Aggregation Operators in Geospatial Queries for Open Street Map*

Jesús M. Almendros-Jiménez, Antonio Becerra-Terón and Manuel Torres

Information Systems Group. University of Almería
04120-Spain. {jalmen,abecerra,mtorres}@ual.es

Abstract. One of the most stablished Volunteered Geographic Information (VGI) systems is Open Street Map (OSM) offering data from the earth of urban and rural maps. Recently [1], we have presented a library for querying OSM data with the XML query language XQuery. This library is based on the well-known spatial operators defined by Clementini and Egenhofer, providing a repertoire of XQuery functions which encapsulates the search on the XML document representing a layer of OSM, and makes the definition and composition of queries on top of OSM layers easier. This paper goes towards the incorporation in the library of aggregation operators in order to be able to express queries involving data summarization and ranking. A rich repertoire of aggregation operators has been defined which, in combination with the previously proposed library, makes possible to easily formulate aggregation-based queries. Also we present a Web-based tool, called *XOSM (XQuery for Open Street Map)*, developed in our group, that uses the proposed library to query and visualize OSM data.

1 Introduction

Volunteered Geographic Information (VGI) is a term introduced by Goodchild [12] to describe geographic information systems based on crowdsourcing, in which users collaborate to a collection of spatial data of urban and rural areas of the earth. VGI makes available a very large resource of geographic data. The exploitation of data coming from such resources requires an additional effort in the form of tools and effective processing techniques. *Open Street Map* (OSM) [5] is one of the most relevant VGI systems, with almost two millions of registered users. OSM data can be visualized from the OSM web site[1], and many applications[2] have also built for the handling of maps.

In spite of the growing interest in OSM and the fact many tools have been developed, the main tasks tools are able to carry out are edition, export, rendering, conversion, analysis, routing and navigation, but less attention has been

* This work was funded by the EU ERDF and the Spanish Ministry of Economy and Competitiveness (MINECO) under Project TIN2013-44742-C4-4-R, and by the Andalusian Regional Government (Spain) under Project P10-TIC-6114.

[1] http://www.openstreetmap.org

[2] http://wiki.openstreetmap.org/wiki/Software

paid for querying. Urban areas are considerably more contributed by users, existing a wide coverage of towns and cities in OSM. Querying urban maps can be seen from many points of view. One of the most popular querying mechanism is the so-called *routing* or *navigation*, which gives the most suitable route to go from one point to another of the city. In this case, the input of the query are two points (or streets) and the output is a sequence of instructions to be followed in order to reach the destination. Nevertheless, querying an urban map can also be interesting for city sightseeing. In fact, the places of interests around a given geo-localized point are the major goal. In this case, the input of the query is a point and a city area, close to the point, and the output is a set of points. The tourist would also like to query streets and buildings close to a given street when he/she is looking for a hotel, or to query parking areas, restaurants, high ways, etc. In such queries, the input is a given point (or street) and the output is a number of streets, parking areas, restaurants, high ways, etc. Additionally, in a city sightseeing, keyword based searching is useful. Let us suppose that the tourist is looking for leisure and shopping places, restaurants areas, as well as a pharmacy. In this case, the input of the query is a keyword, and the output is a set of points, streets, areas, etc.

XQuery [20, 2] is a programming language proposed by the W3C as standard for the handing of XML documents. It is a functional language in which *for-let-orderby-where-return (FLOWR)* expressions are able to traverse XML documents. It can express Boolean conditions and provides format to output documents. XQuery has a sublanguage, called *XPath* [6], whose role is to address nodes on the XML tree. XPath is properly a query language equipped with Boolean conditions and many path-based operators. XQuery adds expressivity to XPath by providing mechanisms to join several XML documents.

Recently [1], we have presented a library for querying OSM with the XML query language XQuery. This library is based on the well-known spatial operators defined by Clementini and Egenhofer [7, 9], providing a repertoire of XQuery functions which encapsulates the search on the XML document representing a layer of OSM. In essence, the library provides a repertoire of *OSM operators*, for nodes and ways which, in combination with *higher order* facilities of XQuery, makes easy the definition and composition of *spatial* and *keyword search* based queries. OSM data are indexed by (1) an *R-tree* structure [15] for spatial data, where nodes and ways are enclosed by *Minimum Bounding Rectangles (MBRs)*, as well as by (2) an *XML indexing* structure for textual data. Indexing enables shorter answer time.

In this paper we extend the library with aggregation operators. We have incorporated a rich repertoire of aggregation operators which, in combination with the previously proposed library, makes possible to formulate aggregation based queries easily. Data summarization and ranking is crucial in database systems and query languages have to be equipped with aggregation operators. In the special case of spatial databases, aggregation is required to summarize both spatial and non-spatial data. Several proposals of aggregation operators for spatial data have been proposed in the literature (see [21, 8] among others).

They have been studied in the context of *Spatial Data Warehouses* and the *OLAP* model. The term *SOLAP* was coined in this framework, and extensions to the well-known OLAP model have been proposed. Additionally, R-tree based structures have been proposed to deal with spatial indexing and aggregation (see [19] for an example). Our proposal, even when cannot be properly considered a SOLAP approach, is inspired by this framework.

Basically, the library makes possible to compute the *maximum* and *minimum*, the *mode*, etc., of non-spatial data associated to OSM elements occurring in an OSM layer, and also, to retrieve the OSM elements to which these values correspond to. The library enables the retrieval of objects of a given OSM layer according to a certain *ranking*. The ranking can be defined for measures like distance, area, perimeter, etc., but other measures can also be used. The ranking operators have a parameter indicating the measure to be ranked. For instance, we can get the hotels with the maximum number of stars, the restaurants with the most typical cuisine, etc. Additionally, the library provides functions like *sum*, *count*, *average*, etc., enabling data *summarization*. Moreover, XQuery makes possible the composition of queries, that is, the result of a query can be the input of another one, and thus several rankings can be combined. Thus, the library permits to count hotels with the highest number of stars, compute distance averages of monuments to a certain city point, etc.

We have developed a Web-based tool[3], called XOSM, for querying and visualization of OSM maps. In XOSM the user can select an area of the OSM map, and after an indexing process, to query the area using the XQuery library. Results are highlighted in the map. The XOSM tool is based on a client-server architecture, in which the Web page sends requests to the API Rest server of *BaseX* XQuery interpreter [14]. The XOSM is equipped with three main modules. (1) *Indexing*: an XML-based R-tree is generated from the select area of the map; (2) *Query Engine*: Parsing and execution of queries making use of the XQuery library; (3) *Result Layout*: Visualization of results with *LeafLet* and *jQuery*.

The rest of this article is organized as follows. Section 2 will present the basic elements of OSM, it will summarize the OSM indexing process and describe the main OSM operators. Section 3 will define the OSM aggregation operators and Section 4 will present the XOSM tool, showing examples and benchmarks for several datasets. Section 5 will compare with related work and finally, Section 6 will conclude and present future work.

2 Open Street Map Querying

2.1 Open Street Map Basic Elements

OpenStreetMap uses a topological data structure which includes the following core elements: (1) *Nodes* which are points with a geographic position, stored as coordinates (pairs of a latitude and a longitude) according to WGS84. They are used in ways, and allow to describe map features without a size, like points of

[3] http://indalog.ual.es/XOSM

interest and mountain peaks. (2) *Ways* are ordered lists of nodes, representing a poly-line, or possibly a polygon if they form a closed loop. They are used to represent streets, rivers, among others, as well as areas; for instance, forests, parks, parkings and lakes. (3) *Relations* are ordered lists nodes, ways and relations. Relations are used for representing the relationship of existing nodes and ways. (4) *Tags* are key-value pairs (both arbitrary strings). They are used to store *metadata* about the map objects (such as their type, name and physical features). Tags are attached to a node, a way, a relation, or to a member of a relation. For instance, the street *"Calzada de Castro"* of the Almería city (Spain) is represented by a way as follows:

```
<way id='-3731'>
    <nd ref='-3625' />
    <nd ref='-3623' />
    <nd ref='-3621' />
    <tag k='highway' v='residential' />
    <tag k='name' v='Calle Calzada de Castro' />
    <tag k='oneway' v='yes' />
</way>
```

wherein the representation includes the set of node identifiers as well as the tags for expressing that *"Calzada de Castro"* is a residential oneway. In spite of the simplicity of the XML representation of OSM, many features[4] in a OSM layer can be described.

2.2 Open Street Map Indexing Process and Functions

In order to handle large city maps, in which the layer can include many objects, an R-tree structure to index spatial objects has been implemented. The R-tree structure is based as usual on MBRs to hierarchically organize the content of an OSM map, and they are also used to enclose OSM nodes and ways in leaves. The R-tree structure has been implemented as an XML document as follows:

```
<node x="-2.4574724" y="36.8305714" z="-2.4473768" t="36.849285">
<node x="-2.4565026" y="36.8319462" z="-2.4476476" t="36.849285">
<node x="-2.4557511" y="36.8319462" z="-2.4491401" t="36.8414807">
<leaf x="-2.4557511" y="36.8347249" z="-2.4522051" t="36.8396123">
<mbr x="-2.4533564" y="36.8383646" z="-2.452359" t="36.8384662">
<way id="-11215" visible="true">
....
</way>
<node id="-10263" visible="true" lat="36.8384662" lon="-2.452359"/>
<node id="-10833" visible="true" lat="36.8383646" lon="-2.4533564"/>
</mbr>
...
</node>
```

The tag based structure of XML is used to represent the R-tree with two main tags called *node* and *leaf*. A tag *node* represents an inner node, while a tag *leaf* represents the leaves. Leaves store OSM ways and nodes. In addition, the tag *mbr* is used in order to represent MBRs. The root element of the XML document is the root node of the R-tree, and the children can be (inner) nodes or leaves. x, y, z and t attributes of nodes are the left (x, y) and right (z, t) corners

[4] http://wiki.openstreetmap.org/wiki/Map_Features

Table 1. Index-based Functions

Name	*Definition*
getLayerByName(rt,n,d)	Nodes and ways of rt at distance d to an OSM element with name n
getLayerByElement(rt,e,d)	Nodes and ways of rt at distance d to an OSM element e
getElementByName(rt,n)	OSM representation in rt of an OSM element with name n
getElementsByKeyword(rt,k)	Nodes and ways of rt annotated with the keyword k

of the MBRs. MBRs are also represented by left and right corners. A function of the our library called *load_file* is used to transform a given OSM layer to an R-tree. OSM to R-tree transformation is called indexing process. The functions of the library to query OSM layers work with R-trees instead of OSM layers. In the XOSM tool, the first step consists in the selection of an OSM map area and the indexing of the area.

We have implemented in XQuery a set of *index-based functions*, shown in Table 1, to handle the R-tree of an OSM layer. From them, the function *getLayerByName* obtains, given the name of an OSM element (node or way), the nodes or ways of the OSM layer close to the given element. With this aim, a *distance* value has to be provided. Closeness means that the shortest distance between the MBRs associated to the returned elements (i.e., nodes and ways) and the MBR of the given element is smaller than the given distance value. *getLayerByElement* is similar to *getLayerByName*, but the OSM representation of a node or way is given as input instead of the name. Additionally, we provide two functions: *getElementByName* to retrieve the OSM representation of a given name, and *getElementsByKeyword* to retrieve the set of nodes and ways annotated with a keyword. The query language under the proposed library allows (1) geo-localized queries, using *getLayerByName* or *getLayerByElement* as basis, in the sense that, queries are focused on a certain area of interest; and (2) keyword-based queries, using *getElementsByKeyword* as basis, for the retrieval of a set of nodes/ways annotated with a given keyword. A particular element can be retrieved by *getElementByName*.

2.3 Open Street Map Operators

Additionally, a repertoire of OSM operators has been designed in order to express (a) spatial and (b) keyword queries over OSM layers. With respect to (a), two kinds of operators are considered: *Coordinate based OSM operators*, shown in Table 2 and *Clementini based OSM operators*, shown in Table 3. With respect to (b), the *Keyword based OSM operators* shown in Table 4 are considered.

On the other hand, XQuery 3.0 is equipped with higher order facilities. It makes possible to define functions in which arguments can also be functions. In

Table 2. (Spatial) Coordinate Based OSM Operators

Name	*Definition*	*Spatial Operation*
isIn(s1,s2), *isNext(s1,s2)* and *isAway(s1,s2)*	true whenever the shortest distance between $s1$ and $s2$ is smaller (in central angel degrees) than 0.0001, 0.001 and 0.01, respectively	Distance
furtherNorthNodes(p_1,p_2) and *furtherNorthWays*(s_1,s_2)	true whenever p_1 (resp. s_1) is further north than p_2 (resp. s_2)	Using latitudes of in north and south hemispheres
furtherSouthNodes(p_1,p_2) and *furtherSouthWays*(s_1,s_2)	true whenever p_1 (resp. s_1) is further south than p_2 (resp. s_2)	furtherNorthNodes and furtherNorthWays negation
furtherEastNodes(p_1,p_2) and *furtherEastWays*(s_1,s_2)	true whenever p_1 (resp. s_1) is further east than p_2 (resp. s_2)	Using latitudes of in west and east hemispheres
furtherWestNodes(p_1,p_2) and *furtherWestWays*(s_1,s_2)	true whenever p_1 (resp. s_1) is further west than p_2 (resp. s_2)	furtherEastNodes and furtherEastWays negation

Table 3. (Spatial) Clementini Based OSM Operators

Name	*Definition*	*Clementini's Operator* [7]
inWay(p,s)	true whenever p (point) is in s (street)	Contains
inSameWay(p_1,p_2,m)	true whenever p_1 (point) and p_2 (point) are in the same street of the OSM map m	inWay and Equals
intersectionPoint(s_1,s_2)	the intersection point of s_1 (street) and s_2 (street)	Intersection
isCrossing(s_1,s_2)	true whenever s_1 (street) crosses s_2 (street)	Crosses
isNotCrossing(s_1,s_2)	true whenever s_1 is not crossing s_2	Negation of isCrossing
isEndingTo(s_1,s_2)	true whenever s_1 ends to s_2	Intersection and Equals
isContinuationOf(s_1,s_2)	true whenever s_2 is a continuation of s_1	Equals

Table 4. Keyword Based OSM Operators

Name	*Definition*
searchKeyword(e,kv)	true whenever the OSM element e has some k or v equal to kv
searchKeywordSet(e,($kv_1,\ldots,kv_n$))	true whenever the OSM element e has some k or v equal to some kv_i of ($kv_1,\ldots,kv_n$)
searchTag(e,k_0,v_0)	true whenever the OSM element e has some k and v equal to k_0 and v_0, respectively
getTagValue(e,k_0)	the value v of k equal to k_0 in the OSM element e

Table 5. Higher order functions of XQuery

Name	*Definition*
fn:for-each(s,f)	Applies the function f to every element of the sequence s
fn:filter(s,p)	Selects the elements of the sequence s for which p is true
fn:sort(s,f)	Sort the elements of the sequence s with respect to the value of the function f

fact, XQuery provides a library of built-in higher order functions (see Table 5). Making use of this capability, our library makes possible to combine higher order

functions with Coordinate and Clementini Based OSM Operators, and Keyword Based OSM Operators in order to express (a) Spatial and (b) Keyword Based Queries, respectively.

For instance, with respect to (a), the higher order function *filter* combined with the spatial OSM operator *isCrossing* can be used to retrieve all the streets that cross a given street (for instance, the street *"Calzada de Castro"* in Almería city, Spain) as follows:

```
let $layer := rt:getLayerByName(.,"Calle Calzada de Castro",0.001),
    $street := rt:getElementByName(.,"Calle Calzada de Castro")
return
    fn:filter($layer,xosm_sp:isCrossing(?, $street))
```

Here *getLayerByName* obtains, from the indexed OSM layer (represented by '.'), all the elements close, i.e. with MBRs at distance 110 meters (0.001 in central angel degrees), to the street *"Calzada de Castro"*[5], and *getElementByName* retrieves the OSM way representing the street *"Calzada de Castro"*. The symbol '*?*' indicates the *isCrossing* argument to be filtered.

With respect to (b) (i.e., Keyword Based Queries), the keyword operator *searchKeyword* (i.e. true whenever an OMS element has the given keyword) in combination with the higher order function *filter* can be used to retrieve, from the indexed OSM layer, all the schools close to the street *"Calzada de Castro"*. In this case, the search is restricted to vicinity of the street *"Calzada de Castro"*.

```
let $layer := rt:getLayerByName(.,"Calle Calzada de Castro",0.001)
return
    fn:filter($layer,xosm_kw:searchKeyword(?,"school"))
```

Let us remark that we have required a distance of *0.001* (i.e., 110 meters) to *"Calzada de Castro"*, which means MBRs of schools are at distance *0.001* from the MBR of *"Calzada de Castro"*. Increasing distance, farther away schools are retrieved.

3 Aggregation Operators

Now, we show the proposed operators for expressing aggregation queries (see Table 6). The aggregation operators are inspired by SOLAP operators. In [21], a taxonomy of operators has been stablished of the so-called *numeric operators* whose result is numeric[6]. They consider two levels of operators. The first level includes *numeric-spatial* and *numeric-multidimensional* operators. Numeric-spatial operators can be topological (Boolean (i.e., one-zero) Clementini's operators), and metric (*area*, *length*, *perimeter* and *distance*), while numeric-multidimensional operators are *max*, *min*, *sum*, *count* and *distinct count*, among others. A second level is defined as combinations of operators of the first level.

[5] "Calle" means street in spanish.

[6] In [21] they also consider *spatial operators* whose result is spatial (*ConvexHull*, *Envelope*, *Centroid*, *Boundary*, *Intersection*, *Union*, *Difference* and *Buffer*). They also consider navigation and temporal operators.

Table 6. Aggregation Based OMS Operators

Type	*Name*
Distributive	*topologicalCount(sq,e,b)*, *metricMin(sq,m)*, *metricMax(sq,m)*, *metricSum(sq,m)*, *minDistance(sq,e)* and *maxDistance(sq,e)*
Algebraic	*metricAvg(sq,m)*, *metricStdev(sq,m)*, *avgDistance(sq)*, *metricTopCount(sq,k,m)*, *metricBottomCount(sq,k,m)*, *topCountDistance(sq,k)* and *bottomCountDistance(sq,k)*
Holistic	*metricMedian(sq,m)*, *metricMode(sq,m)* and *metricRank(sq,m,k)*

Numeric-multidimensional operators can be classified by *distributive*, *algebraic* or *holistic* [13]. An aggregate function is *distributive* if it can be computed in a distributed manner, that is, the result derived by applying the function to the n aggregate values is the same as that derived by applying the function to the entire data set (without partitioning). An aggregate function is *algebraic* if it can be computed by an algebraic function with m arguments (where m is a bounded positive integer), each of which is obtained by applying a distributive aggregate function. Finally, an aggregate function is *holistic* when there does not exist an algebraic function with m arguments that characterizes the computation.

Let us now introduce the proposed aggregation operators. With respect to the first level, numeric spatial operators (both topological and metric), they are derived from the Boolean spatial operators defined in Section 2.3, and from the JTS library in the case of metric operators (i.e., *area*, *length*, *perimeter* and *distance*). Numeric multidimensional operators of the first level are taken from the built-in XQuery functions.

With respect to the second level, they are summarized in Table 6. *Distributive* operators are defined as follows: (1) *topologicalCount(sq,e,b)* returns the number of objects of a sequence of objects *sq* that meet a given Boolean spatial relation b with the OSM element e (i.e., *isIn*, *isNext*, *furtherNorthNodes*, *isCrossing*, and so on); (2) *metricMin(sq,m)*, resp. (3) *metrixMax(sq,m)*, returns the objects of *sq* having the minimum, resp. maximum, value of a given metric operator m; (4) *metricSum(sq,m)* returns the result of adding the values of *sq* of a given metric operator m; and, finally, (5) *minDistance(sq,e)*, resp. (6) *maxDistance(sq,e)*, returns the object of *sq* with the minimum, resp. maximum, distance with respect to the OSM element e.

In the case of *algebraic* operators, we have defined the following operators: (1) *metricAvg(sq,m)* returns the average value of a given metric operator m in *sq*; (2) *avgDistance(sq)*, the same as *metricAvg* but for *distance*; (3) *metricStdev(sq,m)* returns the standard deviation of a given metric operator m in *sq*; (4) *metricTopCount(sq,k,m)* (resp. (5) *metricBottomCount(sq,k,m)*) returns the k elements with the highest (resp. the lowest) values of a given metric operator m; and, finally, (6) *topCountDistance(sq,k)* and (7) *bottomCountDistance(sq,k)* are similar to the last ones but for *distance*.

Finally, with regard to *holistic* operators, we have considered the following ones: (1) *metricMedian(sq,m)* and (2) *metricMode(sq,m)* returning the median and mode value of a given metric operator m, respectively, and (3) *metri-*

cRank(sq,m,e) which returns the position of an element e in the ranking of the metric operator m.

We would like to remark that our proposal of aggregation operators is richer than the proposed in [21]. In all the cases of metric operators, functions passed as arguments can be one of *area*, *perimeter*, *length* and *distance*, as well as any function which computes numeric values from OSM elements. This is the case of *metricMin*, *metricMax*, *metricSum*, *metricAvg*, *metricStdev*, *metricTopCount*, *metricBottomCount*, *metricMedian*, *metricMode* and *metricRank*. Additionally, operators *metricMin* and *metricMax* which return an unique value in [21], can here return more than one value. There *metricMin* and *metricMax* are used for area, perimeter and length which rarely are equal for more than one value. Here, *metricMin* and *metricMax* can be applied to any operator returning a numeric value, for instance, number of stars of hotels, which can be the same for several hotels. Finally, *metricMode* can be applied to any operator, not only numeric.

Now, let us see the implementation of the operators in our framework. Let us start with the distributive operators. For instance, *topologicalCount* is defined as follows:

```
declare function xosm_ag:topologicalCount($sq as node()*,$e as node(),$b as
    xs:string)
{
 count(fn:filter($sq,function($o){xosm_sp:booleanQuery($o, $e, $b)}))
};
```

Due to the XQuery higher order facilities, *topologicalCount*, in our approach, has an input parameter (i.e., *$b*) representing the name of the spatial operator. *topologicalCount* is defined in terms of *count* (built-in XQuery function), *filter* (higher-order function) and *booleanQuery*. *booleanQuery* is a function of our library for checking a given boolean relation (for instance, topological) between two OSM elements. *metricMax* is also defined using the higher order facility as follows:

```
declare function xosm_ag:metricMax($sq as node()*, $m as xs:string)
{
 let $l := xosm_ag:metricList($sq,$m),
     $max := fn:max(data($l/tag[@k=$m]/@v))
 return
     fn:filter($l, function($o){xosm_kw:searchTag($o,$m,$max)})
};
```

metricMax uses the built-in XQuery function *max*, *filter*, the keyword operator *searchTag* (i.e. true whenever the OSM element has a tag with the given values), and an auxiliary function *metricList* of our library. *metricList* annotates the computed values from the metric operator (represented by the input parameter *$m*, i.e. area, perimeter, etc) in each OSM element. These values are added as a tag in each element.

With regard to algebraic operators, *metricBottomCount* is defined as follows:

```
declare function xosm_ag:metricBottomCount($sq as node()*,$m as xs:string,$k
    as xs:integer)
{
  let $l := xosm_ag:metricList($sq,$m)
  return
  fn:subsequence(fn:sort($l,function($o){xosm_kw:getTagValue($o,$m)}),1,$k)
};
```

which is defined in terms of *metricList*, the higher order function *sort*, the built-in XQuery function *subsequence* and the keyword operator *getTagValue*. *topCountDistance* is similarly defined as follows:

```
declare function xosm_ag:topCountDistance($sq as node()*, $k as xs:integer)
{
 fn:subsequence(fn:sort($sq,
        function($o){-(xosm_kw:getTagValue($o,"distance"))}),1,$k)
};
```

topCountDistance can be easily implemented because distances have been annotated in the R-tree to each OSM element. Finally, we have also implemented the holistic ones; for instance *metricMedian* is defined as follows:

```
declare function xosm_ag:metricMedian($sq as node()*, $m as xs:string)
{
  let $l := xosm_ag:metricList($sq,$m),
       $ol := fn:sort($l/*,function($o){xosm_kw:getTagValue($o,$m)}),
       $c := count($ol)
  return
    if ($c mod 2 != 0) then $ol[xs:integer($c div 2)+1]
    else
    ($ol[$c div 2]/tag[@k=$m]/@v + $ol[($c div 2) + 1]/tag[@k=$m]/@v) div 2
};
```

metricMedian is defined in terms of *metricList*, *sort* and *count*. Finally, *metricRank* is defined in terms if *metricList*, *sort* and the keyword operator *getTagValue*, as follows:

```
declare function xosm_ag:metricRank($sq as node()*,$m as xs:string, $k as xs:
    integer)
{
  let $l := xosm_ag:metricList($sq,$m)
  return fn:sort($l,function($o){xosm_kw:getTagValue($o,$m)})[$k]
};
```

4 XOSM Tool

In this section, we will show the XOSM (XQuery for Open Street Map) tool, developed by our group. The Web-based tool facilitates map querying, enabling the selection of an area of the OSM map and the creation of an index (see Figure 1). Once the index has been created, queries can be executed from the XQuery shell (see Figure 2). Additionally, the tool is equipped with a batch of pre-defined spatial, keyword and aggregation queries (among them, the included in this paper). XOSM visualizes the answer highlighting ways in a different color and with an icon in the case of nodes (see Figure 2). In case the result is a value (integer, real, etc.,) XOSM visualizes the result in a pop-up window.

4.1 Examples of Queries

Now, we show some examples of queries. Figures 3 and 4 show the results of the queries in the XOSM tool.

Example 1. The first query requests the size of park areas close to the "Paseo de Almería" street. The query is expressed as follows:

Fig. 1. Select Area, Index Name and Create Index

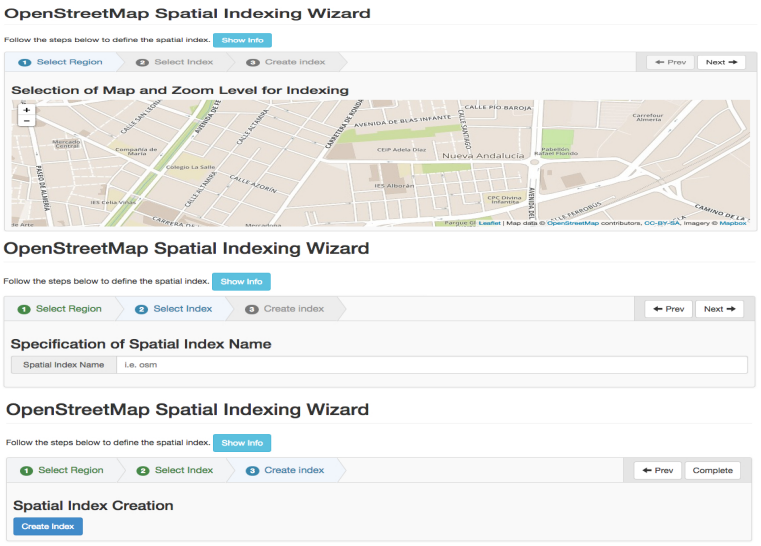
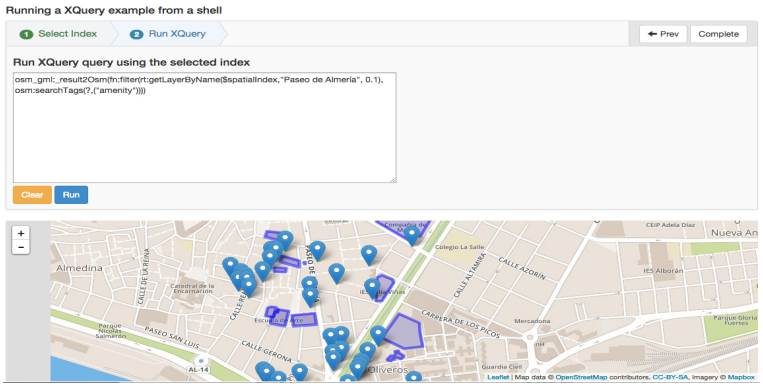


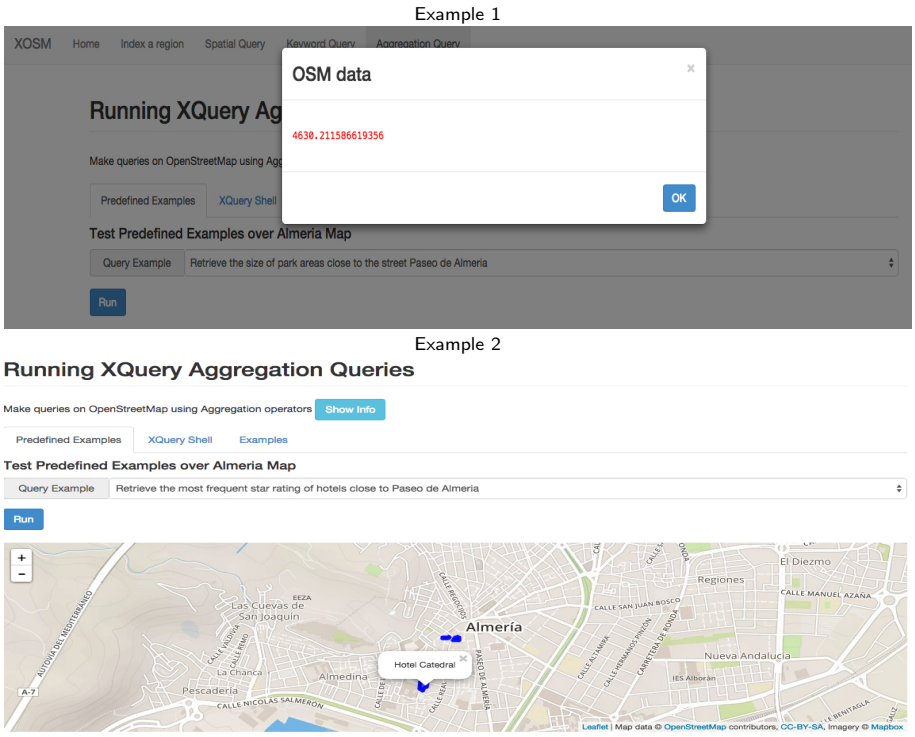
Fig. 2. Execution of Queries




```
let $layer := rt:getLayerByName(.,"Paseo de Almeria",0.003),
    $parkAreas := fn:filter($layer,xosm_kw:searchKeyword(?,"park"))
return
    xosm_ag:metricSum($parkAreas,"area")
```

Here, *getLayerByName* allows us to retrieve all the OSM elements at a distance of 330 meters from "Paseo de Almería". Next, *filter* and *searchKeyword* select those OSM elements labeled with the tag "park", and finally, *metricSum*

Fig. 3. Results of Examples 1 and 2 in XOSM



is used to sum sizes of park areas.

Example 2. The second example requests the hotels with most frequent star rating close to "Paseo de Almería". The query is expressed as follows:

```
let $layer := rt:getLayerByName(.,"Paseo de Almeria",0.003),
    $hotels := fn:filter($layer,xosm_kw:searchKeyword(?,"hotel"))
return
    xosm_ag:metricMode($hotels,"stars")
```

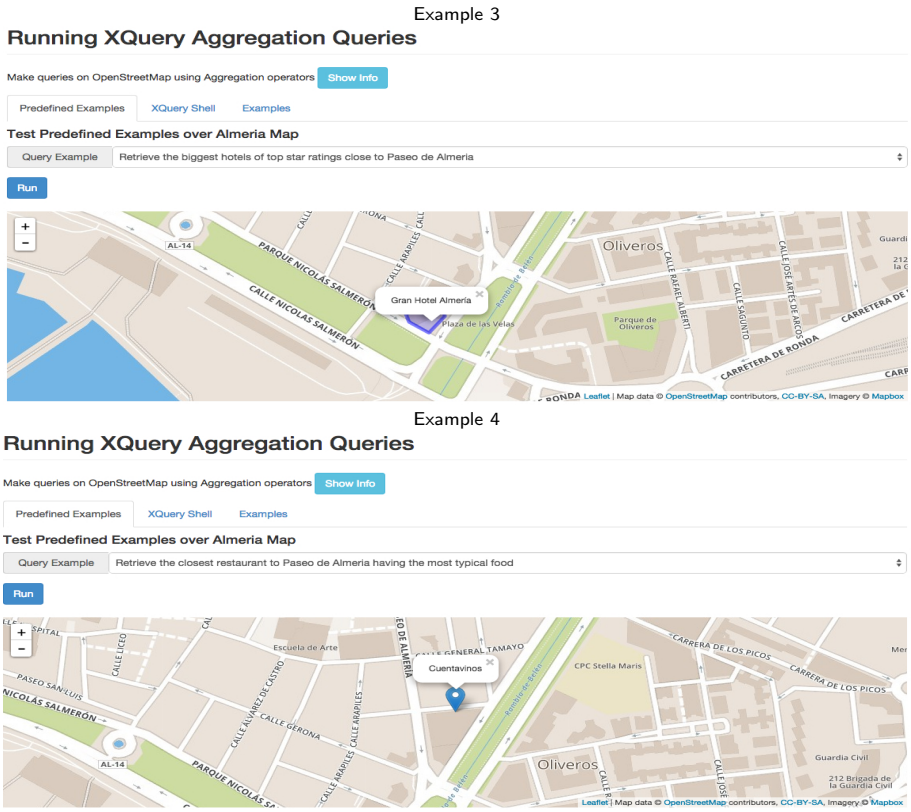
In this query the searched tag with the functions *filter* and *searchKeyword* is "hotel", and the holistic operator *metricMode* is used.

Example 3. The third example requests the biggest hotels of top star ratings close to "Paseo de Almería". The query is expressed as follows:

```
let $layer := rt:getLayerByName(.,"Paseo de Almeria",0.003),
    $hotels := fn:filter($layer,xosm_kw:searchKeyword(?,"hotel"))
return
    xosm_ag:metricMax(xosm_ag:metricMax($hotels,"stars"), "area")
```

In this case the functions *filter* and *searchKeyword* are used to search the tag "hotel", and after the distributive function *metricMax* is used twice. Thus this

Fig. 4. Results of Examples 3 and 4 in XOSM



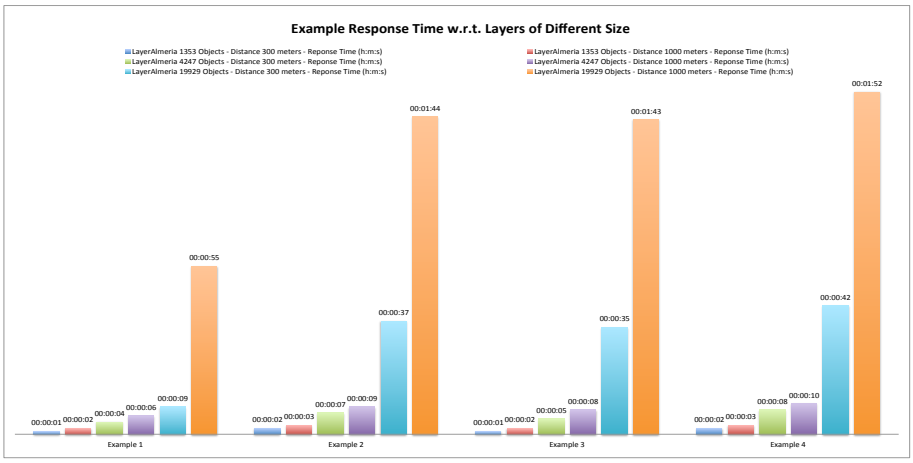
example shows how to compose queries. The first result (i.e., the hotels with the maximum number of stars) is used to compute the second result (i.e., the maximum of sizes).

Example 4. The last example requests the closest restaurant to "Paseo de Almería" having the most typical food. The query can be expressed as follows:

```
let $layer := rt:getLayerByName(.,"Paseo de Almeria",0.003),
    $restaurants := fn:filter($layer,xosm_kw:searchKeyword(?,"restaurant"))
return
    xosm_ag:metricMin(xosm_ag:metricMode($restaurants,"cuisine"),"distance")
```

Again, *filter* is combined with *searchKeyword* for the keyword "restaurant". Now, the distributive operator *metricMin* is composed with the holistic one *metricMode*. *metricMin* is used for the retrieval of the closest restaurant to "Paseo de Almería", while *metricMode* is used for the retrieval of the most typical food.

Fig. 5. Time for Query Execution



4.2 Benchmarks

Now, we show the benchmarks with aggregation queries. We have taken response times for the previous examples, varying size of layers. Layers range from 1353 to 19929 objects (i.e., from 5km to 10km). We have used the *BaseX Query* processor in a HP Proliant (two processors and 16 MB RAM Memory) with Windows Server 2008 R2. The results are shown in Figure 5. We can see that costly queries are those involving composition, but all of them exhibit a good behavior (from several seconds to two minutes). The layers used can be considered of medium-size. For getting reasonable times in big cities, is crucial to be focused on a medium-size city area. Even when the whole layer of big cities can be handled in our approach, the XOSM queries are focused on an smaller sublayer (of a certain area of the city). For this reason, we have tested the examples with the Almería layers, which have a size similar to big areas of big cities. XOSM limits the size of OSM layers to be indexed and queried.

5 Related Work

Most tools are able to query OSM with very simple commands: searching by tag and relation names. This is the case of *JOSM*[7] and *Xapiviewer*[8]. The *OSM Extended API (or XAPI)*[9] is an extended API that offers search queries in OSM with a XPath flavoring. The *Overpass API (or OSM3S)*[10] is an extension to se-

[7] https://josm.openstreetmap.de/
[8] http://osm.dumoulin63.net/xapiviewer/
[9] http://wiki.openstreetmap.org/wiki/Xapi
[10] http://overpass-api.de/

lect parts of the OSM layer. Both XAPI and OSM3S act as a database over the web: the client sends a query to the API and gets back the dataset that corresponds to the query. OSM3S has a proper query language which can be encoded by an XML template. OSM3S offers more sophisticated queries than XAPI, but it is equipped with a rather limited query language. OSM3S is specifically designed for search criteria like location, types of objects, tag values, proximity or combinations of them. OSM3S has the query languages *Overpass XML* and *Overpass QL*. Both languages are equivalent. They handle OSM objects ((a) standalone queries) and set of OSM objects ((b) query composition and filtering). With respect to (a), the query language permits to express queries for searching a particular object, and it is equipped with forward or backward recursion to retrieve links from an object (for instance, it enables to retrieve the nodes of a way). With respect to (b), the query language permits to express queries using several search criteria. Among others, it can express: to find all data in a bounding box (i.e., positioning), to find all data near something else (i.e., proximity), to find all data by tag value (exact value, non-exact value and regular expressions), negation, union, difference, intersection, and filtering, with a rich set of selectors, and by polygon, by area pivot, and so on. However, OSM3S facilities (i.e., query composition and filtering) cannot be combined with spatial operators such as Clementini's crosses or touches. In OSM3S, only one type of spatial intersection is considered (proximity 0 by using across selector). For instance, the query (allowed in our library) *"Retrieve the streets crossing Calzada de Castro street and ending to Avenida de Montserrat street"* is not allowed in OSM3S. On the other hand, OSM3S has a rich query language for keyword search based queries. Aggregation operators are not covered by OSM3S, and thus our proposal can be considered richer than the existent OSM specific query languages.

PostGreSQL is a well-known RDBMS with a spatial extension called PostGIS. PostGIS adds datatypes and spatial operators to PostGreSQL. Indexing of spatial data is carried out by the R-Tree-over-GiST scheme. Open Street Map can be handled by PostGIS with the following tools: (1) osmosis[11]: a Java-based library for OSM loading, writing and ordering; (2) Osm2pgsql[12] on top of osmosis, to transform OSM; (3) Imposm[13] a Python-based tool to import OSM in XML and PBF ("Protocolbuffer Binary Format") formats. We have implemented our own R-tree structure to store XML-based OSM data. However, we plan to adopt PostGIS indexing in the future versions of the XOSM tool.

In the context of RDF and SPARQL, there are several proposals of languages and tools for working with spatial data. GeoSPARQL[3] (standard of the Open Geospatial Consortium) and stSPARQL[17] are the most relevant contributions to this area. Both are very similar. GeoSPARQL provides a vocabulary to express spatial data in RDF, and defines an extension of SPARQL for querying. The vocabulary includes a set of topological operations based on Egenhofer

[11] https://github.com/openstreetmap/osmosis
[12] https://github.com/openstreetmap/osm2pgsql
[13] http://www.imposm.org/

and Region Connection Calculus (RCC8). Parliament[4] is an implementation of the GeoSPARQL using Jena as RDF system. uSeekM also uses the Sesame RDF store as well as PostGIS, and implements GeoSPARQL features. Omitting aggregate functions and updates from stSPARQL, stSPARQL is a subset of GeoSPARQL. stSPARQL is supported by Strabon[18], which extends the RDF store Sesame with spatial data stored in PostGIS. stSPARQL uses SELECT, FILTER and HAVING clauses of SPARQL in combination with spatial predicates to query RDF spatial data. The "OpenGIS Simple Feature Access" standard (OGC-SFA)[14], Egenhofer and RCC-8, are used as languages for topological spatial relations. FILTER can be combined with them to define spatial selections and joins. These can be also used in the SELECT and HAVING clauses. Aggregation is present in stSPARQL in the form of union, intersection and extent (i.e., minimum bounding box of a set of geometries). stSPARQL uses a B-tree to index non spatial data, while R-Tree-over-GiST is used is used for spatial indexing (provided by PostGIS). stSPARQL is mapped to SQL queries executed under PostGreSQL. Virtuoso, OWLIM and AllegroGraph are RDF-based engine supporting geometries of points. Open Street Map has been integrated in the RDF area thanks to the OSM Semantic Network, in which OSM data resources are available in RDF format.

While the context is different (RDF/SPARQL versus XML/XQuery) we found analogies with the existent approaches. With regard to topological operators, our library is built on top of JTS, and thus providing similar expressivity. Indexing is based in all the cases on R-tree like structures. With regard to expressivity of the query language, RDF/SPARQL based languages can be considered similar to our proposal, except for aggregation operators. The only case of language equipped with aggregation is stSPARQL, including union, intersection and extent. Thus our library provides a richer set of aggregation operators.

Unfortunately, we cannot compare our benchmarks with existent implementations of similar tools due to the following reasons. Even when OSM has been used for providing benchmarks in a recent work [10], they use OSM as dataset for Description Logic based reasoners rather than to evaluate spatial queries. There are some proposals [16, 11] for Spatial RDF benchmarking, but none of the queries involve the same kind of aggregation we propose. Benchmarks of RDF/SPARQL tools are generally concerned with execution times of large RDF resources, that most of cases are built from several RDF namespaces.

Our goal is less ambitious, focused on the development of a tool (XOSM) for map querying and visualization. XOSM limits the size of maps to be indexed and queried. This is also the case of other OSM APIs. In any case, XOSM has to answer in short time, and the efficiency of XOSM has been one of our main concerns during the development of our work.

[14] http://www.opengeospatial.org/standards/sfs

6 Conclusion and Future Work

In this paper, we have presented an extension of a previously defined library for querying Open Street Map, and a Web-based tool for the visualization and querying of OSM maps. The extension incorporates aggregation operators which makes possible to rank and summarize data from OSM maps. We have shown how to work with this kind of queries using the developed XOSM tool. As future work, we would like to extend our work as follows. Firstly, we would like to use an R-tree based structure to improve performance of aggregation operators. The idea would be to annotate each MBR of the R-tree with a pre-defined set of aggregation values (maximum, minimum, mode, etc.,) of each area. It would make possible to implement more efficient algorithms which discard those MBRs of upper (or lower) values. Secondly, and related to the first goal, top-k algorithms will be implemented making use of the new R-tree structure.

References

1. Jesús M. Almendros-Jiménez and Antonio Becerra-Terón. Querying Open Street Map with XQuery. In *Proceedings of the 1st International Conference on Geographical Information Systems Theory, Applications and Management*, pages 61–71, 2015.
2. Roger Bamford, Vinayak Borkar, Matthias Brantner, Peter M Fischer, Daniela Florescu, David Graf, Donald Kossmann, Tim Kraska, Dan Muresan, Sorin Nasoi, et al. XQuery reloaded. *Proceedings of the VLDB Endowment*, 2(2):1342–1353, 2009.
3. Robert Battle and Dave Kolas. Geosparql: enabling a geospatial semantic web. *Semantic Web Journal*, 3(4):355–370, 2011.
4. Robert Battle and Dave Kolas. Enabling the geospatial semantic web with Parliament and GeoSPARQL. *Semantic Web*, 3(4):355–370, 2012.
5. Jonathan Bennett. *OpenStreetMap – Be your own cartographer*. Packt Publishing Ltd, 2010.
6. Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérome Siméon. XML Path Language (XPath) 2.0. Technical report, 2010.
7. Eliseo Clementini and Paolino Di Felice. Spatial operators. *ACM SIGMOD Record*, 29(3):31–38, 2000.
8. Joel da Silva, Anjolina G de Oliveira, Robson N Fidalgo, Ana Carolina Salgado, and Valéria C Times. Modelling and querying geographical data warehouses. *Information Systems*, 35(5):592–614, 2010.
9. Max J. Egenhofer. Spatial SQL: A Query and Presentation Language. *IEEE Trans. Knowl. Data Eng.*, 6(1):86–95, 1994.
10. Thomas Eiter, Patrik Schneider, Mantas Šimkus, and Guohui Xiao. Using OpenStreetMap Data to Create Benchmarks for Description Logic Reasoners. In *Proceedings of the 3rd International Workshop on OWL Reasoner Evaluation (ORE 2014)*, pages 51–57. CEUR Workshop Proceedings, Vol-1207, 2014.
11. George Garbis, Kostis Kyzirakos, and Manolis Koubarakis. Geographica: A Benchmark for Geospatial RDF Stores. In *The Semantic Web–ISWC 2013*, pages 343–359. Springer, 2013.

12. Michael F Goodchild. Citizens as sensors: the world of volunteered geography. *GeoJournal*, 69(4):211–221, 2007.
13. Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.
14. Christian Grun. BaseX. The XML Database, 2014. http://basex.org.
15. Marios Hadjieleftheriou, Yannis Manolopoulos, Yannis Theodoridis, and Vassilis J Tsotras. R-Trees–A Dynamic Index Structure for Spatial Searching. In *Encyclopedia of GIS*, pages 993–1002. Springer, 2008.
16. Dave Kolas. A Benchmark for Spatial Semantic Web Systems. In *International Workshop on Scalable Semantic Web Knowledge Base Systems*, 2008.
17. Manolis Koubarakis and Kostis Kyzirakos. Modeling and querying metadata in the semantic sensor web: The model stRDF and the query language stSPARQL. In *The semantic web: research and applications*, pages 425–439. Springer, 2010.
18. Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Strabon: a semantic geospatial DBMS. In *The Semantic Web–ISWC 2012*, pages 295–311. Springer, 2012.
19. Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. Efficient OLAP operations in spatial data warehouses. In *Advances in spatial and temporal databases*, pages 443–459. Springer, 2001.
20. Jonathan Robie, Don Chamberlin, Michael Dyck, and John Snelson. XQuery 3.0: An XML query language. *W3C Proposed Recommendation*, 2014.
21. Carla V Ruiz and Valéria Cesário Times. A Taxonomy of SOLAP Operators. In *SBBD*, pages 151–165, 2009.