AD-HOC GEOREFERENCING OF WEB-PAGES USING STREET-NAME PREFIX TREES

Andrei Tabarcea
Faculty of Automatic Control and Computer Engineering, Technical University of Iaşi, Romania
tabarcea@cs.joensuu.fi

Ville Hautamäki
*Institute for Infocomm Research, A*STAR, Singapore*
vishv@i2r.a-star.edu.sg

Pasi Fränti
Speech and Image Processing Unit, University of Eastern Finland, Joensuu, Finland
franti@cs.joensuu.fi

Keywords: Search engine, LBS, Database, Prefix tree, Georeferencing, Mobile device, Location information, Personal navigation, WWW.

Abstract: A bottleneck of constructing location-based web searches is that most web-pages do not contain any explicit geocoding such as geotags. Alternative solution can be based on ad-hoc georeferencing which relies on street addresses, but the problem is how to extract and validate the address strings from free-form text. We propose a rule-based solution that detects address-based locations using a gazetteer and street-name prefix trees created from the gazetteer. We compare this approach against a method that doesn't require a gazetteer (a heuristic method that assumes that street-name has a certain structure) and a method that also uses data structures created from the gazetteer in the form of street-name arrays. Experiments using our location based search engine prototype (MOPSI) for Finland and Singapore, show that the proposed prefix-tree solution is twice as fast and 10% more accurate than its rule-based alternative and 10 times faster if an array structure is used when accessing the gazetteer.

1 INTRODUCTION

Location-based services (LBS) have become popular during recent years due to increasingly wide availability of GPS positioning in multimedia mobile phones. For instance, according to Nokia's own estimate more than half of their phones would include GPS by 2010-2012. In case of lacking GPS, positioning can also be based on cellular network or even on IP address for rough estimation. It is therefore expected that location-based services are emerging very fast to our everyday life via mobile phones and other consumer electronics.

Locations-based services such as YellowPages[1] , Google Maps[2] and Nokia Ovi Services[3] are traditionally based on databases where all entries have been explicitly georeferenced when stored in the database. An alternative approach has been outlined in (Hariharan et al., 2002) and (Fränti et al., 2010) based on web search and using ad-hoc georeferencing of the web-pages. We denote this approach as *location-based search engine* and emphasize it has seemingly small but significant distinction from traditional location-based services.

The bottleneck of this approach is that only very few pages have explicit georeferencing in form of geotaging, using address field or by other means. On the other hand, it is rather common that web-pages include street or postal addresses as free (non-tagged) text. According to (McCurley, 2001), most of relevant services (especially commercial ones) can be found in this way. The main problem however, is how to find valid address elements from the web-pages both reliably and efficiently.

[1] http://en.02.fi/yellow+pages/
[2] http:/maps.google.com/
[3] http://www.ovi.com/services/

In this paper, we propose a method for extracting street names based on street-name prefix tree and a gazetteer. A potentially relevant web-page (by its content) is first analyzed by extracting all potential street address elements. The hypothesized addresses are then validated by the gazetteer. The pages (or part of them) with validated address are attached by the exact location obtained from the gazetteer and a prototype solution can be found at the *MOPSI Search* website[4].

Extraction of the potential street-name portion of the address field in most languages is very regular. It usually ends to *way*, *drive*, *road*, or in Finnish language to a suffix such as *-katu*, *-kuja*, *-tie*. A simple *heuristic*, used earlier in (Fränti et al., 2010), performs a search for regular expressions with predefined endings (suffixes). However, not all street-names follow the predefined pattern and street-names that have a different suffix, such as *Neulavahe*, would not be detected. We therefore process all strings from the web-page since it can be done at the same cost when parsing the document.

Another problem is that we might detect as an address a portion of text that is not an actual address, causing a false detection. We therefore validate all hypothesized addresses by a gazetteer and discard the false detections. Our gazetteer is a *geocoded* database that contains geographical coordinates attached to address strings. As a side-product, the validation process provides the geocoding, i.e. converts the given address to a pair of coordinates. The process of recognizing geographic context is referred to as *geoparsing* and the process of assigning geographic coordinates to an address is known as *geocoding*.

One way to detect addresses from free form text is to build a classifier and let it detect addresses from the web-pages as in (Viola et al., 2005). However, customizing the classifier to other languages and countries takes a considerable work as new ground truth tagged text corpus must be created by hand. In our approach, no ground truth tagging is needed. The only things needed are a gazetteer and simple rules on how the street name appears in relation to other address fields. Efficient use of the gazetteer is possible because we know the user's current location and its interest area consists only on those services that are close to him. Therefore, we can build fast access structure to that partial gazetteer.

Matching of the potential address strings can be done *brute force* by comparing each word in the document to the retrieved table of street-names. However, this can be rather inefficient if the database is large. We therefore use the prefix tree as a search structure, which is critical for the performance of the matching. A set of prefix trees is constructed from all street-names in a given municipality and the ones in the proximity of the user's location are used. The proposed solution is faster and more accurate than the heuristic solution alone and much faster than the brute force.

2 RELATED WORK

There has been a lot of progress in location-based search during the last years, starting with commercial services like Google Maps, Yahoo! Local[5], Bing Maps[6] and Yellow Pages, or with research projects such as (Jones et al., 2004), (Morimoto et al., 2003) and (Ahlers et al., 2008a).

A spatially-aware search engine (SPIRIT) was developed in (Jones et al., 2004) using geographic ontology, textual and spatial indexing of web-pages. In (Morimoto et al., 2003), a system for extracting geographic information from web-pages gathered by crawling programs is presented, whilst the system in (Ahlers et al., 2008a) relies on web crawling which is targeted to create topical web indices. Our approach differs from these since we don't rely on explicit indexing, but apply ad-hoc georeferencing by detecting postal addresses from free-text.

A categorization scheme of web queries is defined in (Gravano et al., 2003) based on global or local geographic locality. In this view, our search engine handles local queries. In (Wang et al., 2005), three types of locations from web resources are defined: *provider location* (physical location of the provider who owns the web resource), *content location* (the geographic location of the content) and *serving location* (the geographical scope it can reach). Our goal is to search for the content location.

Methods of detecting tagged location of a web resource are found in (Buyukkokten et al., 1999) and (McCurley, 2001). In (Buyukkokten et al., 1999), "whois" records are analyzed and phone numbers of network administrators are used jointly with zip code and area database to assign coordinates to so-called Class A and B domains and to determine the "globality" of a web-site. In (McCurley, 2001), the sources for geospatial context are classified as being for the hosts of a web-page (usually found in "whois" databases and the way the traffic is routed

[4] http://cs.joensuu.fi/mopsi/

[5] http://local.yahoo.com/

[6] http://www.bing.com/maps/

on the Internet), and for its content (postal addresses and codes, telephone numbers, geographic feature names). Additional geographical information is found from hyperlinks and meta tags.

In (Hill et al., 1999), a gazetteer is defined as a geospatial dictionary of geographic names and its minimum components as a geographic name, a geographic location represented by coordinates, and a type designation. Our gazetteer is a geocoded database which contains postal addresses and their corresponding coordinates.

On the other hand, *name entity recognition* without gazetteers is discussed in (Mikheev et al., 1999) and it turns out to work well with people and organizations, but bad with locations. Our solution of postal address detection without a gazetteer (the heuristic method) is much simpler and exploits structural characteristics of postal addresses.

The majority of location-based search systems use gazetteers. For example, the system in (Amitay et al., 2004) uses a three-step process: spotting, disambiguation and focus determination. Our address detection algorithm uses the first two steps.

In (Borges et al., 2007), an ontology-based approach that extracts geographic knowledge is presented. The address is divided into 3 parts: basic address (street and building number), complement (optional, may be neighborhood name) and location identifiers (phone number, postal code, city name). It can be complete, incomplete or partial. The address recognition consists of the processes of geoparsing and geocoding, which uses a gazetteer as described in (Souza et al., 2005). A spatial index (geoindex) is built for each page. The geoparsing process relies on a set of rules and creating patterns implemented as regular expression from four elements: *basic address*, *postal code*, *phone number* and *city/state*. Our approach is different in a sense that it relies merely on text matching, although our heuristic uses matching via regular expression.

In (Can et al., 2005), a syntactic approach to postal address detection is proposed. It consists of two steps: a vision-based text segmentation and a syntactic pattern recognition method. The text segmentation analyses the html tags and detects cue blocks (for the purpose of indications, annotation, and explanation) and body blocks (main text body content). The recognition of postal address relies on calculating the confidence of the detected blocks, which in turn is based on tokenization of the words, which uses city names, state names, street and organization suffixes, but not street names. Our approach is simpler, as we filter out all the html tags before the matching process, and different, as our address detection relies on street-name detection.

In (Cai et al. 2005), location-based data is retrieved by recognizing postal addresses. The method is ontology-based conceptual information retrieval combined with graph matching. The concepts (knowledge/address elements) in a document are identified and linked in the graph by semantic relations. A set of rules is used on the graph and graph matching methods are used to compute similarity and map concept nodes. The concept set used is actually a gazetteer.

In (Silva et al., 2006), a graph-ranking algorithm for assigning the geographic scope of a web-page is proposed. Georeferencing is aided by a geo-ontology knowledge base, which uses a set of rules, relationships and heuristics.

In (Lee et al., 2007), regular expressions are used to detect patterns of typical address elements and database to validate results. The detected street name candidate is then retrieved from the address database to compare all street names for a specific area. In case of a positive match, house numbers are searched and the final address is validated through the database. Our heuristic address detection algorithm is similar to this solution.

In (Ahlers et al., 2008b) a geoparser that identify address level location is built using a database rather than rely on metadata or other structured annotation. The database used by the geoparser contains postal codes, city names, street names, and also every city-postal code combination is also used for validation. The address detection assumes that the address blocks have a certain structure, and that there are certain dependencies between the address elements. We utilize the idea of identifying a number of address elements in our geoparsing algorithm, and validating the address by geocoding it. However, our contribution is that we use own geocoded database and rely on street-name detection based on prefix-trees data structures, while (Ahlers et al., 2008a) uses freely available geocoders.

3 LOCATION-BASED SEARCH ENGINE

3.1 System Description

A location-based search engine is one of the practical applications of the proposed ad hoc georeferencing of web-pages. The basic idea behind the location-based search engine has been presented already in (Hariharan et al, 2002) and the first prototype application, *MOPSI Search*, has been

described in (Kuittinen, 2006) and (Fränti et al., 2010). It consists of the following components:

1. User interfaces for mobile devices and web.
2. Core server software: search engine and database administrator.
3. Geocoded street-name database with spatial indexing: the gazetteer of the project.

Our approach to the location-based search is to use an external search engine for query-based searching and to post-process the search results provided by that engine, extracting the street or postal addresses. These addresses are then translated into coordinates using a geocoded street-name database for result validation and ranking.

The core server software (Figure 1) is the key component in the system as it implements the georeferencing module. It consists of:

1. Relevant municipalities detector
2. Page parser
3. Address and description detector
4. Address validator

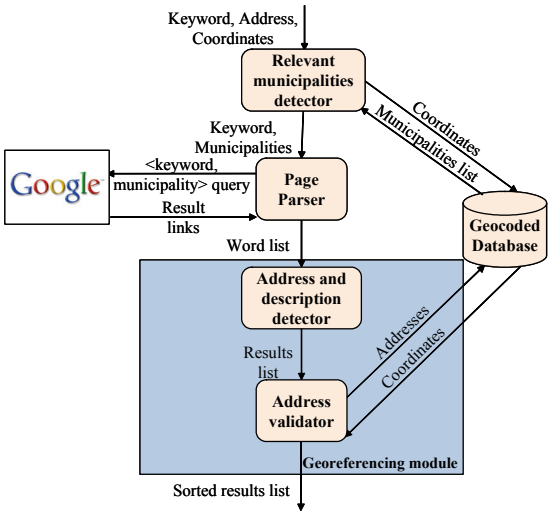

Figure 1: Architecture of the core server software.

The *Relevant municipalities detector* uses the Geocoded database to find all the municipalities that are within a predefined search range (e.g. 10 km square) centered in the user's location.

The *Page parser* uses an external search engine to perform a *<keyword, municipality>* query for every municipality detected at the previous step. It downloads the web-pages found, strips the html tags, and extracts the text.

The *Address and description detector* searches for address blocks, descriptions and telephone numbers in the web-pages found by the Page parser. The *Address validator* uses the Geocoded database to convert street addresses from the previous step to geographical coordinates, then validates and filters the addresses according to a distance threshold. The validated addresses are used to georeference the web-pages.

3.2 Street-address Detection

Current prototype uses a rule-based pattern matching algorithm, which starts with the detection of street-names (Figure 2).

```
StreetNameDetection(words)
{
WHILE i < count(words) DO
   {
   IF words[i] = street name THEN
         {
         Search for street number, postal code and other
         address elements near words[i].
         IF address elements found THEN
               {
               Create address block
               Get coordinates using Geocoded database
               IF coordinates found THEN
                  Add address block to address list
               }
         }
   i = i + 1;
   }
}
```

Figure 2: Pseudocode for address detection.

If a street-name is found, an address-block candidate is constructed by detecting other typical address elements, such as street numbers, postal codes and municipal names. The application looks for these elements in the vicinity of the found street-name. Variations about how addresses are constructed are taken into account. An address-block candidate is validated using the Geocoded database. If the detected address has corresponding coordinates in the database, it is considered as valid; otherwise it is discarded.

Since a plain address without any additional information is not a useful search result alone, the application extracts descriptive information relative to the address. The current implementation simply extracts a part of the text preceding the address as descriptive information. This information is used to create a search result, which is composed of the following: *descriptive phrase*, *telephone number* (if detected), *address*, *web link*, *map link* and *Euclidian distance* between user and the target location.

3.3 Street-name Detection

Street-name detection is the starting point of the address detection. One practical issue is the availability of a gazetteer, as it can be used as a street-name database. Such databases are commonly available, but not necessarily free for commercial purposes. Our application uses a gazetteer of National Land Survey of Finland, and for Singapore, we use the street data from OpenStreetMap[7].

The methods that don't use gazetteer usually assume that a street-name has a certain structure, whilst the methods which use a gazetteer rely on fast word matching. For comparison, we implemented both approaches: a *heuristic method* that does not use a gazetteer, and two text matching methods that use data extracted from a street-name database.

3.3.1 Heuristic Method

Our heuristic method relies on regular expression matching. The structure of most of the addresses has certain particularities. For example, street-names can start with the same prefix or end with the same suffix, they can be in the vicinity of standard words and they are always followed or preceded by a number. In this case, the address block detection also starts with the street-name detection and relies on a set of regular expressions.

According to our experiments, this approach has very good results for Finnish street-names, because most of them end in words like “katu” (street), “tie” (road), “kuja” (lane) or “polku” (path) and has the advantage that it does not need any database or other data structures to store the street-names, and it is reasonably fast.

The accuracy may vary from country to country and the main disadvantage is that the method has to be tailored for every country and language because of the various ways an address block can be constructed. For example, in Finland it is common that the address block has <*street-name, street number, postal code, municipality*> structure, with the street type (e.g. road, lane, street, avenue) as a suffix, whilst in Singapore the <*street number, street name, street type, municipality, postal code*> is more common, but more variations exist. For example, in Singapore a street-name can be written using abbreviations such as Av. instead of Avenue, which is much rarer in Finnish addresses.

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

3.3.2 Brute-force Matching using Street-name Arrays

A brute-force text matching method checks every word in a web-page against a street-name database. We use an optimized brute-force solution that checks the word against all street-names in the proximity of the location the query is made, for example the street-names in the municipality where the user location is.

We use arrays of street-names that are created beforehand from the gazetteer. Each array is used to store all the street-names in a municipality and the search is done using language-specific functions. Since our search engine is written using PHP scripts, we use the *array_search* and *in_array* functions optimized to find an object in an array.

3.3.3 Text Matching using Street-name Prefix Trees

This method uses prefix trees of street-names, which are created beforehand using the information in our gazetteer. The gazetteer used in the project is a geocoded database which contains all the postal addresses in Finland with their corresponding coordinates. For Singapore, the prefix trees were constructed from street names extracted from freely available map data. Statistical data about both gazetteers are detailed in Table 1.

Table 1: Gazetteer statistics.

	Finland	**Singapore**
Number of municipalities	410	1
Total number of street names	92 572	573
Number of streets per municipality	474	573
Average street name length	11.6	6.1
Total size (MB)	2 982	0.18

In general, the postal addresses are not unique, and the same street-name can be found in many cities. A prefix tree is therefore built for each municipality and just the prefix trees corresponding to the search area are loaded during a search.

4 STREET-NAME PREFIX TREE

The prefix tree (or *trie*) is a fast ordered tree data structure used for retrieval (Navarro et al., 2002). The prefix tree stores a collection of strings, indexed from the beginning of a word (i.e. prefix). The root node represents an empty string and its children

store the first letter of the string. The same principle is applied at every level of the tree so that the internal nodes describe all the sub-strings (prefix) of the particular string. The recursive version of the algorithm is presented in Figure 3.

```
ConstructTrie(streetnames)
{
    Create empty node root
    FOR i = 1 TO count(streetnames) DO
        {
        AddString(root, streetnames[i], i);
        }
}
AddString(node, string, index)
{
    IF (length(string) > 0) THEN
        {
        IF (string[0] is not the key of a child of node)
        THEN
            Create new node child with the value
string[0]
        ELSE
            Set child as the child of node with the key
            string[0]
        AddString(child, substring(string, 1), index);
        }
    ELSE  //node is a terminal node
        {
        node.index = index;
        }
}
```

Figure 3: Prefix tree generation pseudocode.

The nodes of the prefix tree can also have values associated with them, although the only values that are commonly used are the values of the leaf nodes and the values of some inner nodes. In our case, we use the values to mark the end of a street name in the tree structure. Usually, only the leaf nodes are the end of a street name, but if a street name is a prefix of another street name, then an inner node can also be the end of a street name.

Dictionary search is one of the most common applications of the prefix tree. It traverses the prefix tree until it reaches a leaf node, or when a node does not have any children whose key contains the desired letter. The recursive version of the prefix tree search algorithm is presented in Figure 4.

In our implementation, we create prefix trees from street-names of each municipality. Therefore, the street-name detection becomes a dictionary search using prefix tree. Because the Finnish street-names usually end with a limited number of suffixes, the names were introduced in the prefix-tree in reverse order and the search in the prefix-tree is done starting from the last letter. Figure 5 gives an example of a prefix tree pre-computed from the Geocoded database.

```
FindString(root, string)
{
    IF (strlen(string) == 0) THEN
        RETURN root.index;  //we have reached last
    node
    ELSE
    {
        IF (string[0] is not the key of a child of root)
        THEN
            RETURN -1; //string is not found
        ELSE
        {
            Set child as the child of root with the key
            string[0]
            RETURN FindString(child,
        substring(string,1))
        }
    }
}
```

Figure 4: Pseudocode of the Prefix Tree search.

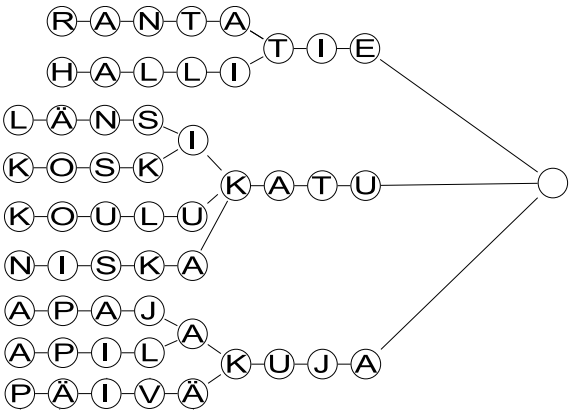

Figure 5: A sample prefix tree built from street-names.

Table 2 summarizes the computed Prefix Trees for Finland and Singapore. It highlights the fact that the gazetteer obtained from the OpenStreetMap is not complete. One of the main advantages of using prefix trees or other pre-built data structures to access street-name data from the gazetteer is the fact that the storage size is reduced (from 3 GB to 74 MB) and the gazetteer is used only for address validation and geocoding.

Table 2: Prefix tree statistics.

	Finland	Singapore
Maximum tree depth	34	14
Average tree depth	12.7	7.4
Average tree width	105	167
Average number of nodes per tree	2338	2335
Total size (MB)	74.4	0.18

5 EXPERIMENTS

We tested the proposed MOPSI location-based search engine using 20 different search locations and 10 keywords to construct <keyword, municipality> queries. We downloaded the content of the first 10 search results for each query of Google search engine and the downloaded content was used as data input for the MOPSI prototype.

The search locations were divided into 2 groups: 10 rural 10 urban municipalities (Figure 6), and the test keywords were divided into 5 commercial and 5 non-commercial ones (Table 3).

Table 3: Keywords used for experiments.

Commercial	hotel, restaurant, pizzeria, cinema, car repair
Non-commercial	hospital, museum, police station, swimming hall, church

The addresses detected by each method were validated using our geocoded database. The size of the downloaded data in the rural and urban municipalities is 13.9 and 11.2 MB, respectively.

Table 4 shows the average time for address detection and the number of detected addresses for the considered municipalities. The average time is calculated per query over all searches. According to the results, the proposed Prefix Tree method is considerably faster than the Brute Force method, and 2-3 times faster than the Heuristic approach, which does not use the gazetteer. Typical search times of the Prefix Tree are less than 1 second per query.

The detected addresses are validated using our gazetteer. The accuracy (number of validated addresses) of the Brute Force and Prefix Tree methods are higher than that of the Heuristic method. The biggest difference between urban and rural municipalities is that the number and the density of streets are much larger in the urban municipalities and, therefore, the methods using gazetteer (Prefix Tree and Brute Force) are slower in rural municipalities. Nevertheless, the Prefix Tree method is the fastest even in this case.

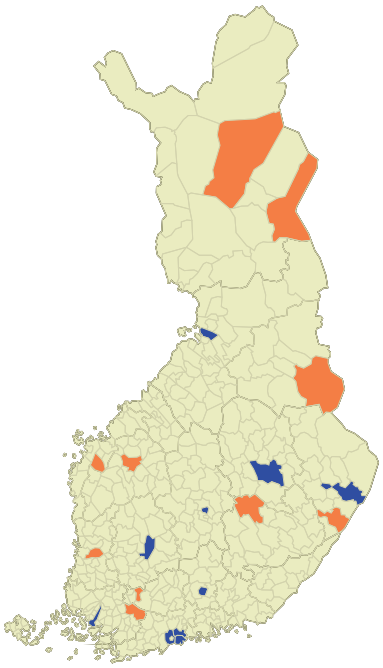

Figure 6: Locations used for experiments. The urban locations (blue): Espoo, Helsinki, Joensuu, Jyväskylä, Kuopio, Lahti, Oulu, Tampere, Turku, Vantaa; the rural locations (orange): Forssa, Kitee, Kuhmo, Laihia, Lapua, Pieksämäki, Salla, Sodankylä, Somero, Ulvila.

The results also show that, for the Heuristic solution, street density and city size do not affect much the search times. In case of Prefix Tree method, the average search time is somewhat bigger in rural municipality (0.51 vs. 0.87 seconds). The Brute Force method is affected most by the street density as the search times in the street array are bigger than the ones in the prefix tree, resulting in more than 3 times longer search time in urban areas.

In total, the proposed Prefix Tree method is twice as fast as and 10% more accurate than the Heuristic method, on average. It reaches the same accuracy than the Brute Force search but using only 10% of the processing time.

6 CONCLUSIONS

Our main goal to design a gazetteer-based street address detector was to increase the accuracy in comparison to the fast heuristic method that was used in the earlier implementation (Fränti et al., 2010). This goal was achieved, as the proposed

prefix tree solution is 57% faster and 10% more accurate, on average, than the heuristic solution. In comparison to Brute Force, it is 10 times faster.

The resulting solution improves the speed and quality of web-page georeferencing and removes one bottleneck for creating efficient location-based search engine as the prototype *MOPSI search.*

Table 4: Average search times for the address detection.

Method	Time (s)	Standard deviation	Number of validated addresses
Rural municipalities			
Brute-Force	3.01	2.43	3.7
Heuristic	1.54	1.15	2.5
Prefix Tree	0.51	0.35	3.7
Urban Municipalities			
Brute-Force	10.18	7.11	19.8
Heuristic	1.70	1.24	18.6
Prefix Tree	0.87	0.85	19.8
Total			
Brute-Force	6.59	6.40	11.8
Heuristic	1.62	1.20	10.5
Prefix Tree	0.69	0.68	11.8

ACKNOWLEDGEMENTS

The research was supported by EU/EAKR and the work of Ville Hautamäki by the Academy of Finland, under project 131298.

REFERENCES

Ahlers D. and Boll S. (2008a). Retrieving address-based locations from the web. *Int. Workshop on Geographic Information Retrieval*, 27-34, Napa Valey, CA.

Ahlers D. and Boll S. (2008b). Urban Web Crawling. *ACM Int.workshop on Location and the web*. Vol. 300, 25–32. Beijing, China.

Amitay E., Har'El N., Sivan R. and Soffer A. (2004). Web-a-where: geotagging web content. *ACM SIGIR Conf. on Research and Development in Information Retrieval*, Sheffield, UK, 273–280.

Borges K., Laender A., Medeiros C. and Davis Jr. C. (2007). Discovering geographic locations in web pages using urban addresses. *ACM Workshop on Geographical Information Retrieval. Lisbon, Portugal*, 31-36.

Buyukkokten O., Cho J., Garcia-Molina H., Gravano L. and Shivakumar N. (1999). Exploiting geographical location information of web pages. *WebDB (Informal Proceedings)*, – dbpubs.stanford.edu

Cai W., Wang S. and Jiang Q. (2005). Address Extraction: Extraction of Location-Based Information from the Web. *Web Technologies Research and Development - APWeb 2005*, Volume 3399/2005, 925-937

Can L., Qian Z., Xiaofeng M. and Wenyin L. (2005). Postal address detection from web documents. Web Information Retrieval and Integration. *Int. Workshop on Challenges in Web Information Retrieval and Integration*, 40 - 45

Fränti P., Kuittinen J., Tabarcea A. and Sakala L. (2010). MOPSI Location-based Search Engine: Concept, Architecture and Prototype. *ACM Symposium on Applied Computing*, Sierre, Switzerland.

Gravano L., V Hatzivassiloglou V. and Lichtenstein R. (2003). Categorizing web queries according to geographical locality. *Int. Conf. on Information and Knowledge Management*, New Orleans, LA, 325–333.

Hariharan G., Fränti P. and Mehta S. (2002). Data mining for personal navigation. *SPIE Conf. on Data Mining and Knowledge Discovery*, vol. 4730, 355-365.

Hill L., Frew J. and Zheng Q. (1999). Geographic names: The implementation of a gazetteer in a georeferenced digital library. *D-Lib Mag.*, January 1999, 5 (1)

Jones C.B., Abdelmoty A.I. , Finch D., Fu G. and Vaid S. (2004). The SPIRIT spatial search engine: Architecture, ontologies and spatial indexing. *LNCS Lecture Notes in Computer Science*, Springer.

Kuittinen J. (2006). Using location information in search engines. *MSc thesis*, Univ. of Joensuu (in Finnish)

Lee H.C., Liu H. and Miller R.J. (2007). Geographically-Sensitive Link Analysis. *IEEE/WIC/ACM Int. Conf. on Web Intelligence*, Silicon Valley, CA, 628–634.

McCurley, K.S. (2001). Geospatial mapping and navigation of the web. *Int. Conf. on WWW*, 221-229.

Mikheev A., Moens M. and Grover C. (1999). Named entity recognition without gazetteers. *Conf. on European Chapter of the Association for Computational Linguistics*, Bergen, Norway, 1–8.

Morimoto Y., Aono M., Houle M.E. and McCurley K.S. (2003). Extracting spatial knowledge from the web. *Symposium on Applications and the Internet*, 326–333.

Navarro G. and Raffinot M. (2002). *Flexible Pattern Matching in Strings*. Cambridge University Press.

Silva M.J., Martins B., Chaves M., Afonso A.P. and Cardoso N. (2006). Adding geographic scopes to web resources. *Computers, Environment and Urban Systems*, 30 (4), GIR, 378–399.

Souza L.A., Davis C.A. Jr.,Borges, K.A.V., Delboni T.M. and Laender A.H.F. (2005). The role of gazetteers in geographic knowledge discovery on the Web. *3rd Latin American Web Congress*, 9.

Viola P. and Narasimhan M. (2005). Learning to extract information from semi-structured text using a discriminative context free grammar. *ACM SIGIR Conf. on Research and Development in Information Retrieval*, Salvador, Brazil, 330–337.

Wang C., Xie X., Wang L., Lu Y. and Ma W.Y. (2005). Detecting geographic locations from web resources. Workshop *on Geographic Information Retrieval*, Bremen, Germany, 17 – 24.