Combining Symbolic and Spatial Exploratory Search – the Homerun Explorer

Jörg Roth

Department of Computer Science
Nuremberg Institute of Technology
Kesslerplatz 12
90489 Nuremberg, Germany
Joerg.Roth@th-nuernberg.de

Abstract: The Homerun Explorer provides an exploratory search service on many million records of geo data. It supports symbolic and geometric conditions as well as spatial joins. A user can deal with large result sets and iteratively refine a search. Predefined indexes allow the quick execution even on desktop computes without the need of powerful servers.

Keywords: Geo data, spatial search, textual search, geographic maps

1 Introduction

Large geo data pools such as *Open Street Map* (*OSM*) [2] have some similarities to World Wide Web resources: from the many million or even billion entries, a user often is only interested in a small subset. Current approaches for textual searches as used in Web search engines have limitation, if queries contain spatial conditions: pure text queries can only retrieve documents that explicitly mention this location as a text, e.g. as a city name. Text search engines fail for more complex spatial queries such as '*not nearby position xy*', as these queries need to evaluate *geometric* conditions. If queries spatially relate objects to other objects that are only implicitly defined by their type (e.g. '*all farms that are not near to a waste deposit*'), we talk about *spatial joins* [13]. Such queries are virtually impossible for purely text-based searches.

In this paper we introduce the search platform *Homerun Explorer*:

- It supports combined symbolic and spatial search as well as spatial joins.
- It supports the user to enter complex combined queries.
- The system supports *exploratory* search.

Exploratory search means, the user does not exactly plan a query, but just begins to enter the knowledge about the desired geo objects. After she or he received a search feed-back, the query can immediately be refined. The user repeats, until the results meet the expectations. Exploratory search has high demands on the search system: it

should accept inaccurate input (e.g. misspelled text) and the response time must be very low (not more than a few seconds).

2 Related Work

The tools in the first place that deal with geo information are *GIS* (*Geo Information Systems*) [14]. They use so-called *map overlays* to combine different object attributes to a new representation. An overlay formulates conditions that can both consider thematic data (such as type or object properties) and object geometries. They can be combined by operations such as union, intersection, or difference. GIS searches have important differences to our intended search:

- In GIS, the search process is not exploratory but systematic, i.e. the map overlay steps are properly planned a priori. They are based on a certain and well-defined data analysis task.
- In GIS, the output of a search often is a large set of objects. Analysis results are thus object counts, averages or spatial distributions or statistic results. In contrast, our intended search tries to recover certain objects from a huge number of geo objects.

Searches that focus on texts are widely known in the area of the World Wide Web [1]. Similar mechanisms can be used locally, e.g. to search documents in file systems or document repositories. Special add-ons such as *Lucene* [3] allow a developer to search texts in traditional relational databases. Even though text search engines nowadays have complex mechanisms, especially for indexing, they mainly look up texts in documents without to know the actual meaning. This especially is a problem, if we have to deal with ambiguity (see section 3.1). Moreover, text search engines can only deal with locations, if they explicitly appear as text. E.g., we can only find texts that are related to the city Nuremberg, if the word 'Nuremberg' appears in a document, and not only a name of a quarter of Nuremberg. Complex geometric constraints such as '*nearer than 500 m to a certain position*' or '*not nearer than 20 km to the next nuclear power plant*' are not searchable at all, unless these texts incidentally appear word by word in a document.

Pure map platforms such as *Google Maps* [15], in contrast, basically rely on map display. Usually, they provide a text search facility that places result markers on the map, but:

- They do not provide a real combination of spatial and textual searches.
- They mainly provide a search of *Points of Interests* (*POIs*), i.e. point-like objects.
- They usually do not provide capabilities for spatial joining (see section 3.3).
- The search engines run on powerful server environments. It is not possible to execute queries on a local database.

Even though such tools have certain drawbacks, they heavily inspired our work.

3 The Homerun Explorer

The Homerun Explorer is one component of the Homerun environment [4]. Homerun provides a platform for low-cost developments in the area of location-based services, especially of small services outside the mass market. It imports data from Open Street Map into an own database structure, provides map rendering with the *dorenda* environment and street navigation with *donavio* [6, 7, 8, 9, 10]. Client platforms support desktop computers as well as mobile devices [11].

The development of the Homerun Explorer was driven by a certain demand: after importing large data sets (e.g. 21 million records of 'Germany' data), it was very difficult to find certain objects, e.g. a special shop in a city. Until now, it was only possible to query an object by its exactly known position or its exact name. Fuzzily defined queries such as '*near… that has a name like…*' were not possible.

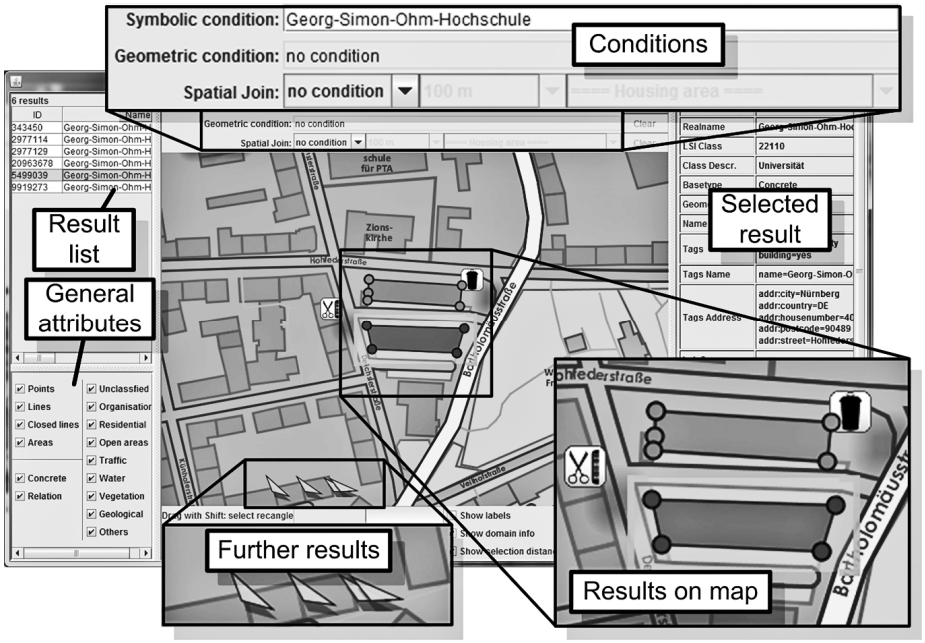

Fig. 1. The Homerun Explorer main window

Fig. 1 shows the main window. Symbolic conditions are entered in the upper area. General attributes (lower left corner) can restrict results to object types such as '*only traffic objects with line geometries*'. Results are presented as list (left) and on a zoomable map (center). If results appear outside the current view, they are represented as small arrows on the map border.

Fig. 2 shows the basic data flow for query execution. All parts of a query are passed to the *Symbolic* and *Spatial Search Engine*. Each engine can restrict the other engine (see section 3.4) to avoid large intermediate results. The two results are intersected, i.e. all query components are combined by logical *AND*.

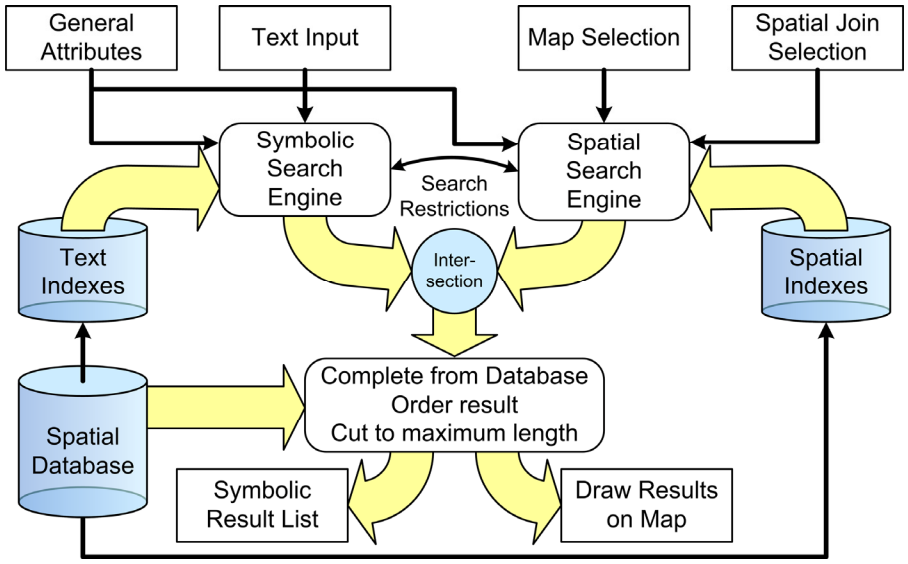

Fig. 2. The basic data flow

The search indexes remain in the application's working memory. To keep them as small as possible, they only store search-related data. As the final result should display *all* object properties, additional data has to be loaded from the persistent database. Results are ordered by their relevance and cut to a maximum length.

3.1 Symbolic Search

The Homerun Explorer accepts texts that specify name, type or locality of geo objects or any combination of these. Our symbolic search system fulfills the following goals:

- A user is not requested to learn a special structured input. She or he can type an arbitrary sequence of words in a text line.
- Whenever the user pauses text typing for 1000 ms, the next query is executed in the background. The user always gets a feedback to the current query. As a consequence, query processing has to provide high responsiveness and our search must be able to look up incomplete prefixes.
- The user does not necessarily have to know the correct spelling of, e.g. names. Thus, we have to provide fuzzy search. We have different degrees of text matches, ranging from exact spelling, some misspellings to equality of the *double metaphone* [12] that only considers the basic pronunciation.

In contrast to Web searches, our search texts formulate queries to geo objects. Thus, texts both cover object descriptions (e.g. names) as well as texts that describe their locality. E.g. in the search text

'*China Restaurant Wing Tan Nuremberg Munich Avenue*'

we have the following structure:

- '*China Restaurant*' describes the *class* (i.e. type) of the object;
- '*Wing Tan*' describes the *name* of the object;
- both '*Nuremberg*' and '*Munich Avenue*' describe the *locality*.

The meaning of each word (name, class or locality) is discovered by our search engine. The user may change the word ordering. The result remains same, as long as words that belong to the same meaning still appear in the same order, e.g.

'*Munich Avenue Nuremberg Wing Tan China Restaurant*'

is similar to the search above, but

'*Nuremberg Avenue Munich Wing Tan Restaurant China*'

may look for any restaurant (not necessarily a China restaurant) in different places (also in China). We assume a maximum of four different parts of textual queries, we call *query elements*:

- The *name* of a geo object.
- The *classification* of an object (e.g. *restaurant*, *road*, *shop*, *tree*, *lake*). As a class can be described with different words, we use a synonym list.
- A description of the *locality*. We can identify the following localities: *countries*, *states*, *regions*, *cities*, *districts* and *streets*.
- A *second* description of *locality*. With the two locality entries we can specify e.g. street or district in a city, or a city in a region.

In principle, we could add more descriptions of locality, but typical searches do not contain more than two. Note that a query does not necessarily have to contain all four query elements. E.g., very often, meaningful names already completely identify an object.

For a textual query, we have to map query words to their intended meaning. A naïve approach would use a dictionary that contains the meaning of each word, e.g. *Nuremberg* → *City*, *Franconia* → *Region*. This approach does not work, as a user can enter arbitrary unknown words. In addition, many words have multiple meanings. As an example, in the query '*Nürnberger Hof*' we got the following possible interpretations:

- '*Nürnberger Hof*' is the *name* of a restaurant in Wiesbaden;
- '*Hof*' is a German object *class* name for '*a farm*';
- '*Hof*' is a city in Bavaria, i.e. a *locality*;
- '*Nürnberger*' could express the *locality* of Nuremberg.

We have to face two problems: first, how does the search engine deal with ambiguity; second: how do we communicate the different interpretations to the user. To start with the second issue: we present *all* results that are based on *any* possible interpretation to

the user, but rank it by a hit rate. The hit rate indicates the probability that a user actually intended a specific meaning. It is based on two factors:

- The covered query elements: we assign a *base rank* to each query element (highest: *name*, then *locality*, lowest: *class*). This takes into account that a name often fully identifies an object.
- A *match rank* for each query element: for this, we compute a distance between query words and actual words in the database. Exact spelling receives a high value, whereas double metaphone matches only low values.

The final hit rank is the dot product of base rank and match rank.

To deal with ambiguity, we use a brute force approach. We iterate through all possible assigns of input words to the maximum of our four query elements, whereas inside a query element the word sequence must be the same as in the query. E.g. three valid assigns of the first query above are

- name='*Nuremberg Avenue Munich Wing'*, class='*Tan Restaurant China*';
- name='*Avenue*', class='*Tan Nuremberg Munich*', locality_1='*China Restaurant Wing*';
- name='*Wing Tan*', class='*China Restaurant*', locality_1='*Munich Avenue*', locality_2='*Nuremberg*'.

Obviously, most of the combinations are not reasonable. Only those combinations that get results in the database for *all* assigned query elements are collected, thus reasonable combinations are implicitly found (e.g. the third one above). To limit the total number of combinations, we restrict the number of words per query element to 4. For the query above with 7 words, we get a total of 792 combinations. The overall maximum of combinations is 1200 (for 9 and 10 query words), thus always remains reasonably low. In addition, some combinations can be removed:

- As the two locality entries are considered as equal, different combinations with swapped $\textit{locality}_1$ and $\textit{locality}_2$ are reduced to one combination.
- If only a locality and neither name nor class are described, this query may return too many hits, thus is removed. In this case, the user usually intended to describe the object name.

For each combination, we query our data and compute the corresponding rank. If the rank is above a certain limit, the result is added.

Sometimes, the same object is retrieved by different combinations. This is because object class and locality are part of many names. E.g., the name '*Ohm University Nuremberg*' both contains the object class (*university*) and city (*Nuremberg*). Corresponding queries would receive multiple hits. Thus, whenever an object appears multiple times in the result, only the one with the highest rank is kept.

Search Preparations

In the original Open Street Map database, contributors are free to classify objects. OSM mainly provides an informal classification based on key-value pairs. They do

not have to meet certain formal requirements. The OSM classification is very elaborative and often causes ambiguity. E.g., there are at least 3 ways to classify a '*coffee bar*'. Moreover, it is based on strings that are difficult to look up in databases and do not represent a class-to-subclass relation. In [9] we present more issues of the original OSM classification. As a consequence, we store imported geo data in an own relational database with a strict classification schema. During the import from OSM data into our database, we pre-compute the following object properties:

- The object *class*: we express objects classes by an integer number that represents a classification hierarchy [6, 7]. As an important benefit: subtrees of classes can be represented by number intervals. E.g. '*all restaurants*' are represented by a certain interval $[i, j]$ whereas '*Italian restaurants*' are represented by a smaller interval $[i_2, j_2] \subset [i, j]$. This enables quick indexing mechanisms of SQL databases. The mapping of original OSM classifications to our classification numbers uses a rule-based approach with several thousands rules.
- The *name*: from the different ways to express a name in OSM we extract the most meaningful one. Note that the most meaningful name depends on the actual object class. E.g. motorways, streets and cities have different tags to express their name.
- The *is-in-relation*: For each object we detect all important surrounding objects (i.e. where an object '*is-in*'). Usually these larger objects (i.e. our localities) are *countries*, *states*, *regions*, *cities*, *districts* and *streets*. The is-in-relation is not an obligatory part of an OSM description. Thus, our import uses a time-consuming mechanism to create the is-in-relation for all objects.

For each classification we defined a list of synonyms that may occur in a query. E.g., for the class '*Italian restaurant*' we store: '*Pizzeria*', '*Italian Restaurant*', '*Restaurant*', '*Italian*', '*Pizza*', '*Pasta*'.

Looking up Texts

The pre-computation enables to quickly check classes and localities. However, searches based on texts are still difficult for SQL databases: first, pure SQL only supports exact matches of strings and substrings, second substring searches cannot use column indexes, thus mainly compare with *all* rows. To check texts during typing a query, we thus use own indexes stored in the runtime memory. Our approach works as follows:

- All searchable strings (i.e. object names, class names including synomyms and locality names) are first simplified according to typical misspellings. E.g. all long 'i' vowels in German ('ie', 'ii', 'ih', 'ieh') are unified to a single representation.
- We iterate through all leading substrings with a predefined minimum length (currently 3 letters) and store them in a prefix hashtable. For e.g., Nuremberg, we store '*Nur'*, '*Nure*', '*Nurem*', … '*Nuremberg*' as searchable keys with a reference to the Nuremberg object.
- In addition we store the double metaphone string in another hashtable.

If the user now enters the leading string '*Nur*', the prefix hashtable returns the Nuremberg object, bus also '*Nurmistraße*', '*Nurtschweg*', '*Nurbanum*', '*Nurda-Park*' etc.

For the double metaphone '*NR*' of '*Nur*' we get '*Neuwöhr*', '*Nohra*', '*Neuweier*', '*Nouar*', '*Noer*', '*Nierow*' etc. from the double metaphone hashtable.

If a name has multiple words (separated by, e.g., blank or '-'), we also store these entries for subsets of words. Thus, a user may omit some words in a query.

As hashtables work with nearly constant time without any iterative search, all these hits are returned immediately. For name, class and locality we have separate sets of hashtables. We check all query elements – a final result must appear in all involved hashtable hits.

3.2 Spatial Search

Some localities cannot be expressed as texts, or the user does not know a textual description of it. E.g. the condition '*500 m around a certain position*' can only be formulated geometrically. For such queries, a map is inevitable. A user can define rectangles, circles or arbitrary free-hand forms to define a selection.

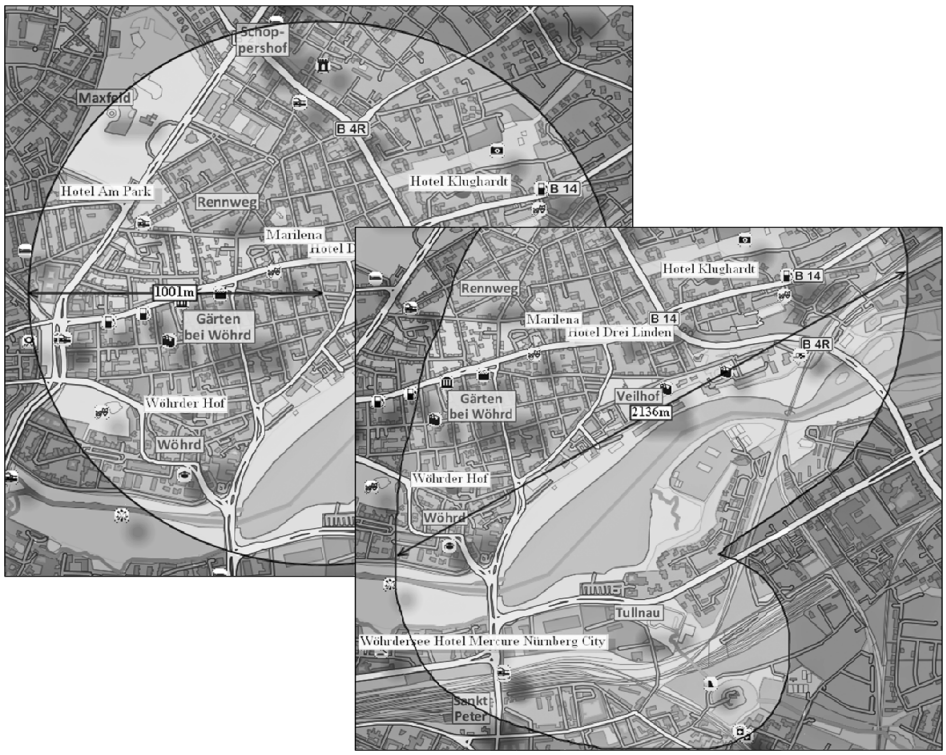

Fig. 3. User-defined spatial selections

Fig. 3 left shows a query for hotels inside a certain circle. '*Hotel*' is entered in the textual area and interpreted as class description. Fig. 3 right shows a free-hand selection for the same query. Without any further conditions a '*Hotel*' query would return *all* hotels in the database.

3.3 Spatial Joins

For explicit geometric conditions as presented above, the user already knows the locality (e.g. inside a selection circle, or given by the city name). In contrast, the *spatial join* expresses conditions for objects of unknown locations. Similar to a table join in relational databases, a spatial join relates two potentially large sets of rows to a smaller set where each pair fulfils a certain condition. Typical spatial join conditions are '*closer than*' or '*farther than*'. Thus, we can formulate queries such as '*nearer than 200 meters to any river*' or '*farther than 10 km to any nuclear power plant*' without to know its actual location.

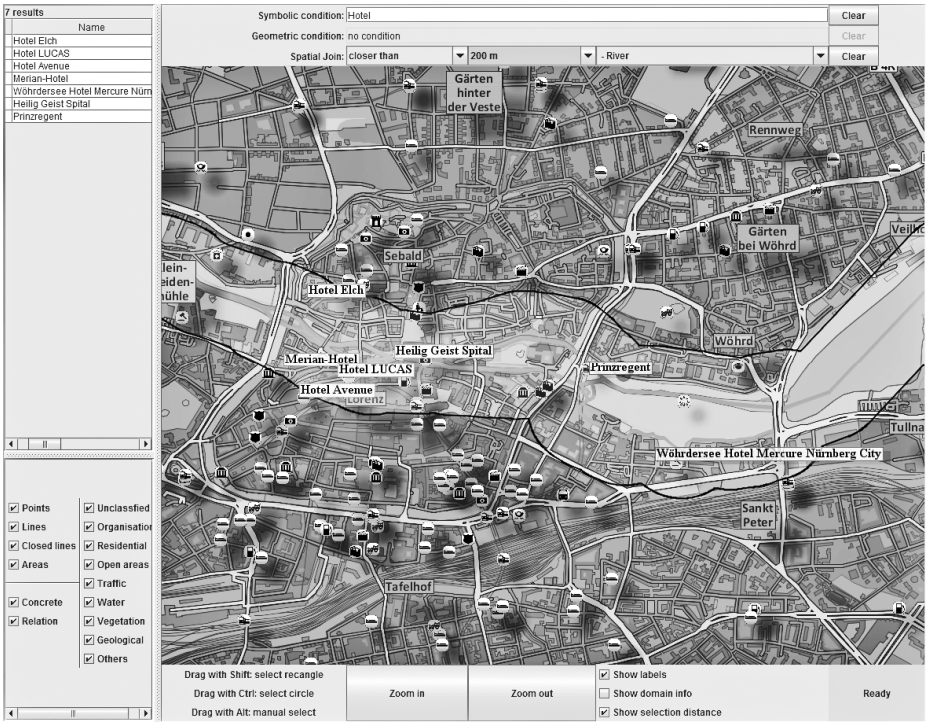

Fig. 4. A spatial join query

Fig. 4 shows an example: the user wants to look up hotels near a river (up to 200 m distance), without explicitly selecting a specific region or telling which river. The tool thus automatically computes the area of potential hits around all rivers. The text input '*Hotel*' then finds all relevant hotels. With the spatial join, we could further query,

e.g.: '*all computer game shops that are near to a computer seller*', '*all Italian restaurants near to any metro station*', '*all hotels that are not close to a highway*' or '*all nuclear power plants that are close to coast*'.

The approach to join works as follows:

- We first look up the objects that are focus of the join (e.g. *'all rivers*'). We create the sum geometry of those objects.
- The sum geometry is enlarged using the geometric *buffer* operation [5]. The buffer radius is the distance of the join request (e.g. 200 m in the query above). Note that the result of a buffer is a polygonal area, sometimes with holes, even though the input geometry only consists of points and line strings. Further note that coordinates are stored as pairs of latitude/longitude, but the distance is expressed in meters, thus we have to use the great circle formula to compute the buffer.
- If the user wants to query objects that are '*farther than*', the buffer area is inverted i.e. the result represents the area outside the buffer region.
- The resulting area is used as a geometric condition, similar to a user-selected area.

Spatial joins produce large intermediate results. E.g. the 10 km region around any river in the database is very large, thus we have to incorporate mechanisms to reduce the result set as described in the next section.

3.4 Dealing with Large Result Sets

Queries can produce large results. If e.g. the user enters a text that only specifies an object class without any locality, some thousands or even million entries would match. We have to face two problems:

- For large results, fetching the information from the database would take a long time.
- Large result lists are difficult to present to the user. First, the textual hit list could be too large to get an overview. Second, the hit markers on the map could be drawn too densely to be useful.

These effects would conflict our major goal: we encourage the user to exploratory browse the system in a trial and error manner. Thus, results must appear quickly and the user should always keep overview.

For each pattern of symbolic condition (S), geometric selection (G) and spatial join (J) we defined a strategy to limit the results as presented in table 1. The table describes how the symbolic and spatial join searches are restricted. If a geometric selection is available, the symbolic search does not have to produce million results that are finally reduced to few hits due to the final intersection. Thus, the symbolic search engine directly considers the geometric condition and only checks relevant symbolic hits.

Table 1. Restricting search results

Pattern	Meaning	Symbolic search restriction	Spatial join restriction	Result restriction
S	All objects matching the search text.	no	n/a	Ordered by symbolic hit rank. The result list is cut to a maximum length.
SG	All objects inside the user selection matching the search text.	Geom. User Selection	n/a	
SJ	All objects that match the search text and that are nearer or farther to all objects of a certain class.	no	Symb. Cond. or multiples of displayed map area	
SGJ	All objects inside the user selection matching the search text and that are nearer or farther to all objects of a certain class.	Geom. User Selection	Geom. User Selection	
G	All objects at all inside the user selection.	n/a	n/a	If result exceeds a max. length, entries are removed according to their distance to the displayed map centre.
J	All objects at all, nearer or farther to all objects of a certain class.	n/a	Multiples of displayed map area	
GJ	All objects inside the user selection nearer or farther to all objects of a certain class.	n/a	Geom. User Selection	

If a spatial join is combined with a geometric selection, it is obvious to restrict the join area to this selection. For pattern SJ the symbolic results provide good restriction, if they are all close-by. If not, or if the join is the only condition in the query (pattern J), we use multiples of the displayed map area to restrict the join: we then extend the map area in all four main directions to get a 9 times larger area. As users mainly focus on the displayed area, this approach is reasonable.

The result list of all query patterns is cut to a maximum length (currently 2000 hits). If the query contains a symbolic condition, the hits are ordered by their hit rank (section 3.1) and hits with lowest ranks beyond the maximum list length are removed. For query patterns without any symbolic condition, we order the hits by their geometric distance to the displayed map centre. Thus, only hits far from the visible centre are removed. If the user scrolls through the map, the query has to be recomputed as other hits may appear.

We integrated a further mechanism not shown in table 1: we consider the current map zoom level to remove too small and too large objects from the result set. If the map shows a large area (e.g. more than a city), small objects are not displayed as the user cannot properly identify their location. On the other hand, if the displayed area is

small (e.g. shows only some buildings), it is not reasonable to show large polygons that may cover multiples screens.

4 Conclusions and Future Work

The Homerun Explorer allows user to deal with large geo databases; it enables exploratory search of combined textual and spatial queries and supports spatial joins. It deals with fuzzily defined queries, handles large results and provides quick overview of results. The user does not have to learn a special input structure for texts. In contrast to pure text searches, our environment discovers the meaning of query strings and executes a structured search.

To the three query types (symbolic, spatial selection and spatial join) we want to add a further type in the future: *isochrones*. With isochrones we can express conditions such as: '*all positions that can be reached by a 30 minute car ride*', or '*5 min walk to any subway station*'. For this, we want to integrate the *donavio* navigation environment into the Homerun Explorer.

References

[1] Belew R. K., Finding Out About: A Cognitive Perspective on Search Engine Technology and the WWW, Cambridge University Press, 2001

[2] Bennett J., OpenStreetMap, Packt Publishing, 2010

[3] Hatcher E., Gospodnetić, O., Lucene in Action, Manning Publication, 2010

[4] The Homerun project homepage, http://www.wireless-earth.org/homerun_eng.html

[5] JTS Topology Suite homepage, http://sourceforge.net/projects/jts-topo-suite/

[6] Roth J., Modelling Geo Data for Location-based Services, 3. GI/ITG KuVS Fachgespräch 'Ortsbezogene Anwendungen und Dienste', Sept. 7-8, 2006, Berlin, Institute for Computer Science - Freie Universität Berlin, Sept. 2006, 20-25

[7] Roth J., Managing Geo Data for Location-based Services – The Hybris Framework, 4th Annual Meeting on Information Technology & Computer Science (ITCS 2008), Stuttgart (Germany), Feb. 20, 2008

[8] Roth J., The Extended Split Index to Efficiently Store and Retrieve Spatial Data With Standard Databases, IADIS International Conference Applied Computing 2009, Rome (Italy), Nov. 19-21, 2009, Vol. I, 85-92

[9] Roth J., Übernahme von Geodatenbeständen aus Open Street Map und Bereitstellung einer effizienten Zugriffsmöglichkeit für ortsbezogene Dienste, Praxis der Informationsverarbeitung und Kommunikation (PIK), Vol. 13, No. 4, 2010, 268-277

[10] Roth, J.: Der Einsatz von OpenStreetMap-Daten in der akademischen Informatik-Ausbildung, 7. GI/ITG KuVS Fachgespräch 'Ortsbezogene Anwendungen und Dienste', Logos-Verlag, 2011, 65-72

[11] Roth, J.: Moving Geo Databases to Smart Phones – An Approach for Offline Location-based Applications, Innovative Internet Computing Systems (I2CS), Berlin (Germany), June 15-17, 2011, GI Lecture Notes in Informatics, Vol. P-186, 228-238

[12] Philips L., Hanging on the Metaphone, Computer Language Magazine, Vol. 7, No. 12, 1990

[13] Shekhar S., Chawla S., Spatial Databases: A Tour, Prentice Hall. 2003

[14] Stillwell J., Clarke G. (eds.), Applied GIS and Spatial Analysis, John Wiley & Sons, 2003

[15] Svennerberg G., Beginning Google Maps API 3, Apress, 2010