GimmeHop: A recommender system for mobile devices using ontology reasoners and fuzzy logic

Ignacio Huitzil[a], Fernando Bobillo[a,b], Fernando Alegre[c]

[a] *University of Zaragoza, Zaragoza, Spain*
[b] *Aragon Institute of Engineering Research (I3A), Zaragoza, Spain*
[c] *Entertainment Solutions, Zaragoza, Spain*

Abstract

This paper describes GimmeHop, a beer recommender system for Android mobile devices using fuzzy ontologies to represent the relevant knowledge and semantic reasoners to infer implicit knowledge. GimmeHop use fuzzy quantifiers to deal with incomplete data, fuzzy hedges to deal with the user context, and aggregation operators to manage user preferences. The results of our evaluation measure empirically the data traffic and the running time in the case of remote reasoning, the size of the ontologies that can be locally dealt with in a mobile device in the case of local reasoning, and the quality of the automatically computed linguistic values supported in the user queries.

Keywords: Fuzzy ontologies, Aggregation, Fuzzy quantifiers, Recommender systems

1. Introduction

In our daily lives, we typically use a significant number of mobile applications (*apps*), for example to receive updated information about the weather or the traffic. This is possible because of the ever increasing computing capabilities of mobile devices, and the almost pervasive connectivity that the current wireless networks provide us with.

Because semantic technologies have proved to be very useful in many applications [4], enhancing such applications to enjoy the advantages of semantic technologies has been suggested. In particular, *ontologies* have become a de-facto standard for knowledge representation. Using ontologies for the knowledge representation of smart apps will have several

Email addresses: ihuitzil@unizar.es (Ignacio Huitzil), fbobillo@unizar.es (Fernando Bobillo), fernandoalegrem@gmail.com (Fernando Alegre)

benefits, such as improving knowledge sharing, reusing and maintenance, decoupling of the knowledge from the application, or discovering implicit knowledge.

Semantic reasoners are software implementations providing an automatic discovery of implicit knowledge that can be logically inferred from ontology axioms. This is possible because ontology languages, such as the the standard language for ontology representation OWL 2 (Web Ontology Language) [11], have logical foundations. To do semantic reasoning on mobile devices, it is possible to use local, remote, or hybrid approaches [3]. Unfortunately, the use of semantic reasoners in apps is still rather challenging [5, 58].

While ontologies and semantic web technologies have proved to be very useful in many applications, there are many real world domains with imprecise and vague knowledge. In such scenarios, *fuzzy extensions* of the ontologies with elements of fuzzy logic and fuzzy set theory have been explored [60]. This way, for example, it would be possible to consider places which are *close* to the current user location, obtained using the sensors of the mobile device.

However, all the previous techniques have been explored in an isolated way. Unfortunately, the integrated management of ontologies, fuzzy logic, and semantic reasoners on mobile devices has not received enough attention. In particular, semantic reasoning is usually not performed on the local device but on an external server, requiring good connectivity and leading to some privacy risks. In this paper, we describe an application for mobile devices combining fuzzy logic and semantic reasoners. In particular, we present a semantic beer recommender system called GimmeHop, aiming at providing users with good recommendations about beers. We think that this kind of applications is particularly interesting these days: more foreign beers are imported by many stores and bars, the number of artisan beers is growing significantly, and users are willing to try new beer styles, thanks in large part to to the phenomenon of home-brewing.

An important issue when managing big amounts of real data is that quite often there is missing information, i.e., entities for which the values of some attributes are unknown. In this paper, we use *quantifier-guided aggregation* [56] to provide recommendations even in cases of incomplete information.

The remaining of this paper is organized as follows. Section 2 provides some background on fuzzy logic and fuzzy ontologies. Then, Section 3 describes our recommender system, paying special attention to the use of fuzzy ontologies and the management of data on mobile

devices. Next, Section 4 reports an evaluation of the quality of the linguistic labels and the recommendations, but also of the performance of the system. A detailed comparison our contribution with other previous work can be found in Section 5. Finally, Section 6 sets out some conclusions and ideas for future work.

2. Background

This section overviews some basics notions on fuzzy logic (Section 2.1) and fuzzy ontologies (Section 2.2) that will be used in the rest of this paper. Readers who are familiar with these subjects might jump ahead directly to the next section.

2.1. Fuzzy Sets and Fuzzy Logic

Fuzzy set theory and fuzzy logic were proposed by L. A. Zadeh [59] to manage imprecise and vague knowledge. While in classical set theory elements either belong to a set or not, in fuzzy set theory elements can belong to some degree. More formally, let X be a set of elements called the reference set. A *fuzzy subset* A of X is characterized by a membership function $\mu_A(x)$, or simply $A(x)$, which assigns to every $x \in X$ a degree of truth, measured as a value in a truth space, usually [0,1].

As in the classical case, 0 means no-membership and 1 full membership, but now a value between 0 and 1 represents the extent to which x can be considered as an element of the fuzzy set A. To distinguish between fuzzy sets and classical (non-fuzzy) sets, we refer to the latter as *crisp sets*.

Fuzzy logics provide compositional calculi of degrees of truth. The conjunction, disjunction, complement and implication operations are performed in the fuzzy case by a *t-norm* function $\otimes$, a *t-conorm* function $\oplus$, a *negation* function $\ominus$ and an *implication* function $\Rightarrow$, respectively. For a formal definition of these functions we refer the reader to [24]. Examples of t-norm and t-conorm are the minimum and the maximum, respectively.

Fuzzy modifiers (or fuzzy hedges) apply to fuzzy sets to change their membership function. Popular examples include very or moreOrLess. Formally, a *modifier* is a function $f_m : [0,1] \rightarrow [0,1]$. For example, For instance, we can use the modifier $\mathsf{very}(x) = x^2$ and apply it to the fuzzy set *Young*, so that for each individual x we can be compute the degree of being

very young as $\mu_{VeryYoung}(x) = (\mu_{Young}(x))^2$. Two notable families or fuzzy modifiers are increasing, if $f_m(x) \geq x$, and weakening, if $f_m(x) \leq x$. very is an example of weakening modifier.

Other popular fuzzy operators are *Aggregation Operators* (AOs), mathematical functions that are used to combine different pieces of information (typically, membership degrees to fuzzy sets) [50]. Given a domain $\mathbb{D}$ (such as the reals), an AO of dimension n is a mapping $@ : \mathbb{D}^n \to \mathbb{D}$. For us, $\mathbb{D} = [0,1]$. Thus, an AO aggregates n values $x_1, x_2, ..., x_n$ of n different criteria. Often, an AO $@$ is parameterized with a vector of n weights $W = [w_1, ..., w_n]$ such that $w_i \in [0,1]$ and $\sum_{i=1}^{n} w_i = 1$. In that case the AO is denoted as $@_W$.

A classical example of AO is the *weighted mean* (WMEAN) or weighted sum, defined as

$$@_W^{\text{WS}}(x_1, ..., x_n) = \sum_{i=1}^{n} w_i x_i \ . \tag{1}$$

A very important family of AOs are the *Ordered Weighted Averaging* (OWA) operators [54]. OWA operators provide a parameterized class of mean type AOs. Formally, given a weighting vector W, an OWA operator of dimension n is an AO such that:

$$@_W^{\text{owa}}(x_1, ..., x_n) = \sum_{i=1}^{n} w_i x_{\sigma(i)} \tag{2}$$

where σ is a permutation such that $x_{\sigma(1)} \geq x_{\sigma(2)} \geq \cdots \geq x_{\sigma(n)}$, i.e., $x_{\sigma(i)}$ is the i-th largest of the values $x_1, \ldots, x_n$ to be aggregated. A fundamental aspect of these operators is the reordering step. In particular, a weight w_i is not associated with a specific argument but with an ordered position of the aggregate. As a result, the OWA operator is non-linear. By choosing different weights, OWA operators can implement different AOs, such as arithmetic mean, k-th maximum, k-th minimum, median or order statistic, among others. However, the weighted sum cannot be represented as an OWA operator.

OWA operators verify internality, i.e., $\min(x_1, ..., x_n) \leq @(x_1, ..., x_n) \leq \max(x_1, ..., x_n)$. Since the two extreme cases of OWA operators correspond to the largest t-norm (the minimum) and the smallest t-conorm (maximum), the class of OWA operators nicely "fills the gap" between the intersection and the union [41].

A common problem when working with OWA operators is the definition of the weights. A very popular solution is to rely on the so-called *quantifier-guided aggregation* [56].

Before addressing the problem of computing the weights, let us briefly recall the notion of quantifiers. Classical logic has two quantifiers, the universal $\forall$ and the existential $\exists$ quantifier.

These are extremal ones among several other linguistic quantifiers such as *most, few, about half, some, many*, etc. Quantifiers can be seen as absolute or proportional [24]; we will only consider the proportional ones. A proportional fuzzy quantifier $Q:[0,1]\rightarrow[0,1]$ is a fuzzy subset such that for each $r\in[0,1]$, the membership grade $Q(r)$ indicates the degree to which the proportion r satisfies the linguistic quantifier that Q represents. We will consider the *Regular Increasing Monotone* (RIM) quantifiers [55], that satisfy the boundary conditions $Q(0) = 0$ and $Q(1) = 1$, and are monotone increasing, i.e., $Q(x_1) \leq Q(x_2)$ when $x_1 \leq x_2$. Essentially, these quantifiers are characterized by the idea that as the proportion increases, the degree of satisfaction does not decrease.

A RIM Q can be used to define an OWA weighting vector W_Q of dimension k, where each weight is computed as follows:

$$w_i = Q(\frac{i}{n}) - Q(\frac{i-1}{n}) \tag{3}$$

Note that indeed $w_i\in[0,1]$ and $\sum_i w_i = 1$. For example, in Figure 1, *most* and *almost all* are RIM quantifiers, but *about half* is not.

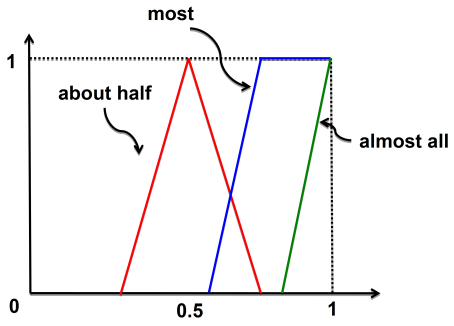


Figure 1: Some examples of fuzzy quantifiers

We propose to use right-shoulder (Figure 2 (d)), linear (Figure 2 (e)), and power functions (Figure 2 (f)) to build RIMs.

2.2. Fuzzy Ontologies

An ontology is an explicit and formal specification of the concepts, individuals and relationships that exist in some area of interest, created by defining axioms that describe the

properties of these entities [43]. Ontologies can provide semantics to data, making knowledge maintenance, information integration, and reuse of components easier.

In several real world domains, knowledge is imprecise and vague, and classical ontologies are not enough. As a solution, fuzzy ontologies extend classical (crisp) ontologies by considering several notions of fuzzy set theory and fuzzy logic [45, 29].

The elements of a fuzzy ontology are the following ones:

- *Fuzzy concepts* (or classes) denote unary predicates and are interpreted as fuzzy sets of individuals, such as YoungHuman. Concepts can be simple (atomic) or complex, built up using different types of concept constructors depending on the expressivity of the ontology language.

- *Individuals* denote domain elements or objects. For example, luke and darthVader.

- *Fuzzy properties* (or roles) denote binary predicates relating a pair of elements and are interpreted as fuzzy binary relations. There are two types of properties: *object properties* link a pair of individuals, whereas *data properties* relate an individual with a data value. For instance, hasFriend relates two human individuals and is an object property, while hasAge links an individual with an integer number and is a data property.

- *Fuzzy datatypes* denote elements that do not belong to the represented domain, but rather to a different domain that is already structured and whose structure is already known to the machine. Crisp data values can be, for example, numerical values, textual values, or dates. Fuzzy datatypes generalize crisp values by using a fuzzy membership function. For example, instead of considering the crisp value 18, now it is possible to consider about18, defined using a triangular function. In this paper, we will restrict to some fuzzy datatypes defined over a subinterval of the rational numbers. In particular, we will consider trapezoidal (Figure 2 (a)), triangular (Figure 2 (b)), left-shoulder (Figure 2 (c)), and right-shoulder (Figure 2 (d)) membership functions.

- *Fuzzy axioms* are formal statements involving these ontology elements, like a recipe that defines how to combine the previous ingredients to represent the knowledge of some particular domain. A novelty of fuzzy ontologies is that fuzzy axioms express statements that are not either true or false but hold to some degree. The available types

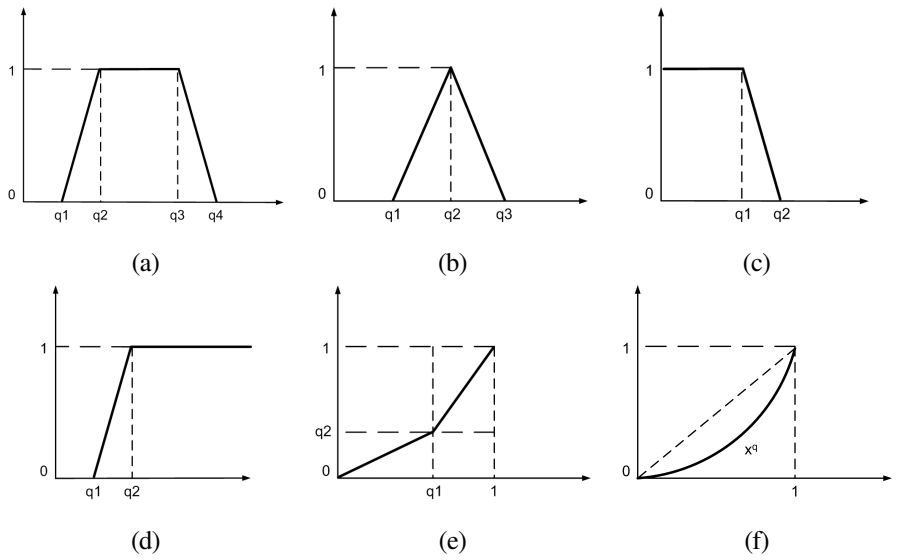


Figure 2: (a) Trapezoidal function; (b) Triangular function; (c) Left-shoulder function; (d) Right shoulder function; (e) Linear function; (f) Power function

of axioms depend on the expressivity of the ontology language, but some typical types are:

- *Concept assertions* state the membership of an individual to a class. For example, we can state that luke belongs to the concept of YoungHuman with at least degree 0.95, meaning that he is rather young.
- *Object property assertions* describe the relation between two individuals, e.g., one can state that luke and darthVader are related via hasFather.
- *Data property assertions* describe the relation between an individual and a data value. For example, it is possible to express that Luke's age is 19 by relating luke and the number 19 via hasAge.
- *Subclass* axioms, stating that a concept is more specific (a subclass) of another one. For example, Woman is more specific than Human.

The interested reader can find a complete list of the crisp OWL 2 concepts, properties, and axioms in [11]. Although there is not a standard fuzzy ontology language, *Fuzzy OWL 2* [6]

is a popular choice. The language extends OWL 2 ontologies [11] with OWL 2 annotations encoding fuzzy information using an XML-like syntax. The key idea of this representation is to start with an OWL 2 ontology created as usual, with a classical ontology editor. Then, it is possible to annotate the elements to represent the features of the fuzzy ontology that OWL 2 cannot directly encode. In particular, it is possible to annotate fuzzy axioms by adding a degree of truth, to represent fuzzy datatypes, and to define specific elements of fuzzy ontologies (such as fuzzy modifiers or aggregated concepts). This language is supported by some fuzzy ontology reasoners such as fuzzyDL [7].

3. GimmeHop System

This section describes our beer recommender system. Section 3.1 starts by describing the fuzzy ontology. Then, Section 3.2 describes the architecture, used technologies, offered services, algorithms, user interface and some relevant implementation details.

3.1. A Fuzzy Beer Ontology

Our fuzzy ontology is able to represent both types and concrete examples of beers. The ontology is distributed in two files, with a general schema including definition of the types of beers and the properties, and another file that populates the schema ontology with individuals (beers) and their attribute values. While the schema can be reused, the license of the beer data does not currently make it possible to distribute the instances (a third-party beer company owns the data). On the contrary, we envision that each bar or beer store interested on the app can populate the schema ontology only with the relevant beers that are available in their particular case.

Our ontology is encoded in OWL 2 EL [51], one of the tractable profiles of OWL 2, and was developed using the Protégé editor [34]. The ontology schema has 411 axioms (215 logical axioms), 121 classes, 5 object properties, 14 data properties, 10 fuzzy datatypes, and 22 country individuals. There are two additional files populating the schema with individuals (we will give more details later on). The ontology architecture is illustrated in Figure 3.

Classes. Figure 4 (a) shows the main classes of the ontology. The most important ones form a hierarchy of beer types, grouped in 5 main families: Ale, Lager, Sour, Wheat, and

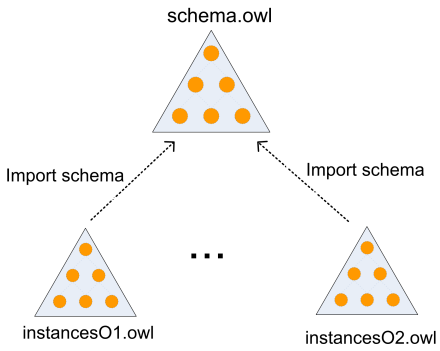


Figure 3: Ontology schema and two instance files

Specialty. These classes are abstract and cannot have instances, but have subclasses (that can be abstract or not). Specialty includes beers with unusual grains, fruits, spices, herbs, etc. The depth of the subontology with the beer types is 5.

There are also some classes representing a brewery (Brewery), a location (Country), a currency (Currency) to specify the beer price, a won award (Award), and a hierarchy of ingredients (Ingredient).

Our ontology imports *DBpedia* [27] and uses its list of countries (e.g., dbpedia:Belgium)[1] and its list of currencies (e.g., dbpedia:Euro). Ingredients of the ontology reuse the categorization in an existing ontology about beers (described in Section 5).

Properties. The main properties of the ontology are depicted in Figure 4 (b). Our 5 object properties link instances of Beer or Brewery with instances of Country (locatedAt), and instances of Beer with instances of Award (wonAward), Brewery (brewedBy), Currency (usesCurrency), and Ingredient (hasIngredient). Note that we do not represent their inverse properties, as they are not allowed in OWL 2 EL [51].

Data properties make it possible to represent different beer attributes:

- ABV (Alcohol By Volume) and IBU (International Bitterness Units) denote alcohol and bitterness degrees, respectively, measured in [0,100] and [0,1000], respectively.

[1] Note that some of them do not belong to the class dbpedia:Country, such as dbpedia:Scotland.

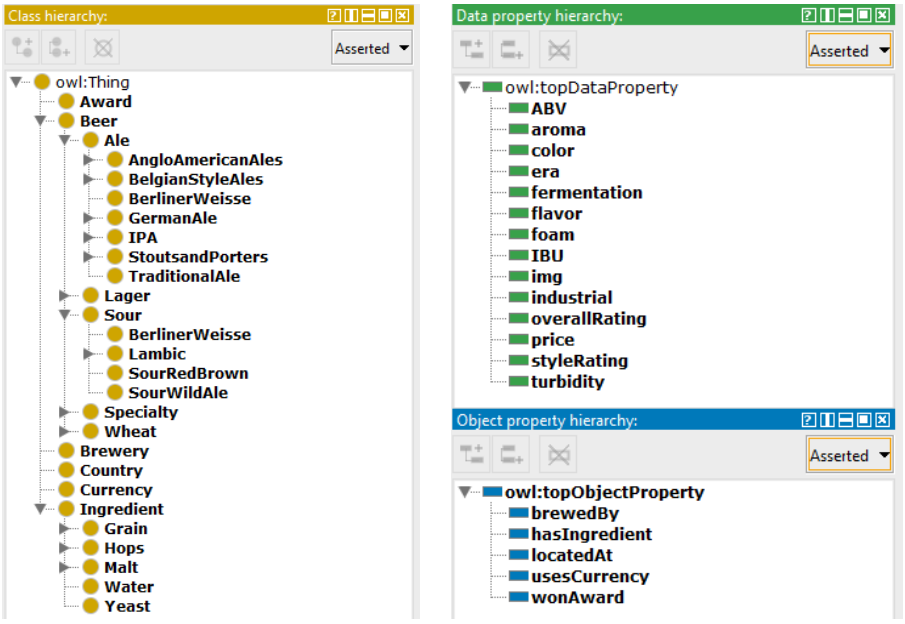


Figure 4: Ontology representation, a) classes and b) data and object properties.

- color is a numerical value representing the color in Standard Reference Method (SRM) units, measured in [0,40].
- turbidity is a numerical value representing how hazy a beer is in European Brewery Convention (EBC) units, measured approximately in [0,200].
- aroma, flavor, and foam represent some information using string values. These properties are not functional so that several values can be attached to a single beer. For example, we can have a data property assertion stating that it smells like bananas and another one stating that it states like clove.
- fermentation has the following possible values: top, bottom, any, wild, and aged.
- era indicates if the beer was brewed in a modern style, in a traditional style, or if belongs to a historical period and is no longer available.
- price is a numerical property representing the cost of the beer (recall that the object property currency indicates the semantics of the number).

- overallRating and styleRating are numerical values in [0,100] representing the percentile of the rating, compared to all the beers or to the beers of the same style, respectively, given by the community of users.
- industrial is a Boolean property (true indicates a industrial beer, false an artisan one).
- img is the path of an image file with a picture of the beer that could be displayed in the GUI.

In the future, the ontology could be extended with a hierarchy of classes representing aromas, flavors, and foam types; and replacing the data properties with object properties. However, because our aim is to manage fuzzy datatypes and there is no easy way to represent those attributes using fuzzy membership functions defined over a a numerical scale, we have left it as future work.

Fuzzy datatypes. The ontology stores precise values using data property assertions (for instance, that the alcohol degree of Guinness Draught is 4.2), but also includes 10 fuzzy datatypes, 5 associated to the alcohol and 5 associated to the bitterness. There are more data properties to which one could also associate fuzzy datatypes, such as price, color, or turbidity. However, as we will discuss later, our beer data did not include that information and hence they were not considered in the current version. Furthermore, we chose not to define fuzzy datatypes for the ratings because in a recommender system the user is interested in items with the best possible rating, as long as they satisfy his/her requirements.

To compute the linguistic values of the data properties of an ontology, we used Datil [22]. [2] Datil is a software implementation of an algorithm to learn fuzzy datatypes for fuzzy ontologies based on the values of the data properties. A clustering algorithm provides a set of centroids, which are then used as the parameters to build fuzzy membership functions partitioning the domain of a values of the data property. Datil implements several unsupervised clustering algorithms such as *k-means* [28], *fuzzy c-means* [2], and *mean shift* [9], and supports different input formats, including OWL 2 ontologies. Examples of fuzzy membership functions built after the centroids (denoted by broken lines) are shown in Figure 9.

[2] http://webdiis.unizar.es/~ihvdis/Datil

After our first experiments, we noticed that the results were counter-intuitive because of the existence of beers with very high alcohol values. Figure 5 shows the number of beers for each alcohol degree (with 1 digit precision). We can see that there are several beers with more than 20 degrees; one of them (Schorschbräu Schorschbock) with 57.7°. A consequence of having such values is that the centroids used to build the fuzzy membership functions are much higher than the expected values for a human expert. For this reason, we extended Datil with a new feature: it is now possible to specify a minimal and a maximal threshold (denoted Θ_1 and Θ_2, respectively), so that lower and greater values, respectively, are ignored for the clustering algorithm. In Section 4.1 we will report some experiments to select the thresholds Θ_1 and Θ_2, and the best clustering algorithm.

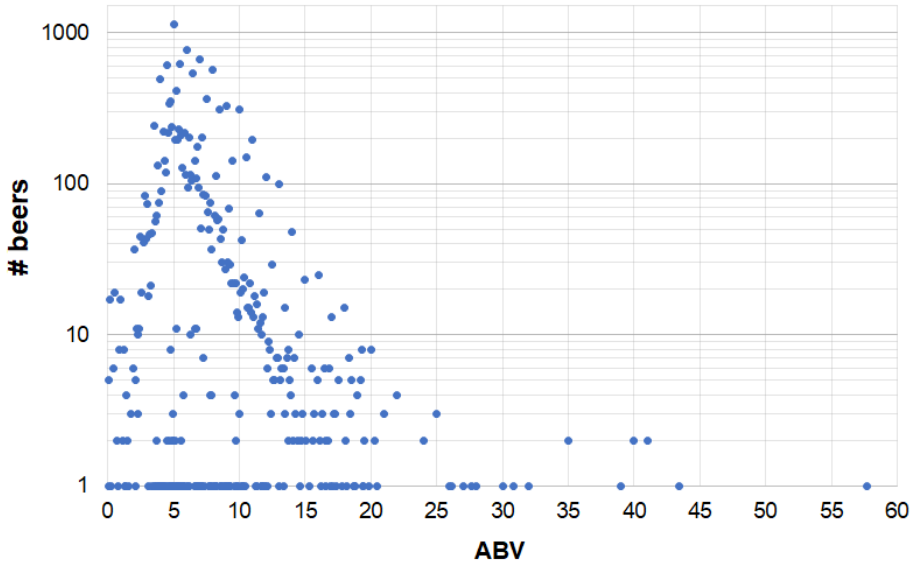


Figure 5: ABV of beers

Class axioms. There are also some axioms imposing some restrictions on the classes. Firstly, there are disjointness axioms stating that two classes cannot share any instance. For example, Ale and Lager are disjoint. Note however that Ale, Sour, and Wheat, three of the five families of beer types, are not disjoint; indeed, BerlinerWeisse is a subclass of those three classes.

We can also find some necessary conditions of the beer types. For example, Lager

restricts the value of the property fermentation to be low. Another typical restriction involves the colour, for instance, Schwarzbier restricts the value of the property colour to be in [17,30] (SRM).

There are also a few General Concept Inclusion (GCI) axioms that make it possible to infer that if a beer is brewed by a brewery located in a country (e.g., Belgium), that country should also be associated to the beer, e.g., (brewedBy `some` (locatedAt `value` dbpedia:Belgium)) `SubClassOf` (locatedAt `value` dbpedia:Belgium).[3]

Individuals. A first file (denoted $O1$) populates the ontology with 15317 beer individuals and 4510 brewery individuals. In general, for each beer we know its beer type (membership to a class), its brewery and country (via 2 object property assertions), and the values of the data properties ABV, IBU, img, overallRating, and styleRating (via 5 data property assertions). Figure 6 illustrates a sample individual.

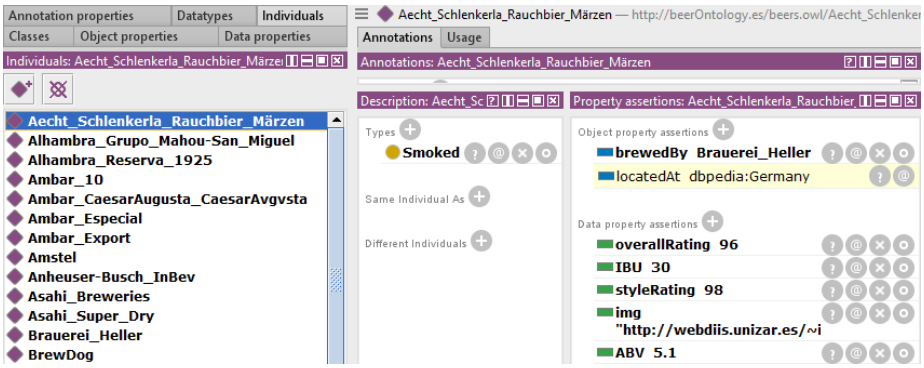


Figure 6: Example of a beer individual

Table 1 shows some statistics about the individuals in ontology $O1$. In particular, it includes the number and the percentage of beers for which information about some features (alcohol, bitterness, country, and style rating) is available. While the alcohol degree is always known, this is not the case for other attributes (indeed, bitterness is unknown for 82% of the beers). Therefore, a beer recommender system needs to take the existence of missing data into account.

[3] We use OWL 2 Manchester syntax for readability, but it does not actually provide any support for GCIs.

Data property	# Individuals	% Individuals
ABV	15317	100
IBU	2786	18
country	4365	28
styleRating	14378	94

Table 1: Statistics of available values for some features of the Beer ontology

Since the number of beers in the ontology is too high, we selected a subset to be used in the evaluation of our system. In particular, we defined another file (denoted $O2$) as a subset of $O1$, with 30 beer individuals and 24 brewery individuals. These beers are likely to be rather popular, to make it easier finding human experts that could evaluate them. The first three columns on Table 2 detail the selected beers, the alcohol degrees, and the linguistic labels, computed using Datil (we will explain later the procedure to obtain them). The table shows the label with the highest membership degree; note that in the case of Mahou Clásica there are two maxima (the membership degrees to *Low* and *Neutral* are the same ones). The list of beers include many examples from Spain (and, in particular, from Zaragoza), because most of the experts that took part in the evaluation live there. Note also that there are no examples with very low or very high alcohol.

3.2. Implementation

GimmeHop is a *knowledge-based recommender system* that receives user requirements for the items that s/he expects to receive (values for some attributes) and provides user with pull-based recommendations (a ranked list of beers). It is also a *context-aware recommender system* that takes into account contextual information in order to recommend items to users under certain circumstances. In general, context can be "any information that can be used to characterize the situation of an entity" [14]. In our case, we consider the position of the user, so it can be seen as a *location-based service*.

Language. GimmeHop works on Android Operating System (OS) mainly because the high availability of semantic reasoners working on Android (as most of them are written in Java, one of the two official programming languages for Android). Furthermore, it is nowadays

Beer	ABV	Datil label	# OK	% OK	# Valid reply	% Valid reply
Carling	4	Low	10	21.7	23	50
Guinness Draught	4.2	Low	2	4.3	43	93.5
Bud Light	4.2	Low	13	28.3	33	71.7
Pilsner Urquell	4.4	Low	11	23.9	32	69.6
Mort Subite Kriek	4.5	Low	4	8.7	24	52.2
Coronita	4.5	Low	18	39.1	46	100
Mahou Clásica	4.8	Low-Neutral	36	78.3	37	80.4
Quilmes Cristal	4.9	Neutral	18	39.1	33	71.7
Cruzcampo Premium Lager	5	Neutral	17	37	40	87
Amstel	5	Neutral	27	58.7	41	89.1
Asahi Super Dry	5	Neutral	8	17.4	25	54.3
Budweiser	5	Neutral	22	47.8	43	93.5
Franziskaner Hefe-Weissbier	5	Neutral	12	26.1	41	89.1
Heineken	5	Neutral	24	52.2	45	97.8
Ámbar CaesarAugusta	5.2	Neutral	9	19.6	29	63
Ámbar Especial	5.2	Neutral	24	52.2	38	82.6
Mahou 5 Estrellas	5.5	Neutral	26	56.5	39	84.8
BrewDog Punk IPA	6	Neutral	9	19.6	24	52.2
Alhambra 1925	6.4	Neutral	13	28.3	38	82.6
Hijos de Rivera 1906 Extra	6.5	Neutral	9	19.6	32	69.6
Leffe Blonde	6.6	Neutral	11	23.9	37	80.4
Ámbar Export	7	Neutral	9	19.6	33	71.7
Chimay Rouge	7	Neutral	9	19.6	27	58.7
Voll Damm	7.2	Neutral	2	4.3	38	82.6
Paulaner Salvator	7.9	Neutral	10	21.7	32	69.6
Pauwel Kwak	8.4	High	4	8.7	14	30.4
Delirium Tremens	8.5	High	12	26.1	27	58.7
Judas	8.5	High	16	34.8	33	71.7
Chimay Bleue	9	High	10	21.7	25	54.3
Ámbar 10	10	High	6	13	22	47.8

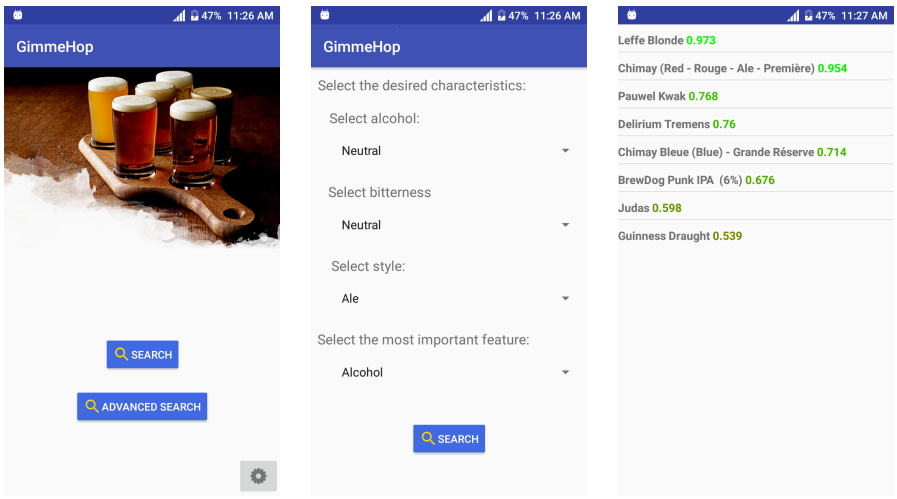
Table 2: List of beers in ontology O2 and replies of our experts.

the most popular mobile OS. Our prototype has been developed in Java 1.8 using the IDE Android Studio 5.3.3.

Queries. GimmeHop supports three types of queries or searches:

- Basic search: the input is the name (or part of the name) of a beer or a brewery. The output is a list of beers or breweries matching syntactically the query.
- Advanced search: the input is a beer type and, optionally, values (using linguistic labels) of ABV and/or IBU. The output is a list of beers retrieved using a semantic reasoner, together with the recommendation degrees (i.e., the satisfiability degree of the query). The list is decreasingly ordered by the recommendation degrees (we will explain later how to compute it). The user is also able to select some user preferences (the most important property).
- Similarity search: the input is a beer and the output is a list of similar beers with the similarity degrees.

User interface. Figure 7 (a) shows the main view of the app, where the user has two options, a (basic) search and an advanced search. Figure 7 (b) shows the form to submit an advanced search. The user must select the style. Optionally, s/he can set the degrees of alcohol and bitterness, using linguistic labels rather than numerical values (it is also possible to select "Indifferent"). The user can also choose the most important property between ABV, IBU, style rating, or indifferent. We wanted to make the interface as simple as possible, a possible extension would be to ask for a complete ordering of the three properties in terms of importance. Figure 7 (c) illustrates the output of the advanced search with respect to the query specified in Figure 7 (b). We can see that the beers are decreasingly ordered and that the numbers (the recommendation degrees) are colored to illustrate the importance of the recommendation (going from green to red as the quality of the recommendation degree decreases).
Figure 8 (a) shows the output of a basic search, a list of beers relevant to the name "Ámbar beer". The system retrieves all the coincidences with beer names in the ontology, without any numerical degree. If the user clicks on any beer presented in a list of results, s/he navigates to another page displaying information about the beer (name, brewery, style, bitterness, alcohol,



a) b) c)

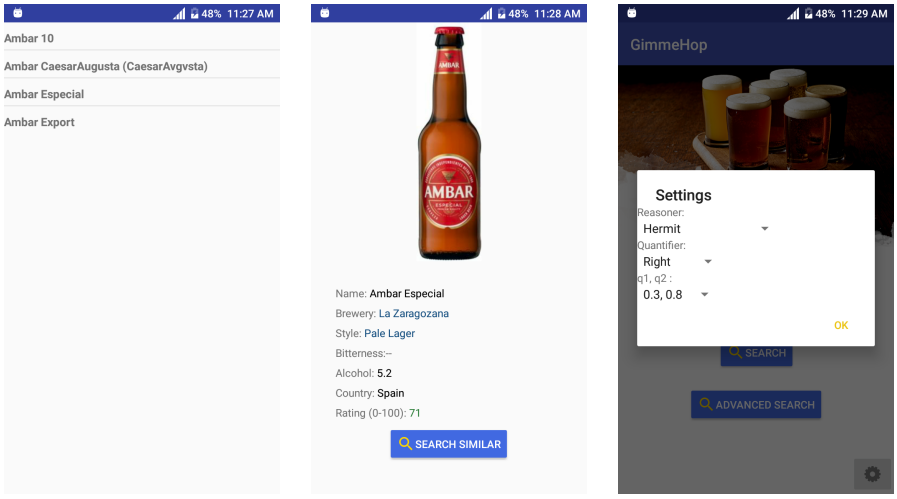
Figure 7: Some screenshots of GimmeHop: (a) initial page, (b) advanced search form, and (c) result of an advanced search

country, and rating). Figure 8 (b) illustrates this information when clicking on the "Ámbar especial" beer. If the user clicks on the button, a similarity search is performed. The results of the similarity search are similar to those of the advanced queries.

Local and remote modes. The app supports two ways to solve the queries: local or remote. Note in particular that this involves using a local semantic reasoner or an external one, respectively. We chose to use the same reasoners in both situations to make a fair comparison, but this does not need to be the case.

In the *local* mode, the user interacts with the graphic user interface, tasks are computed on the mobile, and eventually the user receives the results. The semantic reasoner is invoked when needed, using an Android service, and therefore it must work on a mobile device.

In the *remote* mode, the app uses a client/server architecture, TCP protocol, and sockets to communicate with an external server. The user side send requests, the server computes and filters the top–k results, and sends them to the client so that it can show them to the user. A single server is able to handle requests from different clients. The ontology is loaded and classified when the server is started. Note that the server computes all tasks even if they do



a) b) c)

Figure 8: Some screenshots of GimmeHop: (a) result of a basic search, (b) detailed information about a beer, and (c) settings

not require a semantic reasoner (e.g., the basic search) and that the semantic reasoner might not work on a mobile device.

Settings. App settings are depicted in Figure 8 (c). The user is able to configure the type of fuzzy quantifier and its parameters; recall that the we support right-shoulder (Figure 2 (d)), linear (Figure 2 (e)), and power functions (Figure 2 (f)).

Our app is integrated with two semantic reasoners, namely Hermit [18][4] and TrOWL.[5] TrOWL reasoner can be directly imported in Android projects, but Hermit cannot and required some changes in the code, summarized in [5]. It is possible to choose the reasoner when working on the local mode or when starting the remote server.

The app was designed to be as simple as possible, in the future we will consider allowing to configure the maximum number of answers (200, so far).

[4] http://www.hermit-reasoner.com
[5] http://trowl.org

Algorithms. *Basic search* only compares the name given by the user with all the names of the beers and breweries in the ontology. All strings are split in subwords if they contain the special character "_" (representing a blank space) or a blank space. The system retrieves those beers/breweries where two subwords coincide. As we will see later, advanced and similarity searches use a semantic reasoner, basic search does not.

Advanced search works in the following way. The submitted query includes a beer type T and two optional restrictions on the attributes ABV and IBU. The semantic reasoner solves an instance retrieval task (i.e., retrieving all instances of T). Next, for each retrieved beer b, we compute the rating degree $RatingDegree(b)$, the alcohol degree $ABVDegree(b)$, and the bitterness degree $IBUDegree(b)$. The *rating degree* consists simply in taking the styleRating of the beer, measured in [0,100], and normalizing it to [0,1]. The *alcohol degree* is computed as follows. If the query included a linguistic label for the ABV (i.e., if the value is different from "Indifferent"), denoted L_{abv}, and we know the ABV of the beer, denoted ABV_b, we compute the membership degree of the ABV to L_{abv}, $\mu_{L_{abv}}(ABV_b)$. The *bitterness degree* is computed similarly, but replacing the ABV with the IBU. Finally, the satisfiability degree of the query is obtained by combining the rating degree, the alcohol degree, and the bitterness degree using an aggregation operator:

$$SatDegree_Q(b) = @\Big(RatingDegree(b), ABVDegree(b), IBUDegree(b)\Big)$$

If the user selected a most important property, we use WMEAN; otherwise we use OWA. Note also that for some beers the values of styleRating and IBU are not available, so in general each beer requires a vector of weights with different size (the size of the vector k is the number of values to be aggregated).

It remains to explain how to obtain the k weights of the aggregation operators. If we are using OWA, we use quantified-guided aggregation and compute the degrees using Equation 3. After trying with several quantifiers, we selected the quantifier "Most" defined as a right-shoulder function with $q_1 = 0.3, q_2 = 0.8$. Note this formula nicely adapts to the fact that beers can have missing data. For example, for $k = 2$ this quantifier gives the vector [0.4,0.6], while for $k = 3$ vector is [0.067,0.667,0.267]. If we are using WMEAN, starting by computing a vector $Vaux$ using Equation 3. Then, we associate the highest number w_{max} in $Vaux$ to the most important property selected by the user. To the other $k-1$ properties,

we assign the same weight $(1-w_{max})/(k-1)$. As already mentioned, we deliberately chose to not ask for a complete ordering of preference between the attributes but it could be possible. As a final remark, note that this approach could be trivially generalized to consider more fuzzy attributes, such as the color and the price.

Similarity search is computed as follows. The input beer is compared to each other beer in the system. We compute the similarity between the styles, similarity between the ABVs, and the similarity between the IBU, and compute an average (so that the three criteria have the same importance). Again, this could be trivially generalized to consider more fuzzy attributes. To compute the similarity between the ABVs, we take the absolute difference of the ABVs and compute the membership degree to a left-shoulder function with $q_1 = 0.5$, $q_2 = 1.5$. That is, if the difference is at most 0.5° , the similarity is 1; if the difference is at least 1.5°, the similarity is 0; and for intermediate values, the similarity decreases as the difference increases. The similarity between the IBUs is similar, but the parameters of the left-shoulder function are different ($q_1 = 3$, $q_2 = 15$). Finally, the similarity between the beer styles is 1 if both beers belong to the same style, a value in (0,1) if they do not have to same style but belong to the same family (e.g., Ale), and 0 otherwise.

The full set of axioms in the ontology is only used to check the consistency. To speed up the app, when solving advanced and similarity searches, we only take into account subclass axioms (the class hierarchy).

Location. GimmeHop takes into account the user location (the country) to customize the recommendations. Google Play Services offer location APIs that facilitate adding location awareness to the app with automated location tracking, and other services like activity recognition. GimmeHop can get the location via network (cell tower and Wi-Fi signals) or GPS. We use Android's Network Location Provider because it works both indoors and outdoors, responds faster, and uses less battery power in comparison with GPS Provider. This is enough for use because the app does not require an optimal accuracy, only the country code, e.g., "BE" for Belgium.

After retrieving the user location, GimmeHop proceeds as follows. If the user submits a query Q and the reasoner retrieves a beer b with a recommendation degree $\mu_Q(b)$ and the user is located in the beer country, GimmeHop applies a fuzzy modifier to increment the

degree, assuming that closer beers are more likely to be found and preferable. We use the popular square root function, although other intensifying modifiers would be possible:

$$SatDegree_Q(b) = \sqrt{\mu_Q(b)}$$

4. Evaluation

In this section we firstly report our evaluation of the quality of the linguistic labels (Section 4.1), then an evaluation of the running time (Section 4.2) and the traffic data (Section 4.3). Finally, we discuss the overall behaviour of the system with the help of some sample queries (Section 4.4).

4.1. Evaluation of the Linguistic Labels

As already mentioned, Datil offers several choices to compute the linguistic labels associated to the alcohol and to the bitterness. In this section, we describe the evaluation of the quality of the results given by different clustering algorithms and parameters.

We invited 46 beer aficionados to evaluate the linguistic labels associated to the 30 beers in $O2$. We designed a webpage where each expert was asked to classify the alcohol of each beer using the following scale:

$$\{NoReply\ (0), VeryLow\ (1), Low\ (2), Neutral\ (3), High\ (4), VeryHigh\ (5)\}$$

Experts were specifically asked to select *NoReply* if they had never tried the beer or, more generally, did not feel qualified to answer properly (in the following, we will use the term valid answers to exclude no replies). They did not know the information about the numerical value of the ABV, they just answered according to their user experiences. Beers were presented sequentially, only one at a time. We restricted to the alcohol level, as we think that bitterness is much harder to evaluate, in particular if the answers are not given during a beer tasting. Then, we compared the answers given by the experts with the results given by Datil, selecting the best match.

Datil currently supports 3 clustering algorithms. For k-means and fuzzy c-means we tried with different number of clusters, namely 5 and 7. In mean shift the number of labels is not an input parameter; in all cases the value turned out to be 5. The value of Θ_1 was always 0; the values of Θ_2 were in {14,15,17,20}.

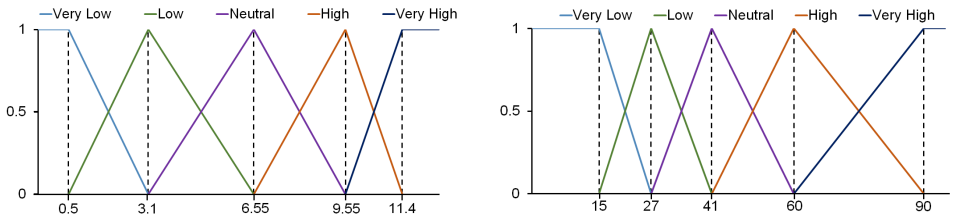
For each clustering algorithm, we take into account two measures:

- The number of coincidences, i.e., the cases when both the expert and Datil gave the same classification.
- The distance between the classification given by the expert and Datil. For instance, if an expert classifies some beer as having *low* alcohol and Datil chose *high*, the distance is $abs(2-4)=2$.

Note that some answers are not directly comparable, as the expert scale has 5 linguistic labels and some clustering algorithms produced 7. To compute the distance in such cases, we merged the two lowest values and the two greatest ones (that is, we merged *VeryVeryLow* and *VeryLow*, as well as *VeryVeryHigh* and *VeryHigh*).

Clustering	Θ_2	# Labels	# OK	% OK	Distance	Avg. distance
K–means	14	5	113	11.4	1598	1.606
		7	71	7.1	1888	1.897
	15	5	233	23.4	1138	1.144
		7	283	28.4	1009	1.014
	17	5	290	29.1	940	0.945
		7	88	8.8	1753	1.762
	20	5	110	11.1	1634	1.642
		7	65	6.5	1960	1.970
Fuzzy c–means	14	5	272	27.3	997	1.002
		7	177	17.8	1437	1.444
	15	5	285	28.6	995	1
		7	174	17.5	1456	1.463
	17	5	279	28	1032	1.037
		7	181	18.2	1432	1.439
	20	5	245	24.6	2416	2.428
		7	98	9.8	1830	1.839
Mean shift	14	5	286	28.7	936	0.941
	15	5	**401**	**40.3**	**775**	**0.779**
	17	5	386	38.8	779	0.783
	20	5	212	21.3	1181	1.187

Table 3: Results for different clustering algorithms and parameters.



(a) (b)

Figure 9: Linguistic labels (using Datil and mean-shift) for a) ABV and b) IBU

Table 3 shows the results: it includes the absolute number of coincidences (# OK) and the percentage of coincidences out of the valid answers (% OK), as well as the absolute sum of distances and the average distance (divided by the number of evaluated beers and the number of valid answers). In all cases the best results are obtained by using mean shift with $\Theta_2 = 15$; the linguistic labels corresponding to that case are depicted in Figure 9(a). For the sake of completeness, Figure 9(b) shows the linguistic labels associated to the bitterness (IBU). The best average distance (0.779) could seem too high at first sight, but the use of fuzzy logic ensures that there is usually a non-zero membership degree to the precedent and the subsequent linguistic labels, so the set of linguistic labels behaves well in practice.

The four latter columns of Table 2 show, for each beer, the number (absolute and percentage) of coincidences and the number (absolute and percentage) of valid answers, respectively, for the best clustering algorithm (mean shift with $\Theta_2 = 15$). We can see that there are beers not as popular as expected (such as Pauwel Kwak, with 30.4% of valid answers), and also popular beers with a small percentage of coincidences, such as Guinness (4.3%). Guinness is an example of counter-intuitive result: the majority of experts thought that it had a high ABV, which is not the case (4.2).

4.2. Evaluation of the Time

In this section we evaluate the running time of GimmeHop, both in the local and in the remote modes. We also try to determine the maximal number of individuals that are acceptable from the perspective of user experience.

We considered two mobile devices for these experiments: a tablet (denoted as A1) and a smartphone (denoted as A2). The tablet A1 is a Lenovo Yoga 2 10.1 (Android 5.0, Quad-core

1.86 GHz, Intel Atom Z3745, 2 GB RAM, released in 2014). The smartphone A2 is a ZTE Blade A610 (Android 6.0, Quad-Core 1.3 GHz ARM Cortex A-53, 2 GB RAM, released in 2016). We also used Amazon Web Services (AWS) and created an instance (denoted S1). S1 is a Ubuntu server 16.04 LST, amd64 xenial image, general purpose type t2.micro, 1 CPU and 1 GB RAM, located in the EU region (Paris). The versions of the semantic reasoners were Hermit 1.3.8 and TrOWL 1.5.

We considered two advanced searches ($Q1$ and $Q1'$), a basic search ($Q2$), and a similarity search ($Q3$). There are two advanced searches because the first one ($Q1$) might be significantly slower than the next ones (such as $Q1'$) . We also separated the loading time, as it is only necessary once. Note that this task could be run when the server starts and not when the client starts.

Firstly, Figure 10 shows the results on the server for both reasoners. Of course, remote reasoning is much faster than local reasoning. We can see that there are significant differences for both reasoners. TrOWL is much faster, although it does not support exact reasoning on OWL 2 (for more expressive languages than OWL 2, reasoning is approximate). We can see that indeed the first advanced query ($Q1$) is slower than the next one ($Q1'$). Furthermore, the advanced query is not always more complex than $Q2$ and $Q3$. Using TrOWL it is possible to get the answer to $Q1$ about $O1$ in 1.5 seconds (not including the loading time), and the next one in 0.9 seconds. Of course, times are much faster on the smaller $O2$.

We will show now the results on the Android devices. In this case, we additionally tested further subontologies of $O1$ with different numbers of individuals, between 30 and 15300. Figure 11 and 12 show the results of $Q1$ and $Q1'$ on A1 and A2 using HermiT and TrOWL, respectively. Similarly, Figure 13 and 14 show the results for $Q2$ and $Q3$. Again, TrOWL was always faster than HermiT, and device $A2$ was always slower than $A1$ (the differences are much higher when using HermiT). There are some missing data because local reasoning was not possible with ontologies with 10000 individuals or more. As expected, $Q1$ was slower than $Q1'$, $Q2$, and $Q3$. When using TrOWL and the ontology with 250 individuals there was a strange outlier on both $A1$ and $A2$, as this ontology is a superset of $O2$ and a subset of the ontology with 500 individuals.

Figure 15 summarizes the result of the first advanced query (including the loading time) for the three devices, the two reasoners, and different ontology sizes. We could say that remote

reasoning is feasible even with all the individuals, but local reasoning requires a smaller number. In such case, we must check that the latency of the first query is not much longer than the average time that mobile users are willing to wait. On $A2$ it seems feasible handling up to 2000 beer individuals; the first advanced query might take almost 16 seconds, but the next ones take less than 2 seconds. On $A1$ it seems feasible handling up to 3000 beer individuals.

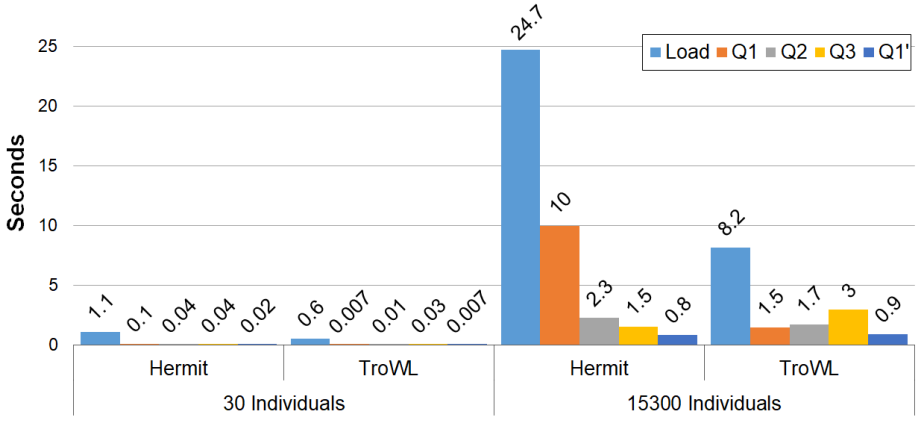


Figure 10: Running time of Load and all queries on S1

4.3. Evaluation of the Data Traffic

Users of apps requiring Internet connection are very often concerned of the data traffic, so we decided to investigate the cost of GimmeHop when using a remote reasoner.

In order to measure the data traffic of our application, we use the "Data Usage Monitoring" (DUM)[6] free app and the app-info utility of Android. The DUM makes it possible to analyze the data in a deeper way, as it provides not only mobile data but also Wi–Fi data, and makes it possible to obtain both sent and received data.

We considered the remote reasoning approach, and two ontologies with 150 ($O3$) and 15317 beers ($O1$). The reason to define $O3$ is that the size of $O2$ is appropriate for the evaluation of the linguistic labels, but it seems not enough to evaluate the data traffic.

We considered the following sequence of queries:

[6] http://play.google.com/store/apps/details?id=com.jsk.datausagemonitor

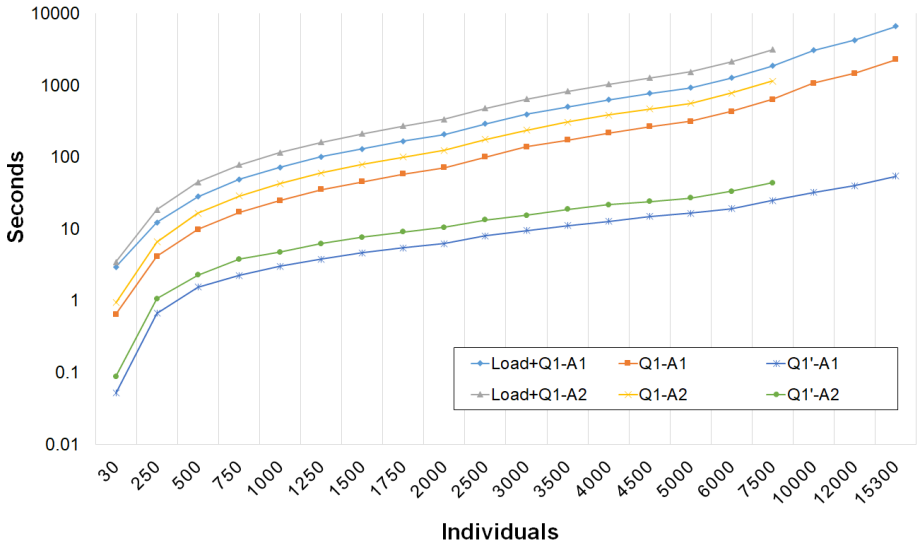


Figure 11: Running time of Load+Q1, Q1 and Q1' on HermiT

- Firstly, we considered an advanced query. We selected the most populated families and half of the times no other specification about the other attributes (ABU and IBU). This led to the worst case, when the number of results that the server sends back to the user is maximal.
- Then, we submitted a basic query, asking to retrieve a beer from its name. We also clicked on the result to display the beer information, including its image.
- Finally, we asked to retrieve similar beers to that one.

This sequence was repeated 12 times and we computed the average values. Table 4 presents the results of our evaluation in KB. The results show that the three methods used to measure the traffic coincide, and we think that the results are acceptable even for the very large ontology $O1$ (recall that the times include 3 queries).

Recall that the server currently only sends the top k results back, with $k = 200$, and note that data traffic could be optimized by making k a configurable parameter and setting smaller values.

We also noticed that selecting the value of the attributes on the advanced search did not

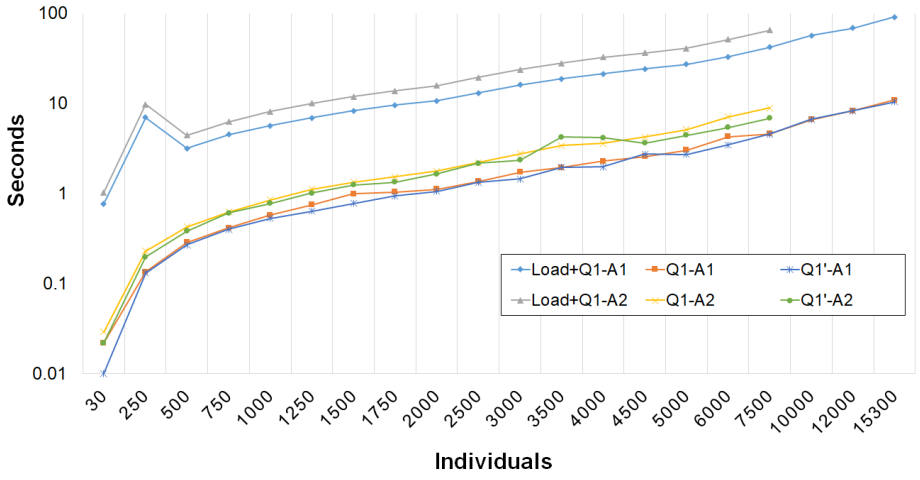


Figure 12: Running time of Load+Q1, Q1 and Q1' on TroWL

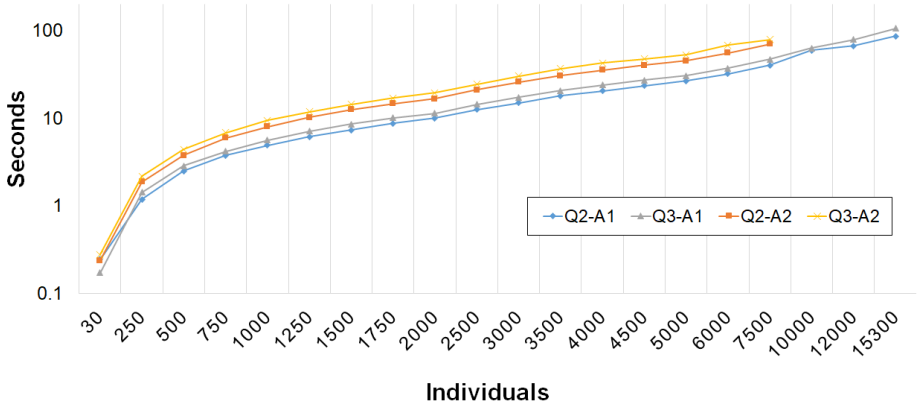


Figure 13: Running time of Q2 and Q3 on HermiT

# Individuals	Wi–Fi	DUM	App–info
150	56.33±6.11	54.85±6.27	54.83±6.19
15317	132.57±1.5	132.19±0.69	132.5 ±0.8

Table 4: Data traffic for a sequence of 3 queries (KB)

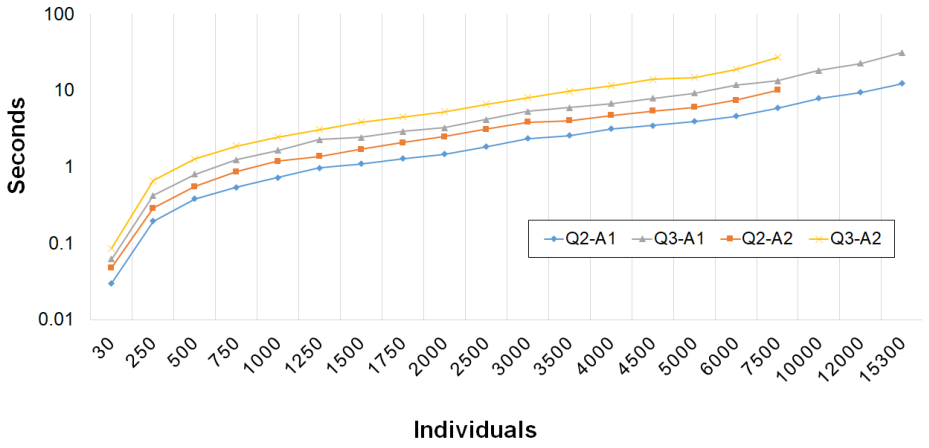


Figure 14: Running time of Q2 and Q3 on TrOWL

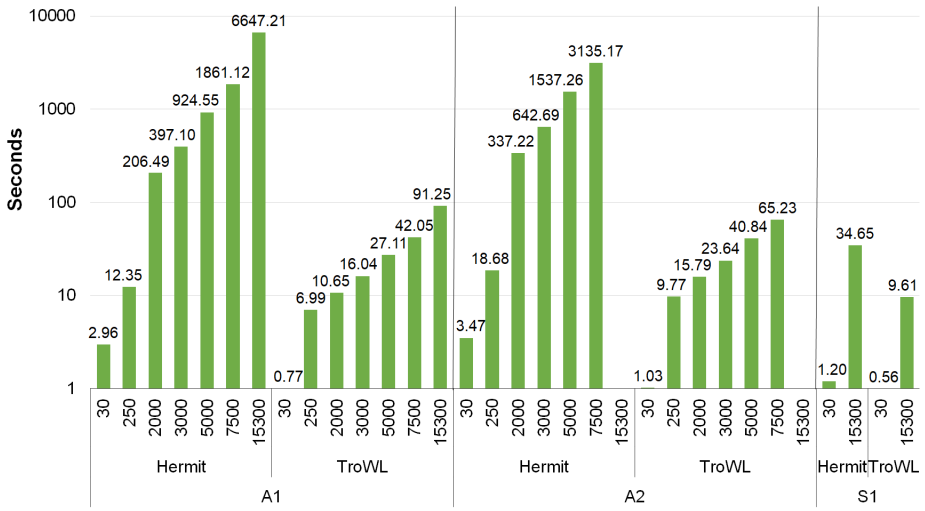


Figure 15: Running time of Load+Q1 on all devices and reasoners

make an impact in our cases, although this is not always the case. In ontology $O1$, the number of individuals was 200 even after restricting the attributes. In ontology $O2$, the number of individuals could be slightly different but without changes in the total data traffic.

When using local reasoning, retrieving the picture of a beer is the only data-consuming operation. We measured that the average traffic data is about 10.3 KB, although there is a

big standard deviation because pictures can have very different file sizes.

4.4. Evaluation of the System and Query Examples

Table 5 shows 3 examples of advanced queries. For each of them, the degree of satisfiability of each beer is shown in the two last columns (both when the user location is unknown and when the user is located in Belgium, respectively). In all three queries the user asks for an Ale beer, with a neutral alcohol and a neutral bitterness. The difference is that in the two first queries the user selects a preference in the most important attribute (alcohol and rating, respectively), so a weighted mean is used to combine the information. However, in the third query no attribute is selected as the most important, so an OWA operator is used.

Beer	ABV	IBU	Rating	Country	Partial degrees	Degree Country=?	Degree Country=BE
Query (WMEAN): ABV=Neutral, IBU=Neutral, Style=Ale, **Important property=Alcohol**							
Leffe Blonde	6.6	-	90	Belgium	[abv=0.98, rating=0.9]	0.95 (1st)	0.97 (1st)
Chimay Rouge	7	-	100	Belgium	[abv=0.85, rating=1]	0.91 (2nd)	0.95 (2nd)
Pauwel Kwak	8.4	-	90	Belgium	[abv=0.38, rating=0.9]	0.59 (4th)	0.77 (3rd)
Delirium Tremens	8.5	-	92	Belgium	[abv=0.35, rating=0.92]	0.57 (5th)	0.76 (4th)
Chimay Bleue	9	-	100	Belgium	[abv=0.18 , rating=1]	0.51 (7th)	0.71 (5th)
BrewDog Punk IPA	6	60	69	Scotland	[abv=0.84, rating=0.69, ibu=0]	0.67 (3rd)	0.67 (6th)
Judas	8.5	-	37	Belgium	[abv=0.35 , rating=0.37]	0.35 (8th)	0.6 (7th)
Guinness Draught	4.2	-	87	Ireland	[abv=0.31, rating=0.87]	0.53 (6th)	0.53 (8th)
Query (WMEAN): ABV=Neutral, IBU=Neutral, Style=Ale, **Important property=Rating**							
Chimay Rouge	7	-	100	Belgium	[abv=0.85, rating=1]	0.94 (1st)	0.97 (1st)
Leffe Blonde	6.6	-	90	Belgium	[abv=0.98, rating=0.9]	0.93 (2nd)	0.97 (2nd)
Pauwel Kwak	8.4	-	90	Belgium	[abv=0.38, rating=0.9]	0.69 (3rd)	0.83 (3rd)
Delirium Tremens	8.5	-	92	Belgium	[abv=0.35, rating=0.92]	0.69 (4th)	0.83 (4th)
Chimay Bleue	9	-	100	Belgium	[abv=0.18, rating=1]	0.67 (5th)	0.82 (5th)
Guinness Draught	4.2	-	87	Ireland	[abv=0.31, rating=0.87]	0.65 (6th)	0.65 (6th)
Judas	8.5	-	37	Belgium	[abv=0.35, rating=0.37]	0.36 (8th)	0.6 (7th)
BrewDog Punk IPA	6	60	69	Scotland	[abv=0.84, rating=0.69, ibu=0]	0.6 (7th)	0.6 (8th)
Query (OWA): ABV=Neutral, IBU=Neutral, Style=Ale, **Important property=Indifferent**							
Leffe Blonde	6.6	-	90	Belgium	[abv=0.98 rating=0.9]	0.93 (1st)	0.97 (1st)
Chimay Rouge	7	-	100	Belgium	[rating=1 abv=0.85]	0.91 (2nd)	0.95 (2nd)
Pauwel Kwak	8.4	-	90	Belgium	[rating=0.9 abv=0.38]	0.59 (3rd)	0.77 (3rd)
Delirium Tremens	8.5	-	92	Belgium	[rating=0.92 abv=0.35]	0.58 (4th)	0.76 (4th)
Chimay Bleue	9	-	100	Belgium	[rating=1 abv=0.18]	0.51 (7th)	0.71 (5th)
Judas	8.5	-	37	Belgium	[rating=0.37 abv=0.35]	0.36 (8th)	0.6 (6th)
Guinness Draught	4.2	-	87	Ireland	[rating=0.87 abv=0.31]	0.54 (5th)	0.54 (7th)
BrewDog Punk IPA	6	60	69	Scotland	[abv=0.84 rating=0.69 ibu=0]	0.52 (6th)	0.52 (8th)

Table 5: Results for 3 sample queries

The first column of Table 5 includes the name of the beer. Columns 2–5 include the values of some features (ABV, IBU, style rating, and country); note that some values are missing. These values are the same ones for each query, but the order of the beers might be

different. Column 6 includes the values to be aggregated, that is, the membership degrees to the fuzzy membership functions defined over ABV and IBU, and the normalized rating.

Note that user preference indeed plays a role when ordering the beers. For instance, the best beer for the first query is Leffe Blonde, but the best beer for the second one is Chimay Rouge. Note that the user location also plays a role in the recommendation. For example, in the first query BrewDog Punk IPA drops from the third position to the sixth one when taking into account the user location.

We also tried three fuzzy quantifiers described using a right-shoulder, a linear, and a power function. It seems that the best results were obtained using the right-shoulder. For example, an effect of the power function is that the weight vector is always ordered in increasing order (if $q < 1$) or decreasing order (if $q > 1$).

We carefully checked this and other similar examples and concluded that the behaviour of the system is reasonable when providing the recommendations. The final degrees might be too small for the user, but the system is effective at providing an ordering among the beers. In several cases where the user was not happy with the result, the reason was that s/he was wrong about the real ABV/type of the beer.

5. Related Work

This section reviews some related work. Our aim is to highlight our contribution with the previous work on the domain, fuzzy ontologies, and semantic apps. We also overview the previous work on semantic reasoners to motivate not having used a reasoner specifically built for mobile devices.

Beer ontologies. There is a previous effor to build a Beer ontology.[7] However, the ontology only contains 19 beer types and 9 beers. Another limitation is that the only existing axioms are subclass/subproperty axioms. On the contrary, we impose some conditions on the beer types (such as necessary conditions or concepts disjointness) and include data property assertions representing attributes of each beer instance. Furthermore, we include fuzzy datatype definitions allowing to deal with linguistic definitions of some attributes.

[7] http://dbs.uni-leipzig.de/files/coma/sources/fd/beer.owl

Fuzzy ontologies. Fuzzy ontologies have proved to be useful in several applications, such as information retrieval [8, 52], image interpretation [12], the Semantic Web and the Internet [45], ambient intelligence [15], ontology merging [49], matchmaking [36], decision making [31], summarization [26], robotics [16], diabetes diagnosis [17], construction [20], or gait recognition [21], and many others [30, 35, 37, 46]. For a more detailed overview, we refer the reader to [10]. Some of these applications only use fuzzy ontologies for knowledge representation (e.g., [26]). In other cases, applications also use a fuzzy ontology reasoner (e.g., [16]). In this work, we show that a classical crisp reasoner can sometimes be used if one manages fuzzy logic operators in an explicit way.

Semantic reasoners for mobile devices. There are several possibilities to do semantic reasoning on mobile devices [3]. The most direct way is to use an *external* solution, relying on servers on the cloud which would perform all the calculations. The main advantage is that one can consider a server which is as powerful as required by the application. However, in ubiquitous and mobile scenarios where context-awareness and privacy preserving play a crucial role, sending sensitive data to a remote server might be an important privacy breach, and even sending non-sensitive data might be dangerous as it could enable the inference of sensitive information. Furthermore, this requires assuming that the connectivity is fast and stable enough, but this is not often the case in mobile computing environments. It is also possible trying to use a *local* solution, which can be challenging because of the limitations of mobile devices in terms of CPU power, memory, or battery. Indeed, there is some evidence that reasoning time is only affordable in small or not very expressive ontologies [5]. For the sake of completeness, let us also mention that it is possible to develop *hybrid* strategies.

There are two options to perform local semantic reasoning:

- To reuse or port previously existing ones. *AndroidSemantic* project made 9 reasoners available for Android devices [5] (including *HermiT* [18] and *TrOWL* [48]). To date, there are no similar ports for iOS devices and none of the previously existing reasoners are implemented in the languages natively supported on iOS (Objective-C or Swift).

- To implement specific reasoners. Currently, there are implementations for the following mobile operating systems:

- J2ME: COROR [47], μ-OR [1], *Pocket KRHyper* [42], and the systems in [25, 33].
- Android: *MiniME* [39][8] and the system in [53].
- Windows Mobile: *mTableau* [44] and MiRE4OWL [23].
- iOS: MiniME-Swift [38].[9]
- Others: The system in [40] is implemented in CLIPS and there is no information about *Delta* [32].

Currently, only three of these specific reasoners support OWL 2 RL [40, 47, 53], and the other ones cannot fully support OWL 2 or any of its profiles. Because the expressivity of our ontology is OWL 2 EL, we needed to reuse existing reasoners not specifically designed for mobile devices.

Semantic apps. The roots of semantic apps can be traced back to knowledge mobilization (KMob), which consists of making knowledge available for real-time use in a form which is adapted to the context of use and to the needs and cognitive profile of the user [19]. One of the main requisites of KMob systems is that they should be ubiquitious and, in particular, accessible from mobile devices. Our system was indeed designed to meet some of the common requirements of KMob systems, such as being proactive, integrative, context-aware, declarative, concise, extensible, and easily maintainable.

Early examples of semantic application for mobile systems were very different to the more recent examples, and they typically relied on external servers. For example, *PDA2* system supports the diagnosis of psychological disorders on PDAs [13]. PDA2 uses an OWL 2 ontology to represent the relevant knowledge and accesses a semantic reasoner stored on an external server.

Some examples using local semantic reasoners include location-based services providers (e.g., *Sherlock* [57] can infer that both a cab and a tram are interesting for a certain mobile user, given the information obtained from sensors on his/her device such as the location

[8] http://sisinflab.poliba.it/swottools/minime
[9] http://sisinflab.poliba.it/swottools/minime-swift

and time) and navigation applications (e.g., *MAR* is a mobile augmented reality explorer to discover points of interest [39]).

The interested reader can find more semantic apps in [58]. Unfortunately, none of the existing semantic apps use fuzzy logic (with an exception that we will discuss below) or supports both local and external reasoning, as our system does.

Semantic apps using fuzzy logic. To the best of our knowledge, there is only one previous application of fuzzy ontologies working on mobile devices: a wine recommendation system [31]. The authors represent wine attributes such as price, alcohol level, sugar, or acidity using fuzzy membership functions. Then, a Java application uses fuzzyDL reasoner [7] to solve instance retrieval queries, where the output is a list of wines that satisfy some features combined using an OWA operator. This application is stored on a server and can be accessed from an Android app.

However, there are several differences with our approach. The main one is that the authors do not use a semantic reasoner running on a mobile device but require it to be stored on an external server. On the contrary, we support both reasoning mechanisms: using a local reasoner or storing it on an external server. Furthermore, the authors require a concrete fuzzy ontology reasoner, while we can use any standard OWL 2 EL reasoner (of course, we need further computations to take care of the fuzzy part). We also take into account user preferences (by supporting weighted mean aggregation in addition to OWA) and manage the user context in a different way (by using fuzzy hedges). Last but not least, we address the problem of dealing with incomplete data by using qualified-guided OWA. It is worth to mention that the wine reccommender system includes a procedure to reach a consensus between multiple users that could also be adopted in our app.

6. Conclusions and Future Work

In this paper we have presented GimmeHop, a recommender system for Android mobile devices, using fuzzy ontologies and semantic reasoners. GimmeHop is a proof of concept showing that fuzzy logic, semantic technologies, and both local and remote reasoning can be combined in mobile applications.

The application domain, beers, is a hot topic which is receiving a notable attention in the last years. In fact, two local companies are interested in the results of our project.

GimmeHop is able to deal with user context (in particular the location) by using fuzzy hedges, with user preferences by using weighted mean aggregation, and with incomplete data by using quantifier-guided OWA to provide weighting vectors with different sizes.

This paper includes an extensive evaluation of several features of the system, namely the data traffic, the running time, the quality of the recommendations and the quality of the linguistic labels. Our experiments about the data traffic and running time show that remote reasoning is feasible and cheap (in terms of both data traffic and time). Local reasoning is only feasible if we limit the number of individuals, the tested devices were able to support 2000–3000 beers. We think that this number would be enough in practice for most bars or stores interested in recommending beers to their users.

Facing a real-world problem has also made it possible to evaluate a previous tool for fuzzy ontology learning, Datil. Our current research identified a new requirement for Datil, the need for minimal and maximal thresholds, which is already implemented. Furthermore, a group of experts evaluated the quality of the linguistic labels given by Datil and helped to identify the best learning strategy.

Future work. There are a lot of directions for our future work. The main one is to take user feedback into account for future recommendations. So far, we take into account a global rating defined by the community, but it would also be interesting to take into account the user personal rating.

Another possible extension is providing social recommendations, i.e., recommending products that a user with similar profiles liked. This would require a characterization of the user profile and letting the aggregation operator take into account the similarity between user profiles.

As already discussed, it would also be desirable to build more complex representations of the possible values for some attributes of a beer, such as flavor, aroma, or foam. Furthermore, the individuals of the ontology could include values for the attributes price, color, or turbidity. In that case, we could use Datil to compute their associated linguistic labels and to include them in the recommendation process as well.

Our definition of *context* could also be extended to consider other factors, such as the temperature (e.g., Berliner Weiße is particularly appropriate for summer) or food to combine with (e.g., Guinness is a good choice to combine with a chocolate cake).

Moreover, we could introduce fuzziness at other levels. For instance, we could assume that a beer belongs to a type with some degree, or that a beer type is a subtype of another one with some degree. In this case, data acquisition seems particularly challenging.

Last but not least, it would also be possible to consider more general aggregation operators (not only OWA or WMEAN), fuzzy quantifiers (not only right-shoulder, linear, and power functions), or intensifying fuzzy modifiers (not only the square root).

Acknowledgment

I. Huitzil was partially funded by Universidad de Zaragoza - Santander Universidades (Ayudas de Movilidad para Latinoamericanos - Estudios de Doctorado). I. Huitzil and F. Bobillo were partially supported by the projects TIN2016-78011-C4-3-R (AEI/ FEDER, UE), JIUZ-2018-TEC-02 (Fundación Ibercaja y Universidad de Zaragoza), and DGA/FEDER.

We are very grateful to Professor Miguel Delgado for many useful lessons along the years. In particular, he was a pioneer in (among many other things) promoting knowledge mobilization by bringing semantic technologies into mobile devices, and combining ontologies and fuzzy logic.

References

[1] S. Ali and S. Kiefer. μ-OR – A micro OWL DL reasoner for ambient intelligent devices. In *Proceedings of the 4th International Conference on Grid and Pervasive Computing (GPC 2009)*, volume 5529 of *Lecture Notes in Computer Science*, pages 305–316. Springer, 2009.

[2] J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Advanced Applications in Pattern Recognition. Plenum Press, 2nd edition, 1987.

[3] C. Bobed, F. Bobillo, E. Mena, and J. Z. Pan. On serializable incremental semantic reasoners. In *Proceedings of the 9th International Conference on Knowledge Capture (K-CAP 2017)*, pages 187–190. ACM, December 2017.

[4] C. Bobed, R. Yus, F. Bobillo, S. Ilarri, J. Bernad, E. Mena, R. Trillo-Lado, and Á. L. Garrido. Emerging semantic-based applications. In M. Workman, editor, *Semantic Web - Implications for Technologies and Business Practices*, pages 41–85. Springer Verlag, 2016.

[5] C. Bobed, R. Yus, F. Bobillo, and E. Mena. Semantic reasoning on mobile devices: Do androids dream of efficient reasoners? *Journal of Web Semantics*, 35(4):167–183, 2015.

[6] F. Bobillo and U. Straccia. Fuzzy ontology representation using OWL 2. *International Journal of Approximate Reasoning*, 52(7):1073–1094, 2011.

[7] F. Bobillo and U. Straccia. The fuzzy ontology reasoner fuzzyDL. *Knowledge-Based Systems*, 95:12–34, 2016.

[8] S. Calegari and E. Sanchez. Object-fuzzy concept network: An enrichment of ontologies in semantic information retrieval. *Journal of the American Society for Information Science and Technology*, 59(13):2171–2185, 2008.

[9] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995.

[10] V. Cross and S. Chen. Fuzzy ontologies: State of the art revisited. In *Proceedings of the 37th Conference of the North American Fuzzy Information Processing Society (NAFIPS 2018)*, volume 831 of *Communications in Computer and Information Science*, pages 230–242. Springer, 2018.

[11] B. Cuenca-Grau, I. Horrocks, B. Motik, B. Parsia, P. F. Patel-Schneider, and U. Sattler. OWL 2: The next step for OWL. *Journal of Web Semantics*, 6(4):309–322, 2008.

[12] S. Dasiopoulou, I. Kompatsiaris, and M. G. Strintzis. Investigating fuzzy DLs-based reasoning in semantic image analysis. *Multimedia Tools and Applications*, 46(2):331–370, 2010.

[13] M. Delgado, J. Gómez-Romero, P. J. Magaña, and R. Pérez-Pérez. A flexible architecture for distributed knowledge based systems with nomadic access through handheld devices. *Expert Systems with Applications*, 29(4):965–975, 2005.

[14] A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5:4–7, 2001.

[15] N. Díaz-Rodríguez, M. Pegalajar-Cuéllar, J. Lilius, and M. Delgado. A fuzzy ontology for semantic modelling and recognition of human behaviour. *Knowledge-Based Systems*, 66:46–60, 2014.

[16] M. Eich, R. Hartanto, S. Kasperski, S. Natarajan, and J. Wollenberg. Towards coordinated multirobot missions for lunar sample collection in an unknown environment. *Journal of Field Robotics*, 31(1):35–74, 2014.

[17] S. El-Sappagh, M. Elmogy, and A. M. Riad. A fuzzy-ontology-oriented case-based reasoning framework for semantic diabetes diagnosis. *Artificial Intelligence in Medicine*, 65(3):179–208, 2015.

[18] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang. HermiT: An OWL 2 reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014.

[19] J. Gómez-Romero. *Knowledge Mobilization: Architectures, Models and Applications*. PhD thesis, University of Granada, Spain, 2008.

[20] J. Gómez-Romero, F. Bobillo, M. Ros, M. Molina-Solana, M. D. Ruiz, and M. J. Martín-Bautista. A fuzzy extension of the semantic building information model. *Automation in Construction*, 57:202–212, 2015.

[21] I. Huitzil, L. Dranca, J. Bernad, and F. Bobillo. Gait recognition using fuzzy ontologies and Kinect sensor data. *International Journal of Approximate Reasoning*, 113:354–371, 2019.

[22] I. Huitzil, U. Straccia, N. Díaz-Rodríguez, and F. Bobillo. Datil: Learning fuzzy ontology datatypes. In *Proceedings of the 17th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 2018), Part II*, pages 100–112, 2018.

[23] T. Kim, I. Park, S. J. Hyun, and D. Lee. MiRE4OWL: Mobile rule engine for OWL. In *Proceedings of the 2nd IEEE International Workshop on Middleware Engineering (ME 2010)*, pages 317–322, 2010.

[24] G. J. Klir and B. Yuan. *Fuzzy sets and fuzzy logic: theory and applications*. Prentice-Hall, Inc., 1995.

[25] M. Koziuk, J. Domaszewicz, R. O. Schoeneich, M. Jablonowski, and P. Boetzel. Mobile context-addressable messaging with DL-Lite domain model. In *Proceedings of the 3rd European Conference on Smart Sensing and Context (EuroSSC 2008)*, volume 5279 of *Lecture Notes in Computer Science*, pages 168–181. Springer, 2008.

[26] C.-S. Lee, Z.-W. Jian, and L.-K. Huang. A fuzzy ontology and its application to news summarization. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 35(5):859–880, 2005.

[27] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. Dbpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[28] S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[29] T. Lukasiewicz and U. Straccia. Managing uncertainty and vagueness in description logics for the semantic web. *Journal of Web Semantics*, 6(4):291–308, 2008.

[30] C. Martínez-Cruz, A. van der Heide, D. Sánchez, and G. Triviño. An approximation to the computational theory of perceptions using ontologies. *Expert Systems with Applications*, 39(10), 2012.

[31] J. A. Morente-Molinera, R. Wikström, E. Herrera-Viedma, and C. Carlsson. A linguistic mobile decision support system based on fuzzy ontology to facilitate knowledge mobilization. *Decision Support Systems*, 81:66–75, 2016.

[32] B. Motik, I. Horrocks, and S. M. Kim. Delta-reasoner: A Semantic Web reasoner for an intelligent mobile platform. In *Proceedings of the 21st World Wide Web Conference (WWW 2012), Companion Volume*, pages 63–72, 2012.

[33] F. Müller, M. Hanselmann, T. Liebig, and O. Noppens. A tableaux-based mobile DL reasoner - An experience report. In *Proceedings of the 19th International Workshop on Description Logics (DL 2006)*, volume 189 of *CEUR Workshop Proceedings*, 2006.

[34] M. A. Musen. The protégé project: a look back and a look forward. *AI Matters*, 1(4):4–12, 2015.

[35] T. D. Noia, M. Mongiello, F. Nocera, and U. Straccia. A fuzzy ontology-based approach for tool-supported decision making in architectural design. *Knowledge and Information Systems*, 58(1):83–112, 2019.

[36] A. Ragone, U. Straccia, F. Bobillo, T. D. Noia, and E. D. Sciascio. Fuzzy bilateral matchmaking in e-marketplaces. In *Proceedings of the 12th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2008), Part III*, volume 5179 of *Lecture Notes in Computer Science*, pages 293–301. Springer-Verlag, 2008.

[37] J. A. Rodger. A fuzzy linguistic ontology payoff method for aerospace real options valuation. *Expert Systems with Applications*, 40(8), 2013.

[38] M. Ruta, F. Scioscia, F. Gramegna, I. Bilenchi, and E. D. Sciascio. Mini-ME Swift: the first OWl reasoner for iOS. In *Proceedings of the 16th Extended Semantic Web Conference (ESWC 2019)*, Lecture Notes in Computer Science. Springer, 2019.

[39] F. Scioscia, M. Ruta, G. Loseto, F. Gramegna, S. Ieva, A. Pinto, and E. D. Sciascio. A mobile matchmaker for the ubiquitous semantic web. *International Journal on Semantic Web and Information Systems*, 10(4):77–100, 2014.

[40] C. Seitz and R. Schönfelder. Rule-based OWL reasoning for specific embedded devices. In *Proceedings of the 10th International Semantic Web Conference (ISWC 2011), Part II*, volume 7032 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 2011.

[41] R. Senge and E. Hüllermeier. Top-down induction of fuzzy pattern trees. *IEEE Transactions on Fuzzy Systems*, 19(2):241–252, 2011.

[42] A. Sinner and T. Kleemann. KRHyper - In your pocket. In *Proceedings of the 20th International Conference on Automated Deduction (CADE-20)*, volume 3632 of *Lecture Notes in Computer Science*, pages 452–458. Springer, 2005.

[43] S. Staab and R. Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004.

[44] L. Steller, S. Krishnaswamy, and M. M. Gaber. Enabling scalable semantic reasoning for mobile services. *International Journal on Semantic Web and Information Systems*, 5(2):91–116, 2009.

[45] U. Straccia. *Foundations of Fuzzy Logic and Semantic Web Languages*. CRC Studies in Informatics Series. Chapman & Hall, 2013.

[46] U. Straccia, E. Tinelli, S. Colucci, T. D. Noia, and E. D. Sciascio. Semantic-based top-k retrieval for competence management. In *Proceedings of the 18th International Symposium on Foundations of Intelligent Systems (ISMIS 2009)*, volume 5722 of *Lecture Notes in Computer Science*, pages 473–482. Springer, 2009.

[47] W. Tai, J. Keeney, and D. O'Sullivan. Resource-constrained reasoning using a reasoner composition approach. *Semantic Web*, 6(1):35–59, 2015.

[48] E. Thomas, J. Z. Pan, and Y. Ren. TrOWL: Tractable OWL 2 reasoning infrastructure. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010)*, 2010.

[49] K. Todorov, C. Hudelot, A. Popescu, and P. Geibel. Fuzzy ontology alignment using background knowledge. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 22(1):75–112, 2014.

[50] V. Torra and Y. Narukawa. *Modeling decisions - Information fusion and aggregation operators*. Springer, 2007.

[51] W3C. OWL 2 Web Ontology Language profiles: OWL 2 EL, 2009. http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/#OWL_2_EL.

[52] M. Wallace. Ontologies and soft computing in flexible querying. *Control and Cybernetics*, 38(2):481–507, 2009.

[53] W. V. Woensel and S. S. R. Abidi. Optimizing semantic reasoning on memory-constrained platforms using the RETE algorithm. In *Proceedings of the 15th Extended*

Semantic Web Conference (ESWC 2018), volume 10843 of *Lecture Notes in Computer Science*, pages 682–696. Springer, 2018.

[54] R. R. Yager. On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):183–190, 1988.

[55] R. R. Yager. Connectives and quantifiers in fuzzy sets. *Fuzzy Sets and Systems*, 40(1):39–75, 1991.

[56] R. R. Yager. Quantifier guided aggregation using OWA operators. *International Journal of Intelligent Systems*, 11(1):49–73, 1996.

[57] R. Yus, E. Mena, S. Ilarri, and A. Illarramendi. SHERLOCK: Semantic management of location-based services in wireless environments. *Pervasive and Mobile Computing*, 15:87–99, 2014.

[58] R. Yus and P. Pappachan. Are apps going semantic? A systematic review of semantic mobile applications. In *Proceedings of the 1st International Workshop on Mobile Deployment of Semantic Technologies (MoDeST 2015)*, volume 1506 of *CEUR Workshop Proceedings*, pages 2–13, 2015.

[59] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[60] F. Zhang, J. Cheng, and Z. Ma. A survey on fuzzy ontologies for the semantic web. *Knowledge Engineering Review*, 31(3):278–321, 2016.