
Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France
E-mail: celine.alec@unicaen.fr

Abstract. Artificial intelligence has become one of the most studied fields in computer science. It aims at building smart machines capable of performing tasks that typically require human intelligence. One of the challenging tasks consists in understanding texts written in natural language. Semantic Web technologies, in particular ontologies, can be used to help agents representing knowledge from a specific domain and reasoning like a human would do. Ontology population from texts aims to translate textual contents into ontological assertions. This paper deals with an approach of automatic ontology population from French textual descriptions. This approach has been designed to be domain-independent, as long as a domain ontology is provided. It relies on a text-based and a knowledge-based analysis, which are fully explained. Experiments performed on French classified advertisements are discussed and provide encouraging results.

Keywords: Ontology population, Knowledge engineering, Knowledge graph, OWL, Semantic Web technologies

1. Introduction

Ontologies [1] are designed to store and share domain knowledge between humans and machines. They have two main parts : TBox and ABox. TBox includes a hierarchy of classes and relationships between these classes, called object properties, or relationships between a concept and a literal, called data properties. ABox is composed of instances, which may be class assertions (instances of concept) or property assertions (instances of property). The process of adding instances to an ontology is called ontology population. A populated ontology may also be referred as a knowledge base or a knowledge graph. Ontologies are particularly popular in the context of the Linked Open Data (LOD) cloud [2], which contains knowledge graphs, interlinked and openly available, like DBpedia [3], Wikidata [4] or YAGO [5]. Knowledge graphs/ontologies can also be used outside of the LOD, for specific needs, especially in industrial contexts [6–9].

The approach proposed in this paper is part of the DECA project (Detection of Errors and Correction of Annotations). This project deals with annotated descriptions, i.e., descriptions that are in the form of texts, to which annotations are appended. This is, for example, the case of news articles, annotated with their themes, or classifieds advertisements, annotated with the criteria they meet. Annotations are theoretically meant to describe characteristics of the object or event described in the description. However, this is not always the case. In fact, erroneous annotations may be frequently observed, either because of typing errors or because of "misuse": descriptions are deliberately wrongly annotated in order to increase their visibility. For example, one can find a real estate advertisement where the city annotation is a well-known city *X*, while the description states "30 minutes from *X*"; or where the area annotation (in m^2) is 150 while the description states "147 m^2". These wrongly annotated descriptions waste a considerable amount of time for readers (or buyers in the case of classified ads) who must sort through the multiple responses complying in theory with their search criteria. The DECA project deals with this problem. Its goal is to detect and correct erroneous annotations thanks to the inconsistencies that can be detected according to the content expressed in the textual description. The solution idea is based on the human process: first, (i) the text needs to be understood, and then (ii) information from both text and annotations needs to be used in a reasoning step to be able to detect and correct wrong annotations. In order to do this, a domain ontology will be used. The final proposition has to be as generic as possible, in the sense that it should be applicable to different types of annotated descriptions of objects, in particular classified ads (real estate ads for sale / rental / seasonal rental, sale of objects : vehicles / fashion / furniture / etc., job offers), but also other types of annotated descriptions (like news articles or encyclopedic pages), on condition that an ontology, specific to the field being dealt with, is available. This paper focuses on the sub-problem of understanding the text (i), that is, the goal is to represent the content of textual descriptions of objects in a domain ontology. It is therefore a problem of ontology population from textual descriptions. Our first use case concerns real estate house sales announcements, but the proposed approach has to be as generic as possible, i.e., it should be able to populate a domain ontology from descriptions in many domains : whether it be a subcategory of classified ads, or descriptions of a certain type of object. The only condition is to have an ontology whose domain is the object's type. Unlike texts from state-of-the-art approaches, descriptions have verbs that are not very discriminating and few named entities, which makes our context original. Our contribution is a proper methodology that integrates not only object properties but also data properties, taking ternary properties into consideration.

The rest of the paper is organized as follows: Section 2 presents the related work and positions our contribution. Section 3 describes our approach. Experiments on our use case are explained and evaluated in Section 4. Section 5 concludes and suggests future work.

2. Related work

Ontologies have first been defined by [10] as specifications of a conceptualization, that is, an ontology is a description of the classes and relationships that can exist for an agent or a community of agents. Ontologies have been widely studied since then. They provide a way to store semantic knowledge in a machine-readable format. In particular, the development of the Web has lead to a considerable amount of data publicly available. The Web is full of information but extracting it such that a machine could be able to automatically understand it is still a challenge of the Semantic Web. Ontologies address the problem of representing semantic knowledge on a particular domain or task. This knowledge may help, for example, to semantically annotate documents, or to deduce new facts based on a reasoning.

Constructing ontologies is a time consuming task. The process of their automatic or semi-automatic construction, enrichment and adaptation is known as ontology learning, which is a wide research area [11, 12] that includes work on ontology enrichment, inconsistency resolution and ontology population. In this paper, we are interested in ontology population, which is the task of adding new instances to an ontology.

Ontology population has been studied in various papers. An overview of related work can be found in the survey [13]. Some approaches [14–16] focus their work on automatic extraction of concept instances without considering relation instances. We focus only on approaches aiming at extracting information from unstructured or semi-structured textual documents and a domain ontology given in input, allowing the addition of instances in the latter, in particular the addition of property assertions. Existing systems are based on various rule-based methods using lexico-syntactic patterns, such as ArtEquAKT [17] or SOBA [18]; or on methods based on machine learning, such as Adaptiva [19], LEILA [20], Ontosophie [21] or *Web→KG* [22]; or even on hybrid methods, like BOEMIE [23]. Using machine learning techniques requires having a sufficient quantity of sentences and their ontology correspondence upstream, which is not the case of our data. We therefore concentrate on approaches that exploit lexico-syntactic patterns.

In [24], the authors use lexico-syntactic patterns ("such as", "is a") allowing them to identify hyponymy relationships. More elaborate versions [25–27] are inspired by this idea. They assist an expert by providing candidate patterns, based either on a sentence analyser or on examples. The expert validates the correct patterns. The ArtEquAKT approach [17] deals with ontology population from the Web in the domain of artists. This approach populates the ontology with property assertions. It uses the verb found in a sentence between two instances of concept from the ontology. On the same principle, Makki [28] also focuses on verbs in order to populate the ontology with property assertions, but it is semi-automatic and domain independant. A list of verbs is extracted from the input corpus for each property from the ontology using Wordnet. A set of seven manually written rules is used to recognise subjects and objects of a potential property assertion. Results are then validated by an expert. In [29], a framework is proposed. The goal is to instantiate a concept as well as the relationships that concern it. First, named entities are identified in the text (exploiting also co-references). Second, triggers are considered, i.e., property names as well as their synonyms; and rules are built based on noun phrases preceded or followed by a trigger. The application of these rules leads to the population of the ontology. [9, 18] use structured data on top of texts, like tables or document tags. In [18], manually written heuristics are used to extract soccer-specific entities, and extraction rules need to be provided for relation instances. In [9], manually-defined mapping rules are exploited to link a tag from a structured document to a concept from the ontology.

Our contribution aims at populating the ontology without using any domain-based patterns or rules. The idea is to perform the population using only the input corpus and domain ontology. The domain terminology is used, just like in related work. However, we need neither an expert, nor manual knowledge. The study of state-of-the-art approaches shows several emerging scientific obstacles when trying to adapt them in the context of ontology population from descriptions of objects.

First of all, the textual analysis to populate properties cannot follow the standard scheme of current approaches, which are in a context where the expression "subject verb complement" in the text must become the triplet <subject> <predicate> <object> in the ontology. Indeed, verbs that are used in the texts from related work are strongly characteristic of a relationship (e.g., "is married to"), but they are not in object descriptions (e.g., "the good has" or "the good possesses"). Sometimes, there is no verb at all (e.g., "2 bedrooms, 1 bathroom."). This makes the population of properties more difficult. Moreover, we also note that most approaches refer to named entities for subject and object of property assertions, which is not, in general, the case in the context of object descriptions. Our context is quite singular because the texts focus on a particular object. That means that this object, or sometimes things linked to this object, are often the subject of the property assertions to be inserted. The ontology therefore follows a "star-shaped model" where the major concept (i.e., the object type) is central and from which a whole set of properties starts.

Furthermore, most of existing approaches focus on the population of object properties and leave data properties aside. Object properties are properties for which the object of the property is a resource while data properties have a literal as their object (string, numeric value, etc.). Data properties are an important part of properties in the context of classified advertisements. For example, in the case of real estate, there are many properties with a numerical value: expression of areas, number of rooms, energy diagnosis, distance to points of interest, etc. The context of the extracted numerical values must therefore be exploited to detect their precise semantics and populate the associated properties. Units of measurement (m^2, km, minutes, euros, etc. in the real estate case) are an important feature and may constitute exploitable clues.

Finally, existing works focus on the population of binary properties allowing a subject to be associated with an object. In our context, we have to deal with certain n-ary properties. For example, to represent the idea that a house for sale is located at 10 km from Paris, we face a ternary property linking a house to a distance and a place. One can imagine the modeling of a concept *Distance* making the representation of this ternary property possible, such that a good for sale is located at a distance d, d concerns a location, and d is associated to a value corresponding to the distance in kilometers. These kinds of ternary properties represent complex notions that are difficult to automatically populate. Their population has, to our knowledge, never been studied.

3. The KOnPoTe approach

We present KOnPoTe (Knowledge graph/ONtology POpulation from TExts), an approach to populate a domain ontology or a knowledge graph according to textual descriptions of elements of this domain. The initial knowledge graph describes the domain with classes, properties and axioms (ontology), and may possibly contain initial individuals. The domain is represented via a concept named hereafter *main class*. We consider a corpus of documents where each document describes an instance of the *main class*. Figure 1 shows the outline of the approach. The knowledge graph (KG), as well as each document from the corpus, are used by a terminology matcher. This leads to matchings between text mentions and entities from the KG (classes, properties and individuals). Then, a relatively complicated population algorithm is applied, to get the final output knowledge graph (see top of Figure 1). This population algorithm is a succession of several treatments (see middle of Figure 1): an initialisation of an object called "Matching treatments" (MT); an instantiation of the *main class* of the KG; a text-based analysis, composed of various treatments (see bottom of Figure 1); as well as an analysis based on the background knowledge from the ontology. In this section, we explain each treatment and unfold them on an example.

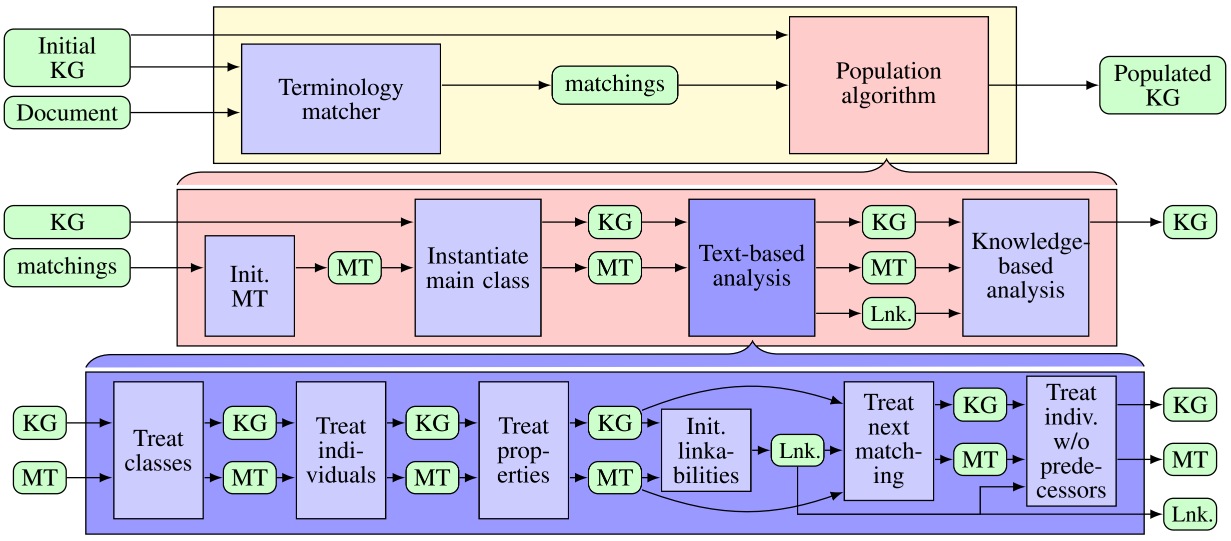

Fig. 1. Outline of the KOnPoTe approach

As mentioned in Section 1, the proposed approach has to be generic. It might be a bit confusing to mention genericity while dealing with a domain ontology, this is why we explain what being generic means in our context. The proposed approach should be applicable to various domains. For a given domain, it is necessary to have as input a KG or ontology of the domain, as well as a corpus of documents, where each document is a text that describes an instance of this domain. Let us insist on the fact that we only want to apply the approach on texts that are descriptions of one type of object, not on every kind of text. By generic, we mean that the proposed algorithm cannot depend on domain-based linguistic rules or patterns. Hence, it can only used domain terminology (from the ontology), syntactic indicators (like sentences, or order of expressions within the text) and knowledge indicators (like domains and ranges of a property). Being able to create such an algorithm is not easy, we therefore put one constraint on the input ontology: the domain is represented by a concept named *main class*. Intuitively, it is likely that the shape of the ontology looks like a star, where the center is the main class, since all knowledge is supposed to describe the main class. In other words, the main class is the domain of object properties whose range is the domain of other object properties, and so on. Note that domains and ranges of properties may not only be explicit classes but also class expressions, using all the expressiveness of OWL2 [30], like union, intersection, complement, etc.

3.1. Initial data

The KG is written in OWL[1] language [31], defined as a standard by the World Wide Web Consortium. Initially, it has a hierarchy of classes, object properties (linking classes together) and data properties (linking a class to a literal). Axioms representable in OWL2 and SWRL[2] rules [32] can be defined. Data properties that are considered may use boolean, numerical (integer, double), or string values. The KG may also contain initial individuals, which are generic. It describes the domain in a general way but does not initially describe any instance of this domain.

[1]Web Ontology Language
[2]Semantic Web Rule Language

In the following of this section, examples are given where the KG describes the house sales domain and where the document is a house sale classified ad. In the context of house sales, the initial KG[3] contains classes such as the main class *Property* (denoting real estate); classes designating rooms like *Room*, *Kitchen*, *Bedroom*; etc. Among the properties, we can cite the object property *is located in* linking a real estate property to the municipality in which it is located, or the data property *area in m^2* connecting a property part (land, house, etc.) or a room to a numerical value. The initial individuals are generic, e.g., instances of the class *Municipality*, or instances of the class *Heating system*. The KG also contains axioms, for example, the fact that a *Property* can only be located in a maximum of one *Municipality*, or the fact that a *Bedroom* is disjoint from a *Kitchen*. Finally, some SWRL rules are defined, for example, to express the fact that the boolean property *separate* has to be populated with the opposite value of the boolean property *open*.

Each element of this KG (classes, properties, individuals) has a Uniform Resource Identifier (URI) and possibly a more advanced terminology: labels (rdfs:label, rdfs:isDefinedBy), units or unit expressions. For these last two cases, a "unit" annotation property has been created, associating an ontological element respectively with its unit or with a fill-in-the-blank expression (whose blank is represented by "xxx"). For example, we can associate the property *area in m^2* with its unit *m^2*, and the property *feePercentage* with a unit expression *fees:* xxx *%*. This representation of units is currently simple. A change in the way of representing units (using, for example, a special unit class linked to a unit name and a unit value) may be considered in a future version.

3.2. The terminology matcher

The terminology matcher is the first step of the approach. As it can be seen in Figure 1, it takes as inputs the initial KG and a document, and outputs matchings between text mentions of the document and the KG. The text is split into sentences and then lemmatized. Likewise, the ontology keywords (URI fragments, labels, units) are lemmatized taking into account *camelCase* and *snake_case* syntaxes. Matchings between the text and the ontology keywords, both with their non-lemmatized and lemmatized versions, are established. Matchings also concern unit expressions, e.g., the mention *fees: 4%* matches the unit expression *fees:* xxx *%*. All included matchings are removed. For example, if there is a matching on the mention "city center" and on its sub-mention "city", only the one on "city center" is considered.

Cormelles-le-Royal : Magnificent house located 15 min from the city center of Caen and 3 min walk from shops and schools, 110 m^2 on a land of 400 m^2. The ground floor consists of a kitchen equipped with essential appliances, a living room with fireplace, facing south-west, as well as a bedroom of 15 m^2. First floor: 2 bedrooms and 1 bath-room. Close to public transport. Wooded and enclosed land. Fees : 4%.

Fig. 2. Example of a document (translated from French to English) and its matchings

Figure 2 shows a translated example of a real estate advertisement[4] as well as the matchings obtained with a KG describing the domain. The matchings concern individuals (Cormelles-le-Royal, city center, etc.), classes (house, land, etc.), object properties (located, close to) or data properties (min, min walk, etc.).

3.3. The population algorithm

The matchings obtained are then supplied as input to the population algorithm. As shown in Figure 1, the latter has four main tasks. The first two are quite basic; they can be seen as pre-processing tasks. The third, called text-based analysis, is a population task using an analysis of the matchings based on textual indicators (sentences, order of matchings within the text). The last task, called knowledge-based analysis, assumes that the previous task tried to link the individuals considered in the document as best as possible, but that there are possibly some gaps. It aims to add missing property assertions based on knowledge from the KG. These tasks are described in the following of this section.

[3]Our use case is in French but examples are translated into English in this paper.
[4]This example is fake but useful to explain our approach on a relatively small text. It is used throughout the paper.

3.3.1. The two pre-processing tasks

Initialisation of the matching treatments The first task is to initialise the matching treatments MT. This consists in creating a set of matching treatments, each of them corresponding to a matching. A matching treatment contains several attributes:

– *matching* : the matching to be treated
– *individuals* : the individual(s) associated to this matching
– *assertions* : the assertion(s) created for this matching (class instantiations and/or property assertions)
– *linkedPreviousIndividuals* : the individual(s) *ind* representing the previous matching in the text such that there is an object property *property* such that the assertion <*ind*, *property*, *individuals*> exists.

At initialisation, the last three attributes (*individuals*, *assertions*, *linkedPreviousIndividuals*) are empty. More detail about each of these attributes is given in the following of the paper (whenever an attribute is modified).

Instantiation of the main class The next task consists in instantiating the main class. Indeed, a document describes an instance of the KG's main class. Therefore, a new individual is added to the KG to represent the document being processed. In our example, the individual *property*1 is created with the assertion <*property*1, *isA*, *Property*> (*Property* being the main class, representing a real estate property). If a document has matching(s) with the main class, i.e., if matchings concern the mention of the main class name (or of its associated terminology), then this assertion is added in the attribute *assertions* of the corresponding matching treatment(s), and the subject of the assertion is added in the attribute *individuals*. This is, for example, the case when an ad states something like "this property is ...", which is not very common but plausible. Algorithm 1 details this task. In the example from Figure 2, there is no mention of "property" in the text, so matching treatments are not updated at this stage.

In the following, this new instance of the main class is called the *main instance*. The next two tasks are relatively complex and are explained in the next two subsections.

Algorithm 1: Instantiation of the main class

```
Input:
– The knowledge graph $KG$
– The set $MT = \{mt_1, mt_2, \ldots, mt_n\}$ of matching treatments
Output: $KG$ and $MT$ are modified
  /* creation of an individual mainInstance and its class assertion */
1 assertion ←<mainInstance,isA,mainClass>
2 KG.add(assertion)
  /* update MT */
3 for $mt \in MT$ do
4     matching ← mt.getMatching()
5     if matching.concernsMainClass() then
6         mt.addAssertion(assertion)// will also add the subject of the assertion in mt.getIndividuals()
7     end
8 end
```

3.3.2. Text-based analysis

This task analyses the matchings and aims to add individuals as well as class and property assertions by considering the text, especially by taking into account the sentence splitting and the order of the matchings. It is detailed at the bottom of Figure 1.

Treatment of the class matchings In a first step, the matchings corresponding to a class from the KG are processed. Each matching concerning a class (except the main class) is analysed, with the goal of creating a new individual, instance of this class. The word(s) preceding the matching in the text are compared to a list of keywords expressing negation (for example, "no"), and the matching is ignored if a trace of negation is found. For example, in "no garage", the matching with "garage" is ignored, no instance of garage is created. If the word preceding the matching corresponds to a number, as many individuals as the number found are created (or only one if there is no number). The algorithm is given in Algorithm 2. Considering the example given in Figure 2, the update of the matching treatments (individuals and assertions added) can be observed on the rows of type "class" in Table 1. These individuals and assertions are added to the KG.

Algorithm 2: Treatment of the class matchings

```
Input:
– The knowledge graph $KG$
– The set $MT = \{mt_1, mt_2, \ldots, mt_n\}$ of matching treatments
Output: $KG$ and $MT$ are modified
1  for $mt \in MT$ do
2      $matching \leftarrow mt$.getMatching()
3      if matching.concernsClass() and !matching.concernsMainClass() then
           /* check if the word(s) before are contained in a given list of
              negation keywords                                          */
4          if matching.tracesOfNegationBefore() then
5              continue
6          end
7          $class \leftarrow matching$.getClass()
8          $nb \leftarrow matching$.intWordBefore()// we try to parse the word before into an
            int (otherwise it is 1)
9          for $i \leftarrow 1$ to $nb$ do
10             $assertion \leftarrow$<newIndividual,isA,class>
11             $KG$.add($assertion$)
12             $mt$.addAssertion($assertion$)// will also add the subject of the assertion
                in $mt$.getIndividuals()
13         end
14     end
15 end
```

Treatment of the individual matchings Then, the matchings of individuals from the KG are processed (cf. Algorithm 3). Matching treatments are updated by adding the concerned individuals. It can be observed, for the studied example, on the rows corresponding to the type "individual" in Table 1.

Algorithm 3: Treatment of the individual matchings

```
Input:
– The set $MT = \{mt_1, mt_2, \ldots, mt_n\}$ of matching treatments
Output: $MT$ is modified
1  for $mt \in MT$ do
2      $matching \leftarrow mt$.getMatching()
3      if matching.concernsIndividual() then
4          $mt$.addIndividual($matching$.getIndividual())
5      end
6  end
```

Table 1
Matching treatments of the example after dealing with class, individual and property matchings (translated from French to English)

Matching	Type	Individuals	Assertions
Cormelles-le-Royal	individual	Cormelles-le-Royal	
house	class	house1	<house1, isA, House>
located	object property		
min	data property	distance1	<distance1, isA, Distance> <distance1, minCar, 15>
city center	individual	city_center	
Caen	individual	Caen	
min walk	data property	~~distance1~~ distance2	~~<distance1, minWalk, 3>~~ <distance2, isA, Distance> <distance2, minWalk, 3>
shops	individual	shops	
schools	individual	schools	
m^2	data property	house1	<house1, area, 110>
land	class	land1	<land1, isA, Land>
m^2	data property	land1	<land1, area, 400>
ground floor	individual	groundFloor	
kitchen	class	kitchen1	<kitchen1, isA, Kitchen>
equipped	data property	kitchen1	<kitchen1, equipped, true>
living room	class	livingRoom1	<livingRoom1, isA, LivingRoom>
fireplace	class	fireplace1	<fireplace1, isA, Fireplace>
facing	data property	livingRoom1	<livingRoom1, exposure, south-west>
bedroom	class	bedroom1	<bedroom1, isA, Bedroom>
m^2	data property	bedroom1	<bedroom1, area, 15>
First floor	individual	firstFloor	
bedrooms	class	bedroom2 bedroom3	<bedroom2, isA, Bedroom> <bedroom3, isA, Bedroom>
bathroom	class	bathroom1	<bathroom1, isA, Sdb>
Close to	object property	property1	<property1, isCloseTo, publicTransport>
public transport	individual	publicTransport (property1 in linked previous individuals)	
Land	class	land2	<land2, isA, Land>
Fees : 4%	data property	property1	<property1, feePercentage, 4>

Treatment of the property matchings The following step concerns property matchings and is described in Algorithm 4. The goal is to create property assertions. To do this, the subject and the object of the assertion to create need to be established. A list of possible subjects is considered. To instantiate it, candidate matchings are browsed, starting from the one preceding the property matching being processed until the beginning of the considered sentence. As soon as a candidate matching treatment has individuals that belong to the domain of the property matched, these individuals constitute the list of possible subjects. For possible objects, it depends whether the property is an object or a data property. In the case of an object property, the matching that directly follows the mention of the property is considered. If such a matching exists, all the individuals of its matching treatment that are in the range of the property are considered as possible objects. In the case of a data property, there are several possibilities:

– If the property matching is on a unit, then the previous word is taken. For example, if the property *area in* m^2 has m^2 as unit, then the text "room of 15 m^2" has a matching between "m^2" and this property, which is instantiated with the value 15.

– If the property matching concerns a unit expression, then the mention of the text that corresponds to the variable is taken. For example, if the property *fee percentage* has *fees: xxx%* as unit expression, then the text "fees: 4%" has a matching on this property, instantiated with the value 4.
– If the property is boolean, then false is taken if there is a trace of negation before or true otherwise. The same list of negation keywords is used as in the treatment of class matchings. For example, the boolean property *convertible* is instantiated with true for the text "the attic is convertible" and false for "the attic is not convertible".
– If the property range is a list of choices, i.e., an expression using *owl:OneOf*, then the next word(s) in the text are compared with the possible choices. The corresponding choice is taken if it exists. For example, if the property *exposure* has "north", "south", "east", "west" as possible values, then the text "facing south", having a matching on "facing" with the property *exposure*, leads to an instantiation with the value "south".
– In other cases, the next word is taken. For example, the text "property built in 2005" instantiates the property *built in* with the value 2005.

In any case, for each possible subject and object, an assertion of the considered property is added, provided that this one does not create an inconsistency into the KG. The matching treatment is updated: the new assertion(s) and their subject individual(s) are added respectively to the attributes *assertions* and *individuals*. The matching treatment of the matching representing the object is also updated: the subject of the assertion is added in its attribute *linked previous individuals*. If no assertion can be added, then the process is repeated with a new instance of the property domain as subject. Once all property matchings are processed, any individuals in the KG that are recognized as equivalent are merged, and *MT* is updated w.r.t. this merger.

Let us explain a bit more line 18 from Algorithm 4. The idea is that, if the assertion already exists before the matching is processed, we do not want this assertion to be linked with this matching, since there is probably another missing assertion. This is particularly important for ternary properties. For example, in the sentence "20 min from X and 20 min from Y", the assertion <*distance*1, *minCar*, 20> is created when the matching about the first "min" is treated. In that case, *distance1* is a new instance of *Distance*, the domain of the property *minCar*, and represents the distance from X. When the matching about the second "min" is being treated, the same assertion is considered but is ignored. The process continues: a new assertion <*distance*2, *minCar*, 20> is created, *distance2* being a new instance of *Distance*, representing the distance from Y.

Table 1 shows the treatment of the property matchings on the example (cf. rows whose type is a property). First, the mention "located" refers to the object property *is located in* whose range is a municipality. It is followed by a numerical value and not by a matching corresponding to a municipality. As a result, the set of possible objects is empty and the property cannot be instantiated. The matching on "min" refers to the property *minCar* associating a distance to a numerical value. It is on a unit, so we consider the previous word as the object: 15. As there is no instance of the class *Distance* at this stage, the set of possible subjects is initially empty, and a new instance of *Distance* is considered, which is named *distance*1. For the matching on "min walk", the process is the same, except that the set of possible subjects considers *distance*1 since it is an individual resulting from a matching preceding the one being processed, from the same sentence, and in the property domain. We therefore try to add an assertion <*distance*1, *minWalk*, 3> but this one is inconsistent (because this distance already represents 15 min by car). Thus, a new individual *distance*2 is created for the subject. For the three mentions "m^2", the possible objects are the previous words, because the matching is on the unit of the property *area*. For the possible subjects, we look for individuals belonging to its domain. The domain of *area* is the union of classes denoting a part of property or a room. For the part about 110 m^2, we have to go back quite far in the sentence to find such an individual (*house*1), for the parts about respectively 400 and 15 m^2, the individual from the previous matching (respectively *land*1 and *bedroom*1) is directly suitable. For the mention "equipped", the subject matches the previous matching ("kitchen") and, as the property is boolean and there is no trace of negation, the object is *true*. For "facing", the subject is *livingRoom*1 (the fireplace not being part of the property domain), and the object is the next word ("southwest"), which belongs to the list (*owl:oneOf*) of possible ranges. Finally, for the last two property matchings ("close to" and "Fees: 4%"), the first set of possible subjects is empty. The domain of these two properties being the main class (*Property*), the main instance (*property*1) is considered as subject. For the object, we consider respectively the matching that directly follows ("public transport") and the value of the unit expression (4). Since the matching

Algorithm 4: Treatment of the property matchings

```
Input: The knowledge graph $KG$ and the set $MT = \{mt_1, mt_2, \ldots, mt_n\}$ of matching treatments
Output: $KG$ and $MT$ are modified
1  for $mt \in MT$ do
2      $matching \leftarrow mt$.getMatching()
3      if $matching.concernsProperty()$ then
4          $prop \leftarrow matching$.getProperty()
5          $domain \leftarrow prop$.getDomain()
6          $range \leftarrow prop$.getRange()
7          $possibleSubjects \leftarrow$ getClosestMatchedInstancesBeforeInDomain($matching$,$domain$,$MT$)// We
            look at the matching treatments before matching (by going backwards
            to the beginning of the sentence). If a matching treatment has
            individuals ∈ domain, we stop and get these individuals.
8          $possibleObjects \leftarrow$ getPossibleObjects($matching$,$range$,$MT$,$KG$)// 2 cases. In case of
            object property: we search if there is a matching directly after
            the matching mention. If there is, we get all its individuals from
            its treatment that are in range. In case of data property:
            different treatments following range and type of matching.
9          if $possibleObjects.isEmpty()$ then
10             return// we do nothing
11         end
12         for $i \leftarrow 1$ to $2$/* 1 for an existing subject, 2 for a new subject     */ do
13             $atLeastOneAddition \leftarrow false$
14             for $subject \in possibleSubjects$ do
15                 for $object \in possibleObjects$ do
16                     $assertion \leftarrow <subject,prop,object>$
17                     if $assertion \in KG$ then
18                         continue// important for ternary properties
19                     end
20                     $addition \leftarrow KG$.addIfConsistent($assertion$)
21                     if $addition$ then
22                         $mt$.addAssertion($assertion$)
23                         $object$.getMatchingTreatment().addLinkedPreviousIndividual($subject$)
24                         $atLeastOneAddition \leftarrow true$
25                     end
26                 end
27             end
28             if $i$=1 and $atLeastOneAddition$ then
29                 break// no need to try a new subject
30             end
31             if $i$=1 and !$atLeastOneAddition$ then
32                 $possibleSubjects \leftarrow$ domainInstantiation($domain$)// return a new instance of
                    domain (or the main instance if domain is the main class)
33             end
34         end
35     end
36 end
37 $KG$.mergeSameAsIndividuals($MT$)// merges all sameAs individuals in KG and
    update MT w.r.t. the merger
```

with "Close to" leads to the assertion <$property1, isCloseTo, publicTransport$>, $property1$ is added as a linked previous individual in the matching treatment of "public transport".

Let us note that new individuals are named according to the class they are instances of (e.g., $distance1$ and $distance2$ for the class *Distance*). If the domain or range to instantiate is not a named class but a class expression, then new individuals are named $indiv1$, $indiv2$, etc.

Initialisation of the linkabilities Thanks to these first three steps (treatments of class/individual/property matchings), some information can be extracted. However, as mentioned in Section 2, verbs, in our context, are not very meaningful, or are even absent. It is thus very likely to miss property assertions, due to an absence of property matching. The remaining of the population algorithm aims at adding property assertions. The challenge here is to add missing assertions (which would increase recall) without adding too many false assertions (which would decrease precision). In this context, we introduce an element that we call linkabilities (*Lnk*), which is exploited not only in the last two steps of this process but also in the following task (knowledge-based analysis). The initialisation of the linkabilities *Lnk* consists in creating for each individual i from MT, the sorted set of its linkabilities $Lnk(i)$. An element of $Lnk(i)$ is a pair *(property, range class expression)*, such that i is linkable, via the property, to an individual belonging to the range class expression. In other words, for an individual i, we look for each property $prop$ for which i can be a subject. If i can be a subject of $prop$, we look for the range expression to which an object obj of a possible assertion <i,$prop$,obj> must necessarily belong. More precisely, the following is performed:

First, for each class c from the KG, definitions, expressed via an equivalence or via a superclass expression, with an object property p and a universal quantifier (owl:allValuesFrom) are considered. In other words, these definitions are of the form "`p ONLY classExpression`" in the OWL Manchester syntax. If such definitions exist, the class c, the object property p mentioned in its definition, and the class expression $classExpression$ from this definition are kept, in a triple that we call $linkWithOnly(c, p, classExpression)$ in the following. If there are several definitions for a same class and a same property, then the intersection of the class expression is taken.

Then, for each individual $i \in MT$:

1. Classes to which i belongs (including superclasses) are considered.
2. For each class c to which i belongs, and each object property p from the KG, the $classExpression$ from $linkWithOnly(c, property, classExpression)$ is considered, if it exists.
3. For each object property in the KG, if i belongs to its domain, then a linkability is established. The range class expression of this linkability is the intersection of the property range and the potential $classExpression$ from 2.

For instance, let us consider the main instance $property1$ of our studied example and its linkability regarding the object property *contains*. The individual $property1$ belongs to the class *Property* and its superclass *owl:Thing*. The class *Property* is, among other things, a subclass of `contains ONLY PartOfProperty`. This means $linkWithOnly(Property, contains, PartOfProperty)$ is considered. The individual $property1$ belongs to the domain of *contains*: a linkability is established. The range class expression is $PartOfProperty$ because it is the intersection of the range of *contains* (i.e., `PartOfProperty or Rooms`) and the class $PartOfProperty$ (coming from the $linkWithOnly$). In other words, the pair ($contains$, $PartOfProperty$) is a linkability for $property1$.

For each individual i, the set of linkabilities $Lnk(i)$ is sorted. The first elements are the linkabilities whose range is the most specific, then those whose property domain is the most specific. Otherwise, the sorting is arbitrarily, according to the property URI. For example, if an instance of *Property* has for linkability l_1 = (*isLocatedIn*, *Municipality*) and l_2 = ($isCloseTo$, *Location*) such that *Municipality* is a subclass of *Location*, then they will be sorted in the order $l_1 < l_2$. The intuition behind this sorted set $Lnk(i)$ is that it will be used in the next steps to link i with an other individual j. The set will be browsed until a linkability is found such that i is linkable to j, in other words, such that j belongs to the range of the linkability. In case there are several candidate properties, we want to find the best one, which is not an obvious task. We chose to give priority to specificity. Hence, if a link has to be added between $property1$ and an instance of municipality, the chosen property will be *isLocatedIn* and not $isCloseTo$, since the range of l_1 is more specific than the one on l_2.

In the example of this paper, the individual $property1$ has sorted linkabilities: (*isLocatedIn*, *Municipality*), ($isCloseTo$, *Location*), ($isSituatedAtADistance$, *Distance*), ($contains$, $PartOfProperty$). The initialisation of the linkabilities creates linkabilities for all the individuals mentioned in Table 1.

Treatment of the next matching Once the linkabilities are created, they are exploited in the treatment of the next matching. This treatment consists in checking if it is possible to link the individual(s) resulting from a matching with the one(s) from the following matching in the text, coming from the same sentence. Property assertions are added whenever it is possible. The algorithm also tries to consider n-ary properties (especially ternary ones). For instance, a succession of matchings involving respectively individuals *a*, *b* and *c* may lead to property assertions of the type <*a*,...,*b*> and <*a*,...,*c*>. In this example, the goal is to link *a* and *b*, which are from consecutive matchings in the text, but also *a* and *c*, which are not. To be able to do this, linked previous individuals are exploited.

The treatment is described in Algorithm 5 and exploits Algorithms 6 and 7. The general idea is to look if an individual (*subject*), coming from the current matching being examined, may be the subject of an assertion having for object an individual from the next matching (*object*). If the next matching has no associated individuals, then the matching after si considered and so on. If *subject* and *object* are not already connected together, we suppose an assertion might be missing. We want to add an assertion <*subject*, *property*, *object*> such that *property* will be the best possible property. To choose it, we consider the sorted set of linkabilities of the subject and take the first property from a linkability such that *object* is in the range expression of the latter and such that it does not add any inconsistency to the KG. The assertion is added to the matching treatment of *subject*, and *subject* is also added as a linked previous individual of the object matching treatment. If no assertion is possible between the individuals of two consecutive matchings, then we try to make an assertion with the linked previous individuals as subject, in order to facilitate the population of the n-ary properties. Once all the text is examined, any individuals in the KG that are recognized as equivalent are merged, and *MT* is updated w.r.t. this merger.

Algorithm 5: Treatment of the next matching

```
Input:
– The knowledge graph $KG$
– The set $MT = \{mt_1, mt_2, \dots, mt_n\}$ of matching treatments
– The set $Lnk$ of linkabilities
Output: $KG$ and $MT$ are modified
1  for $mt \in MT$ do
2      $link \leftarrow false$
3      for $subject \in mt.getIndividuals()$ do
4          $link \leftarrow$ linkIndividualWithNextMatching($subject, mt, KG, MT, Lnk(subject)$) or link
5      end
6      if $!link$ then
           // If no assertion is possible between $subject$ and at least one
           individual from the next matching, then we look at the list of
           linked previous individuals (in order), and we stop as soon as an
           assertion is made: this is useful for n-ary properties.
7          for $linkedPrevIndiv \in mt.getLinkedPreviousIndividuals()$ do
8              $link \leftarrow$ linkIndividualWithNextMatching($linkedPrevIndiv, mt, KG, MT, Lnk(linkedPrevIndiv)$)
9              if $link$ then
10                 break
11             end
12         end
13     end
14 end
15 $KG$.mergeSameAsIndividuals($MT$)
```

Table 2 shows the matchings treatments after this step. First, we try to connect *Cormelles-le-Royal* to *house1* (impossible), then *house1* to *distance1* (impossible) and so on. We can connect *distance1* to *city center* via the property *distanceFromPointOfInterest* (1). The assertion is added in the subject (*distance1*) matching treatment, i.e.,

Algorithm 6: Function linkIndividualWithNextMatching

```
Input:
– The individual $subject$ to be used as the subject
– The matching treatment $mt$ from which $subject$ is considered
– The knowledge graph $KG$
– The set $MT = \{mt_1, mt_2, \ldots, mt_n\}$ of matching treatments
– The sorted set $Lnk(subject)$ of linkabilities for the individual $subject$
Output: boolean indicating if there is an assertion linking $subject$ to an individual from the next matching.
        The assertion may exist before or be created within the function. (+ $KG$ and $MT$ are modified)
1  $nextMT \leftarrow$ getClosestNextMatchingTreatmentWithinTheSameSentenceAndContainingIndividuals($mt$,$MT$)
2  $nextIndividuals \leftarrow nextMT$.getIndividuals()
3  $link \leftarrow false$
4  for $nextIndividual \in nextIndividuals$ do
5      $addedLink \leftarrow false$
6      if $KG$.alreadyLinked($subject$,$nextIndividual$) then
7          $link \leftarrow true$ // Assertion <$subject$,any_property,$nextIndividual$> already exists
8      else
9          $addedLink \leftarrow$ tryToLinkSubjectWithObject($subject$,$nextIndividual$,$KG$,$Lnk(subject)$, $mt$)
10         if $addedLink$ then
11             $nextMT$.addLinkedPreviousIndividual($subject$) // add $subject$ to the list of
                 linked previous individuals of $nextMT$, only if $subject$ is neither
                 already in it nor in the set of individuals of $nextMT$.
12         end
13     end
14     $link \leftarrow addedLink$ or $link$
15 end
16 return $link$
```

Algorithm 7: Function tryToLinkSubjectWithObject

```
Input:
– The individual $subject$ to be used as the subject
– The individual $object$ to be used as the object
– The knowledge graph $KG$
– The sorted set $Lnk(subject)$ of linkabilities for the individual $subject$
– (optional) $mt$ the matching treatment from which $subject$ is considered
Output: boolean indicating if an assertion between $subject$ and $object$ is created (+ $KG$ and $MT$ are
        modified)
1  $assertion \leftarrow$<$subject$,bestAddableProperty($Lnk(subject)$),$object$> // The best property is the
     first one from $Lnk(subject)$ s.t. the object is in its range expression and
     s.t. the ontology is still consistent with this new assertion.
2  if $assertion$ != $null$ then
3      $KG$.add($assertion$)
4      $mt$.addAssertion($assertion$) // only if $mt$ is given as input
5      return $true$
6  end
7  return $false$
```

Table 2
Matching treatments of the example after the treatment of the next matching

Matching	Individuals	Assertions	Linked previous individuals
Cormelles-le-Royal	Cormelles-le-Royal		
house	house1	<house1, isA, House>	
located			
min	distance1	<distance1, isA, Distance> <distance1, minCar, 15> <distance1, distFromPI, city center> (1)	
city center	city center	<distance1, distFromCity, Caen> (2)	distance1 (1)
Caen	Caen		distance1 (2)
min walk	distance2	<distance2, isA, Distance> <distance2, minWalk, 3> <distance2, distFromPI, shops> (3)	
shops	shops	<distance2, distFromPI, schools> (4)	distance2 (3)
schools	schools		distance2 (4)
m^2	house1	<house1, area, 110>	
land	land1	<land1, isA, Land>	
m^2	land1	<land1, area, 400>	
ground floor	groundFloor		
kitchen	kitchen1	<kitchen1, isA, Kitchen>	
equipped	kitchen1	<kitchen1, equipped, true>	
living room	livingRoom1	<livingRoom1, isA, LivingRoom> <livingRoom1, hasForElement, fireplace1> (5)	
fireplace	fireplace1	<fireplace1, isA, Fireplace>	livingRoom1 (5)
facing	livingRoom1	<livingRoom1, exposure, south-west>	
bedroom	bedroom1	<bedroom1, isA, Bedroom>	
m^2	bedroom1	<bedroom1, area, 15>	
First floor	firstFloor		
bedrooms	bedroom2 bedroom3	<bedroom2, isA, Bedroom> <bedroom3, isA, Bedroom>	
bathroom	bathroom1	<bathroom1, isA, Bathroom>	
Close to	property1	<property1, isCloseTo, publicTransport>	
public transport	publicTransport		property1
Land	land2	<land2, isA, Land>	
Fees : 4%	property1	<property1, feePercentage, 4>	

in the one on the mention “min”. The next matching treatment (on the mention “city center”) is updated with the linked previous individual *distance1*. Then, when we try to link the matching treatment on “city center” with the next one (on “Caen”), we cannot link the associated individuals, i.e., we cannot link *city center* to *Caen*. Nevertheless, we can link the linked previous individual of the “city center” matching treatment, i.e., *distance1*, to *Caen*. This is what is done in (2). And so on, we obtain (3), (4) and (5).

Treatment of the individuals without predecessors The last step of the text-based analysis consists in dealing with individuals having no predecessors. Indeed, each document describes an instance of the main class and we expect this instance to be the starting point of property assertions. Therefore, it seems quite intuitive to think that every individual considered, except the main instance, must be the object of at least one assertion. Thus, in this last step, we try to find possible subjects and properties for individuals (except for the main instance) having for the moment no predecessors, i.e., not being the object of a property assertion. To minimise the risk of linking individuals that have nothing to do with each other, we focus only on the individuals resulting from the matchings of a same sentence. This means that, for each individual without predecessors (*object*) of a sentence, we try to connect another individual of this sentence (*subject*) to it, in order to obtain the assertion <*subject*, *prop*, *object*>, where the property *prop* chosen is the "best" in the sense of the linkabilities, just like in the previous step. Finally, any individuals in the KG that are recognized as equivalent are merged, and *MT* is updated w.r.t. this merger. Details are given in Algorithm 8 that exploits Algorithm 7.

Algorithm 8: Treatment of the individuals without predecessors

```
Input:
– The knowledge graph $KG$
– The set $MT = \{mt_1, mt_2, \ldots, mt_n\}$ of matching treatments
– The set $Lnk$ of linkabilities
Output: $KG$ and $MT$ are modified
1  $individualsWithoutPred \leftarrow MT$.getIndividualsWithoutPredecessors()
2  $individualsWithoutPred$.remove($KG$.getMainInstance())
3  for $sentenceNumber \leftarrow 1$ to $MT$.getNumberOfSentences() do
       /* get the matching treatments for sentence #$sentenceNumber$ */
4      $sentenceMatchingTreatments \leftarrow MT$.getMatchingTreatmentsFromSentence($sentenceNumber$)
       /* get all individuals resulting from sentence #$sentenceNumber$ */
5      $allIndividualsFromSentence \leftarrow \emptyset$
6      for $mt \in sentenceMatchingTreatments$ do
7          $allIndividualsFromSentence$.addAll($mt$.getIndividuals())
8      end
       /* beginning of the process: try to add assertions linking an
          individual from sentence #$sentenceNumber$ (as subject) to an other
          individual from the same sentence without predecessors (as object)
          */
9      for $mt \in sentenceMatchingTreatments$ do
10         $sentenceIndividualsWithoutPred \leftarrow mt$.getIndividuals() $\cap$ $individualsWithoutPred$
11         for $object \in sentenceIndividualsWithoutPred$ do
12             for $subject \in allIndividualsFromSentence \setminus object$ do
13                 tryToLinkSubjectWithObject($subject$,$object$,$KG$, $Lnk(subject)$, $mt$)
14             end
15         end
16     end
17 end
18 $KG$.mergeSameAsIndividuals($MT$)
```

On Figure 3, we can observe all the individuals and property assertions added before this step for the studied example. Individuals without predecessors (except the main instance *property*1) are shaded. They are potential object candidates. Table 3 shows the treatment done at this stage. Each row of the table corresponds to a sentence. Individuals without predecessors are in italics. For each of them, we search if it is possible to add an assertion having for subject an individual of the same sentence. For example, for *Cormelles-le-Royal*, we search if we can add assertions <*house*1,...,*Cormelles-le-Royal*>, <*distance*1,...,*Cormelles-le-Royal*>, etc. The only two possibilities for the first sentence are given in the last column but they are not added because they are inconsistent with respect to the KG. Indeed, *distance*1 and *distance*2 already concern other places and cannot concern *Cormelles-le-Royal*. There are no possibilities for the other object candidates from this sentence (*house*1, *distance*1, *distance*2, *land*1). The same is repeated for each sentence. The assertions mentioned in Table 3 are obtained. They are added in the KG and in the MT.

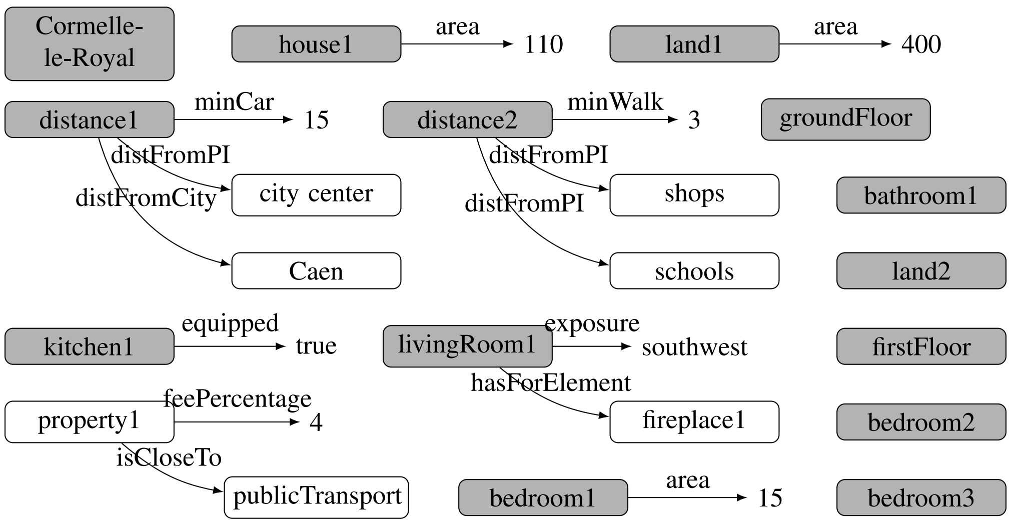

Fig. 3. Individuals and property assertions from the studied example, before the treatment of the individuals without predecessors

Table 3

The treatment of the individuals without predecessors in the studied example

Sentence number	Individuals from the sentence (*object candidates in italics*)	Assertions added (<individual_of_the_sentence, property, *object*>)
1	*Cormelles-le-Royal*, *house1*, *distance1*, city center, Caen, *distance2*, shops, schools, *land1*	~~<distance1, distFromCity, *Cormelles-le-Royal*>~~ ~~<distance2, distFromCity, *Cormelles-le-Royal*>~~ not added because inconsistent
2	*groundFloor*, *kitchen1*, *livingRoom1*, fireplace1, *bedroom1*	<kitchen1, isOnFloor, *groundFloor*> <livingRoom1, isOnFloor, *groundFloor*> <bedroom1, isOnFloor, *groundFloor*>
3	*firstFloor*, *bedroom2*, *bedroom3*, *bathroom1*	<bedroom2, isOnFloor, *firstFloor*> <bedroom3, isOnFloor, *firstFloor*> <bathroom1, isOnFloor, *firstFloor*>
4	property1, publicTransport	
5	*land2*	
6	property1	

3.3.3. *Knowledge-based analysis*

Once all the steps from the text-based analysis are performed, the next (and last) task of the population algorithm is based on an analysis of the knowledge from the KG. This exploits the KG, the MT and the linkabilities. The idea of this analysis is that, despite the efforts in the previous steps to add property assertions, it is possible that some are still missing. Ideally, from the main instance, all the individuals concerned by the document should be reachable via property assertions. Figure 4 shows, among other things, the set of individuals (nodes) and object property assertions (edges) from the studied example. We can see that, from the main instance $property1$, we can only access one individual, $publicTransport$. Our goal is to add assertions of object properties in order to obtain a graph in the shape of a star, starting from $property1$.

Algorithm 9 explains the process used (using Algorithm 7). It exploits a process of batch splitting detailed in Algorithms 10 and 11. The individuals and assertions are put into batches, each individual being in exactly one batch. Each batch is created from an individual *i*, and contains all the individuals that are reachable from *i* (directly or via a sequence of assertions). The first batch created is called the *main batch*. It is composed of the main instance and all the individuals accessible through it, as well as the assertions linking them. Then, all the remaining assertions

Algorithm 9: Knowledge-based analysis

```
Input:
– The knowledge graph $KG$
– The set $MT = \{mt_1, mt_2, \ldots, mt_n\}$ of matching treatments
– The set $Lnk$ of linkabilities
Output: $KG$ and $MT$ are modified
1  $distance \leftarrow 0$// distance from the main instance
2  do
3      $individuals \leftarrow MT.getAllIndividuals() \cup KG$.getMainInstance()// the main instance has
         to be added in case it is not mentioned in $MT$
4      $assertions \leftarrow KG$.getAllObjectPropertyAssertionsUsingReasoningBetween($individuals$)
5      $setOfBatches \leftarrow$ creationOfBatches($assertions$, $individuals$, $KG$.getMainInstance())
         // Individuals are gathered into batches. One individual can only be
         in one batch. We call mainBatch the batch containing the main
         instance.
6      $individualsFromMainBatchToBeLinked \leftarrow setOfBatches$.getIndividualsFromDistance($distance$)
         // set of all the individuals that are at a distance distance from the
         main instance (main instance if $distance = 0$). This is the set of
         subject candidates.
7      if setOfBatches.size()=1 or individualsFromMainBatchToBeLinked.isEmpty() then
           // stop case: either there is only one batch (no need to add any
           new assertion), or we already tried to link every individual from
           the main batch
8          break
9      end
       /* try to link the subject candidates to object candidates from all
          batches except the main batch                                    */
10     for $subject \in individualsFromMainBatchToBeLinked$ do
11         for $batch \in setOfBatches \setminus mainBatch$ do
12             $objects \leftarrow batch$.getIndividualsSortedByNumberOfPredecessors()
13             for $object \in objects$ do
14                 $addition \leftarrow$ tryToLinkSubjectWithObject($subject$,$object$,$KG$,$Lnk(subject)$)
15                 if addition then
16                     break// we manage to link subject to one individual from batch,
                        we do not look at other individuals from batch
17                 end
18             end
19         end
20     end
21     $distance \leftarrow distance + 1$
22     $KG$.mergeSameAsIndividuals($MT$)
23 while true
```

are considered. Among their subject and object individuals, the one that has the least predecessors is taken; the alphabetical order of URI is chosen in case of equality. Its batch is created, and the process continues until every assertion has been treated. Finally, every individual left is put respectively into a new batch. Once all batches are built, the first goal is to link the main instance to one individual of each batch (except the main batch). It will help to get the “star-shaped” form. This linking can be seen as risky because no connection was made in the previous steps of the population algorithm. This is why we try to link the main instance with only one individual from a same batch. This is safer to avoid to add too many noisy assertions. Inside a batch, individuals are sorted according to their number of predecessors (smallest number first), or alphabetically in case of equality. For every batch (except the main batch), we try to make the connection following that order, and stop as soon as a property assertion can be made between the main instance (as subject) and an individual (as object) from the batch being processed. Then, any individuals in the KG that are recognized as equivalent are merged, and *MT* is updated w.r.t. this merger. The process (creation of batches, try to add assertions, merger) is repeated, but considering as subjects all individuals that are at a distance 1 of the main instance, i.e., all individuals for which there is an assertion between the main class and them. In other words, the distance of an individual *i* can be defined as the minimal number of edges between the main instance and *i*, i.e., the minimal number of assertions to go from the main instance to *i*, regardless of the direction of the assertions. This distance is progressively incremented. We stop either when the main batch contains all individuals or when we have already attempted to link every individual from the main batch.

Algorithm 10: Function creationOfBatches

```
Input:
– A set $assertions$ of object property assertions
– A set $individuals$ of individuals
– The main instance $mainInstance$

Output: A set of batches
1  $setOfBatches \leftarrow \emptyset$
2  $remainingAssertions \leftarrow assertions$
   /* Creation of the main batch                                      */
3  $mainBatch \leftarrow$ createBatchFromIndividual($mainInstance$,$remainingAssertions$)
4  $setOfBatches$.add($mainBatch$)
   /* Creation of other batches                                       */
5  while $!remainingAssertions.isEmpty()$ do
6      $indiv \leftarrow remainingAssertions$.getIndividualsSortedByNumberOfPredecessors().first()
       // We take the individual with the smallest number of predecessors
       (sorted by IRI in case of a same number)
7      $setOfBatches$.add(createBatchFromIndividual($indiv$,$remainingAssertions$))
8  end
   /* Creation of a batch for each remaining individual               */
9  $treatedIndividuals \leftarrow setOfBatches$.getAllIndividuals()
10 for $ind \in individuals \setminus treatedIndividuals$ do
11     $setOfBatches$.add(new Batch($ind$))
12 end
13 return $setOfBatches$
```

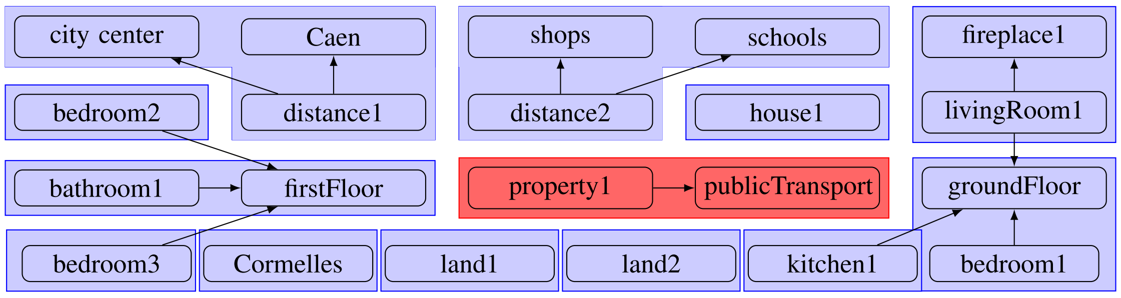

Fig. 4. First division into batches of the studied example

Algorithm 11: Function createBatchFromIndividual

```
Input:
– The individual ind from which the batch has to be created
– A set assertions of object property assertions that are not considered in any batch yet
Output: A batch containing ind, every individual reachable from ind, and all the assertions between them (+
        assertions is modified)
   /* Creation of the batch                                                         */
1  batch ← new Batch(ind)
2  change ← true
3  while change do
4      change ← false
5      for assertion ∈ assertions do
6          if batch.contains(assertion.getSubject()) then
7              batch.add(assertion)// add the assertion, its subject and its object
8              change ← true
9              assertions.remove(assertion)
10         end
11     end
12 end
   /* Update of assertions: assertions that have an object in batch are removed
      from the remaining assertions to deal with                                    */
13 for assertion ∈ assertions do
14     if batch.contains(assertion.getObject()) then
15         assertions.remove(assertion)
16     end
17 end
18 return batch
```

In the studied example, the first division into batches is represented in Figure 4 via frames. The *main batch* is composed of *property*1, *publicTransport* and the assertion between them. Afterwards, among the remaining assertions, the individual that has the least predecessors (first in the alphabetical order) is *bathroom*1. The next batch is thus made of *bathroom*1, *firstFloor* and the assertion between them. The assertion between *bedroom*2 (resp. *bedroom*3) and *firstFloor* is ignored because *firstFloor* is already placed in a batch (cf. line 15 of Algorithm 11). Then, the next individual that is considered is *bedroom*1. The next batch is composed of *bedroom*1, *groundFloor*, as well as the assertion between them. The assertion between *kitchen*1 (resp. *livingRoom*1) and *groundFloor* is ignored. The next batches include *distance*1 and its two successors, then *distance*2 and its two successors, then *livingRoom*1 and its successor *fireplace*1. The remaining individuals are each in one batch.

We consider a distance from the *main instance*, which will be incremented as we go along. We want to connect the individuals from the *main batch* that are at this distance from the *main instance*. At first, the distance is 0, so we try to connect the *main instance* to each of the other batches. Let us take for example the sorted batch (*distance*2, *schools*, *shops*), where *distance*2 is first because it has no predecessors. We check, using the linkabilities, if the main instance *property*1 can be linked to *distance*2. It is possible here via the property *isSituatedAtDistance*. If it had not been possible, we would have tried to link the main instance *property*1 with the next individual of the batch: *schools*. We do the same for each batch. This allows us to add six assertions visible on Figure 5: <*property*1, *isSituatedAtDistance*, *distance*1>, <*property*1, *isSituatedAtDistance*, *distance*2>, <*property*1, *contains*, *house*1>, <*property*1, *isLocatedIn*, *Cormelles-le-Royal*>, <*property*1, *contains*, *land*1> and <*property*1, *contains*, *land*2>. Afterwards, the equivalent individuals are merged. Since the KG expresses the fact that a property can contain only one land, *land*1 and *land*2 are seen as equivalent by a reasoner. They are merged

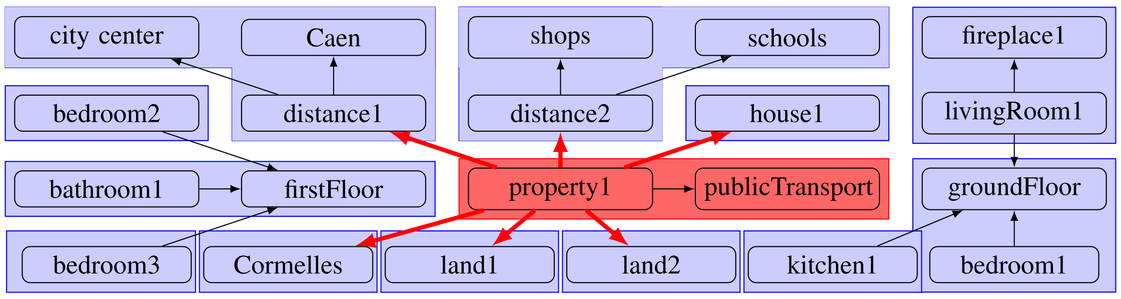

Fig. 5. Additions with distance 0 in the studied example

into *land*1. Then, the batch splitting algorithm is reapplied, leading to the batches of Figure 6. The main batch is bigger than the first time, and there is only six other batches. We now try to link to other batches (as objects) the elements of the main batch that are at a distance of 1 from the main class (as subjects), i.e., *Cormelles-le-Royal*, *distance*1, *distance*2, *house*1, *land*1 and *publicTransport*. The algorithm allows us to obtain <*house*1, *contains*, *bathroom*1>, <*house*1, *contains*, *bedroom*1>, <*house*1, *contains*, *bedroom*2>, <*house*1, *contains*, *bedroom*3>, <*house*1, *contains*, *kitchen*1> and <*house*1, *contains*, *livingRoom*1>. Afterwards, there is only one batch left; in other words, all elements are accessible starting from the main instance. The process is therefore stopped.

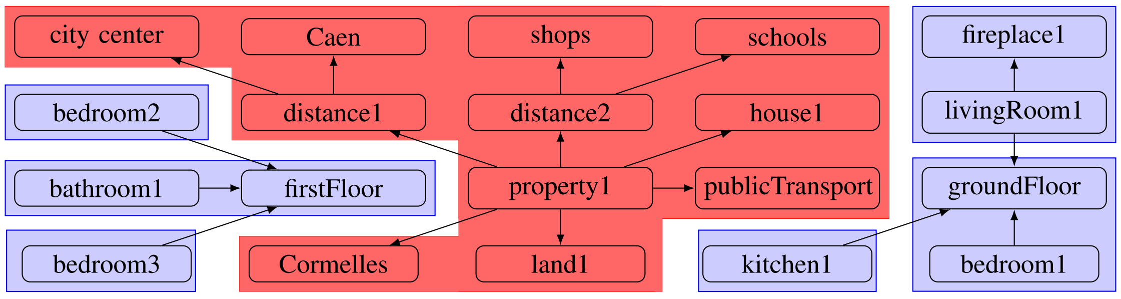

Fig. 6. Second division into batches of the studied example

The example unfolded in this paper is an illustration where our algorithm populates the KG with the desired individuals and assertions. More generally, the algorithm can generate wrong individuals and/or assertions and miss correct ones. The following section details some evaluation results of the algorithm.

4. Evaluation

This section reports an evaluation of the proposed approach on one use case: house sale classified ads. The experimental protocol, the tools used and the results obtained are discussed.

4.1. Experimental protocol

The KOnPoTe approach has been tested on a corpus automatically extracted from a website[5]. This corpus focuses only on ads annotated as sales of a house in Caen. It has been cleaned: the ads that do not really correspond to standard sales of houses (like mills, buildings, etc.) have been removed. It contains 78 ads, written in French. All information from the ads (date of extraction, seller's name, annotations, etc.) have been extracted in the XML format, but we focus only on the textual description of each ad. A knowledge graph (KG) has been built. It describes the

[5]https://www.lecoindelimmo.com/

domain of house sales, with a French terminology, in the OWL language, and exploits the *unit* annotation to express units and unit expressions, as mentioned in Section 3.1. It initially contains a few generic individuals, such as *double glazing*, *public transport*, etc., as well as named entities corresponding to city or village names. On a large scale, we would need a large list of cities, but for this experiment, we decided to represent only those mentioned in the corpus. A Gold Standard knowledge graph (GS) has been constructed. This GS is the initial KG manually populated with assertions representing the descriptions from the corpus. We applied the approach on the initial KG and each description from the corpus, and we sought to compare the obtained KG with the GS, computing precision, recall and F-measure.

$$\mathit{Precision} = \frac{TP}{TP + FP} \qquad \mathit{Recall} = \frac{TP}{TP + FN} \qquad \mathit{F\text{-}measure} = \frac{2 \times \mathit{Precision} \times \mathit{Recall}}{\mathit{Precision} + \mathit{Recall}}$$

In order to compute these metrics, we define property assertions as true positives (TP), false positives (FP) or false negatives (FN). A TP is an assertion present in both the GS and the output KG. A FP is an assertion present in the output KG but absent in the GS. A FN is an assertion present in the GS but absent in the output KG. To be as fair as possible in our results:

(i) class assertions are not taken into account (only property assertions);
(ii) inferred assertions are taken into account;
(iii) properties that lead us to count several times a same element are ignored.

Indeed, (i) class assertions are often redundant with property assertions. For example, with the assertion <$\mathit{distance}1$, $\mathit{distanceFromCity}$, *Caen*>, we know via the domain definition of $\mathit{distanceFromCity}$ that $\mathit{distance}1$ belongs to the class *Distance*. (ii) We consider the set of assertions obtained directly and those obtained by applying a reasoner. Otherwise, the comparison would not make sense. For example, if the GS contains the assertion <*a*, *prop*, *b*> and the resulting KG contains <*b*, prop^{-1}, *a*>, prop^{-1} being the inverse of the property *prop*; then we want to be able to realize that these two assertions amount to the same thing. (iii) We ignore properties that lead us to count several times a same element. For example, if there is a property and its inverse, a reasoner translates an assertion of one into an assertion of the other. We end up with two assertions designating the same thing. One of the two properties is therefore ignored; in other words, all assertions of this property are ignored. As another example, we may cite the case where a super-property is never populated by itself but only through its sub-properties. In that case, we ignore this property, so that a correct (resp. wrong) assertion does not count twice, since inferred assertions are considered.

The studied example from the previous section only considers one ad. In our experiments, since several ads are studied, names of individuals are actually *advertisementX_IndividualName* (*X* being the ad number in the corpus). Note also that KOnPoTe creates some URIs that are not necessarily the same as the ones from the GS. For example, $\mathit{advertisement}61\_\mathit{House}1$ corresponds to $\mathit{advertisement}61\_\mathit{Townhouse}1$ in the GS. To be able to correctly classify an assertion into TP, FP or FN, we manually define equivalences between GS individuals and result individuals representing the same thing. Moreover, our approach may generate equivalent individuals, without detecting that they are equivalent. In this case, we suppose an equivalence axiom (owl:sameAs) is missing and count it as a missing assertion (thus a FN). This avoids counting many wrong errors. For example, let us suppose an individual *i* is subject of 10 assertions in the GS, and is seen as two individuals *i* and *j* in our approach results, each respectively subject of 5 of these 10 assertions. This allows us to consider 10 TP (all the assertions) and 1 FN (the *sameAs* assertion between *i* and *j*), instead of considering 5 TP (the assertions of *i*), 5 FP (the assertions of *j*) and 5 FN (the assertions of *j* that are missing for *i*).

Finally, although the corpus corresponds to house sales annotated as being in Caen, these real estate properties are often in cities outside of Caen. Sometimes, the city is not explicitly mentioned (for example, “in a searched municipality”). In such a case, KOnPoTe creates an individual $\mathit{advertisementX\_municipality}1$ and locates the property in it. This does not seem wrong to us. On the opposite, such an assertion seems more interesting than an absence of assertion, in particular in the context of the DECA project, mentioned in Section 1, where wrong annotations need to be corrected. This kind of assertions are therefore also in the GS.

4.2. Tools used

We tested the approach using Java. We use the following dependencies:

– OWL API [33] to handle the KG;
– Stanford NLP [34] to split the texts into sentences;
– two French lemmatizers : the one from Ahmet Aker[6] (denoted as $Aker$) and TreeTagger[7] [35] (denoted as TT);
– Openllet reasoner[8] (Pellet).

Note also that some small adjustments were made on the results of the tokens from the lemmatizers when we try to take the previous or following words within the text. For example, *10 000* becomes *10000*, *3 . 4* becomes *3.4*, etc.

All experimental files (input, outputs for all tested approaches, and gold standard) are available[9].

4.3. Results and discussion

As mentioned in Section 2, our problematic differs from related work, so there are no existing baselines that can be considered as a fair comparison. We chose to compare KOnPote with three baselines, all with the two lemmatizers. They use the same first steps as KOnPoTe: the terminology matcher, the initialisation of the matchings treatments and the instantiation of the main class. The most basic baseline that we use (*Baseline*) consists in treating only the class, individual and property matchings. In other words, it performs the first three steps of the text-based analysis. Another one (*Baseline+next*) consists in adding the treatment of the next matching (and obviously the initialisation of the linkabilities). Finally, the last one (*Text-based analysis*) consists in doing the complete text-based analysis (but not the knowledge-based analysis).

Table 4 shows the results. Precision, recall and F-measure are computed both macroscopically and microscopically. The macro-computation is the average of the metrics of each add, whereas the micro-computation considers the sum of all VP, FP, FN of each add. Basically, in the macro-average, each add has the same weight; and in the micro-average, each assertion has the same weight.

Table 4
Results on KOnPoTe and three baselines on macro-average (*macro*) and micro-average (*micro*) using lemmatizers *Aker* and *TT*

Approach	Precision_{macro}	Recall_{macro}	F-measure_{macro}	Precision_{micro}	Recall_{micro}	F-measure_{micro}
KOnPoTe_{Aker}	**0.9516**	**0.8740**	**0.9079**	**0.9465**	**0.8606**	**0.9015**
KOnPoTe_{TT}	0.9496	0.8681	0.9039	0.9446	0.8545	0.8973
$\text{Text-based analysis}_{Aker}$	0.8989	0.4648	0.5994	0.8956	0.4726	0.6188
$\text{Text-based analysis}_{TT}$	0.8964	0.4579	0.5929	0.8937	0.4662	0.6127
$\text{Baseline+next}_{Aker}$	0.8911	0.3138	0.4440	0.8741	0.3085	0.4561
$\text{Baseline+next}_{TT}$	0.8879	0.3081	0.4377	0.8732	0.3036	0.4505
Baseline_{Aker}	0.9234	0.1922	0.3099	0.9135	0.1926	0.3182
Baseline_{TT}	0.9230	0.1888	0.3054	0.9138	0.1892	0.3135

We can see that the difference of performance between the two lemmatizers is relatively small. Nevertheless, *Aker* outperforms *TreeTagger*. We also notice that the added modules have a good contribution on the results, since the addition of each module (treatment of the next matching, treatment of the individuals without predecessors, knowledge-based analysis) generates a relatively high gain of F-measure. First, we can see that the baseline, composed of the basic treatments for classes, individuals and properties, gives a relatively high score in precision (> 0.9)

[6]http://staffwww.dcs.shef.ac.uk/people/A.Aker/activityNLPProjects.html
[7]https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/
[8]https://github.com/Galigator/openllet
[9]Experimental files are available at https://doi.org/10.5281/zenodo.5776752. A zip file with a runnable jar for KOnPote with Aker's lemmatizer is available at https://alec.users.greyc.fr/research/konpote/.

but a low score in recall (< 0.2). This means that most of the assertions that are made are correct, but a lot of assertions are missing. The addition of modules allows us to add assertions, which are mostly correct. Indeed, as modules are added, we are able to increase the recall without losing to much precision. More precisely, the treatment of the next matching adds some noise (small loss of precision) but increases the recall by half (from ~ 0.2 to ~ 0.3). The treatment of the individuals without predecessors increases the recall by half too (from ~ 0.3 to ~ 0.45), without decreasing the precision (by increasing it slightly). Finally, the knowledge-based analysis adds a considerable contribution. Both precision and recall increase, leading to an increase of F-measure by half (from ~ 0.6 to ~ 0.9). These three modules are therefore essential: they insert a lot of missing assertions without adding too many wrong assertions.

5. Conclusion and future work

This paper presents KOnPoTe, an approach to populate a domain ontology, or a knowledge graph, from textual descriptions of objects from this domain. The problem studied differs from the state of the art: named entities are not the main features, verbs are not very meaningful, properties are not necessarily only object properties, and can be non-binary. In this context, KOnPoTe presents a processing chain, whose results are promising on a first experimentation. The intuition behind the creation of KOnPoTe is to propose a population algorithm that is not based on a particular domain. The ideas of the algorithms of each module are only linked to the context of the problem, and not to a specific domain.

Many perspectives are open to us. We would like to experiment the approach on other domains. Of course, such experiments are relatively costly: it requires textual descriptions of each object and a knowledge graph describing the domain. To be able to correctly assess the results, the most laborious part is the manual work: creation of a gold standard, as well as potential equivalences between this gold standard and the output knowledge graph, and potential missing sameAs links in the output. We have several domains in mind, first of all still with classified ads: sales (or rentals) of apartments, or of certain types of objects (cars, clothes, etc.); but also in other domains, in particular advertising descriptions: vacation spots, hotels, restaurants, etc. Another idea is to test KOnPoTe on the same domain and corpus but with different domain ontologies (different choices of representation). A deep analysis of such an experiment could lead to a set of constraints or advice to follow in the knowledge representation of the input ontology.

Another future work would be to adapt and test the approach on other languages. Indeed, this approach should be easily adaptable, the only elements specific to French are the list of words evoking negation, and a system translating a possible number described in letters into its correspondence in numbers. However, some parts of the algorithm may have to be adapted with respect to the new language: for instance, in French, an adjective is usually after the noun to which it refers, whereas it is not the case in English (e.g., “kitchen equipped” vs “equipped kitchen”). When this adjective designates a property, the treatment of the property matching will check possible subjects for this property in the matchings before this adjective. In English, this may not be a good solution. For that problem, a use of the Part-Of-Speech tags, instead of our current solution, might be helpful.

Another perspective is to improve the algorithm when a matching concerns several entities of the knowledge graph. As of now, the algorithm considers all entities that are matched. In the current experiment, there is only one case where this happens: the energy consumption. Indeed, the energy consumption can be expressed either via a numerical value, or via a letter. In our ontology, this is represented via two properties, both using the same terminology. This means that a matching on a keyword from this terminology considers both of these properties, so the algorithm tries to instantiate both. Since ranges are different, the instantiation fails for at least one of them, solving the problem. However, we can easily imagine some use cases where this is a real issue. For example, a unit (or a unit expression) is currently associated with only one property (e.g., “m^2” for area, “Fees: xxx%” for fee percentage, etc). In particular, we can mention the fact that “€” is too much general and should correspond to several properties (price with fees, price without fees, property tax). As of now, we use different unit or unit expressions to distinguish these price properties (like “€ without fees”, “property tax is xxx €”). Otherwise, the algorithm should be improved to be able to make a choice between all the entity candidates.

Finally, our main final goal is to use KOnPoTe as a first step to deal with the problem of erroneous annotations of classified advertisements, as mentioned in Section 1.

Acknowledgements

We would like to acknowledge Quentin Leroy and Jean-Philippe Kotowicz for their participation in the elaboration of a first version of the initial knowledge graph, as well as Enor-Anaïs Carré and Morgan Gueret for the constitution of the corpus.

References

[1] S. Staab and R. Studer, *Handbook on ontologies*, Springer, 2009. ISBN 9783540926733 3540926739.

[2] C. Bizer, T. Heath and T. Berners-Lee, Linked Data - The Story So Far, *International Journal on Semantic Web and Information Systems* **5**(3) (2009), 1–22.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak and Z. Ives, DBpedia: A Nucleus for a Web of Open Data, in: *Proceedings of the 6th International Semantic Web Conference (ISWC)*, Lecture Notes in Computer Science, Vol. 4825, Springer, 2008, pp. 722–735.

[4] D. Vrandecic and M. Krötzsch, Wikidata: a free collaborative knowledgebase, *Commun. ACM* **57** (2014), 78–85.

[5] F.M. Suchanek, G. Kasneci and G. Weikum, YAGO: A Large Ontology from Wikipedia and WordNet, *J. Web Semant.* **6** (2008), 203–217.

[6] P. Buche, B. Cuq, J. Fortin and C. Sipieter, Expertise-based decision support for managing food quality in agri-food companies, *Comput. Electron. Agric.* **163** (2019).

[7] C. Alec, C. Reynaud-Delaître and B. Safar, An Ontology-driven Approach for Semantic Annotation of Documents with Specific Concepts, in: *Extended Semantic Web Conference, ESWC*, Lecture Notes in Computer Science, Springer, 2016, pp. 609–624.

[8] C. Alec, C. Reynaud-Delaître, B. Safar, Z. Sellami and U. Berdugo, Automatic Ontology Population from Product Catalogs., in: *International Conference on Knowledge Engineering and Knowledge Management, EKAW*, Vol. 8876, Springer, 2014, pp. 1–12. ISBN 978-3-319-13703-2.

[9] F. Amardeilh, P. Laublet and J.-L. Minel, Document annotation and ontology population from linguistic extractions, in: *K-CAP '05: Proceedings of the 3rd international conference on Knowledge capture*, ACM, New York, NY, USA, 2005, pp. 161–168. ISBN 1-59593-163-5.

[10] T.R. Gruber, A Translation Approach to Portable Ontology Specifications, *Knowl. Acquis.* **5**(2) (1993), 199–220. doi:10.1006/knac.1993.1008.

[11] G. Petasis, V. Karkaletsis, G. Paliouras, A. Krithara and E. Zavitsanos, Ontology Population and Enrichment: State of the Art, in: *Knowledge-Driven Multimedia Information Extraction and Ontology Evolution*, 2011, pp. 134–166.

[12] P. Cimiano, *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387306323.

[13] M. Lubani, S.A.M. Noah and R. Mahmud, Ontology population: Approaches and design aspects, *Journal of Information Science* **45** (2019), 502–515.

[14] D. Maynard, Y. Li and W. Peters, NLP Techniques for Term Extraction and Ontology Population, in: *Ontology Learning and Population*, P. Buitelaar and P. Cimiano, eds, Frontiers in Artificial Intelligence and Applications, Vol. 167, IOS Press, 2008, pp. 107–127. ISBN 978-1-58603-818-2.

[15] H. Tanev and B. Magnini, Weakly Supervised Approaches for Ontology Population, in: *Ontology Learning and Population*, P. Buitelaar and P. Cimiano, eds, Frontiers in Artificial Intelligence and Applications, Vol. 167, IOS Press, 2008, pp. 129–143. ISBN 978-1-58603-818-2.

[16] H.-G. Yoon, Y.-J. Han, S.-B. Park and S.-Y. Park, Ontology Population from Unstructured and Semi-structured Texts, in: *International Conference on Advanced Language Processing and Web Information Technology*, 2007, pp. 135–139. ISBN 978-0-7695-2930-1.

[17] H. Alani, S. Kim, D.E. Millard, M.J. Weal, P.H. Lewis and N.R. Shadbolt, Automatic Ontology-based Knowledge Extraction and Tailored Biography Generation from the Web, *IEEE Intell Syst* (2003), 14–21.

[18] P. Buitelaar, P. Cimiano, S. Racioppa and M. Siegel, Ontology-based Information Extraction with SOBA, in: *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, European Language Resources Association (ELRA), Genoa, Italy, 2006.

[19] C. Brewster, F. Ciravegna and Y. Wilks, User-Centred Ontology Learning for Knowledge Management, in: *NLDB '02: Proceedings of the 6th International Conference on Applications of Natural Language to Information Systems-Revised Papers*, Springer-Verlag, 2002, pp. 203–207. ISBN 3-540-00307-X. http://portal.acm.org/citation.cfm?id=666125.

[20] F.M. Suchanek, G. Ifrim and G. Weikum, LEILA: Learning to Extract Information by Linguistic Analysis, in: *Proceedings of the 2nd Workshop on Ontology Learning and Population: Bridging the Gap between Text and Knowledge*, Association for Computational Linguistics, Sydney, Australia, 2006, pp. 18–25. https://aclanthology.org/W06-0503.

[21] D. Celjuska and D.M. Vargas-vera, Ontosophie: A Semi-Automatic System for Ontology Population from Text, in: *International Conference on Natural Language Processing (ICON)*, 2004.

[22] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam and S. Slattery, Learning to construct knowledge bases from the World Wide Web, *Artificial Intelligence* **118**(1) (2000), 69–113. doi:https://doi.org/10.1016/S0004-3702(00)00004-7. https://www.sciencedirect.com/science/article/pii/S0004370200000047.

[23] S. Castano, I.S.E. Peraldi, A. Ferrara, V. Karkaletsis, A. Kaya, R. Möller, S. Montanelli, G. Petasis and M. Wessel, Multimedia Interpretation for Dynamic Ontology Evolution, *Journal of Logic and Computation* **19**(5) (2008), 859–897. doi:10.1093/logcom/exn049.

[24] M.A. Hearst, Automatic Acquisition of Hyponyms from Large Text Corpora, in: *COLING 1992 Volume 2: The 15th International Conference on Computational Linguistics*, 1992. https://www.aclweb.org/anthology/C92-2082.

[25] M. Finkelstein-landau and E. Morin, Extracting Semantic Relationships between Terms: Supervised vs. Unsupervised Methods (1999).

[26] R. Yangarber and R. Grishman, NYU: Description of the Proteus/PET System as Used for MUC-7 ST, in: *Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29 - May 1, 1998*, 1998. https://aclanthology.org/M98-1011.

[27] Z. Ibrahim, S.A. Noah and M.M. Noor, Rules for Ontology Population from Text of Malaysia Medicinal Herbs Domain, in: *Rough Set and Knowledge Technology*, J. Yu, S. Greco, P. Lingras, G. Wang and A. Skowron, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 386–394. ISBN 978-3-642-16248-0.

[28] J. Makki, A.-M. Alquier and V. Prince, Ontology Population via NLP Techniques in Risk Management, *Int J Hum Soc Sci* (2009), 212–217.

[29] C. Faria, I. Serra and R. Girardi, A domain-independent process for automatic ontology population from text, *Science of Computer Programming* **95** (2014), 26–43, Special Issue on Systems Development by Means of Semantic Technologies. doi:https://doi.org/10.1016/j.scico.2013.12.005. http://www.sciencedirect.com/science/article/pii/S0167642313003419.

[30] B. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider and U. Sattler, OWL 2: The next step for OWL, *Web Semantics: Science, Services and Agents on the World Wide Web* (2008).

[31] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneijder and L.A. Stein, OWL Web Ontology Language Reference, Recommendation, World Wide Web Consortium (W3C), 2004, See http://www.w3.org/TR/owl-ref/.

[32] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof and M. Dean, SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission, World Wide Web Consortium, 2004. http://www.w3.org/Submission/SWRL.

[33] M. Horridge and S. Bechhofer, The OWL API: A Java API for Working with OWL 2 Ontologies, in: *Proceedings of the 6th International Conference on OWL: Experiences and Directions - Volume 529*, OWLED'09, CEUR-WS.org, Aachen, DEU, 2009, pp. 49–58–.

[34] C.D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S.J. Bethard and D. McClosky, The Stanford CoreNLP Natural Language Processing Toolkit, in: *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, pp. 55–60. http://www.aclweb.org/anthology/P/P14/P14-5010.

[35] H. Schmid, Probabilistic Part-of-Speech Tagging Using Decision Trees, 1994.