Towards a search system for the Web exploiting spatial data of a web document

Stefan Dlugolinsky
Institute of Informatics
Slovak Academy of Sciences
Dubravska cesta 9
845 07 Bratislava, Slovakia
stefan.dlugolinsky@savba.sk

Michal Laclavik
Institute of Informatics
Slovak Academy of Sciences
Dubravska cesta 9
845 07 Bratislava, Slovakia
laclavik.ui@savba.sk

Ladislav Hluchy
Institute of Informatics
Slovak Academy of Sciences
Dubravska cesta 9
845 07 Bratislava, Slovakia
ladislav.hluchy@savba.sk

***Abstract*—In this paper, we describe our work in progress in the scope of information retrieval exploiting the spatial data extracted from web documents. We discuss problems of a search for web documents by geographic distance, where the geographic distance of a document is determined automatically using information extraction methods. We present here our approach of building a distributed search system, which deals with several problems of this area. Search by geographic distance is useful, for example if we are looking for the nearest restaurant, hotel or any other business near our location (reference point). Almost every company today presents its business on the Internet sharing business information along with contact information. There can be miscellaneous geographic information extracted from the contact information (but no only from it) and used to compute geographic distance of a document. Under a document's geographic distance, we understand the distance between a search reference point and a geographic location related to the document. In our approach, we chose postal addresses and GPS coordinates for spatial data extraction. The reference point can be dynamically changed and one document can be related to more than one geographic location. Geographic locations are automatically discovered in document's textual content. Document is then indexed by all its known geographic locations, so later when searching, the document can be found near different geographic locations to which it is related. Domain of the search is automatically built by crawling through linked web documents.**

Keywords-web crawling; information extraction; information retrieval; geo-coding;

I. INTRODUCTION

Search on the web by geographic distance covers several problems. It is information extraction, geo-coding, indexing and searching. At first, it is important to identify a geographic location of web document. Under a document's geographic location, we do not understand document's physical location, which could be examined from an IP address of its web server, but we understand a location or locations, which are related to a textual content of the document. Here we need to analyze the document and extract as much as possible geographic information from it, such as postal addresses, GPS coordinates, etc. Then we need to use this information to determine geographic coordinates. It is a geo-coding process. As soon as we have geographic coordinates, we can index the document. This is the next problem, because indexing significantly affects searching. We need to index and search documents by their geographic positions considering that we can change the reference point between searches. In the next chapters, we describe, how we dealt with the mentioned problems and how we built a search service, which is able to search web documents by geographic location and sort them by their distance from the reference point.

II. RELATED WORK

There are many information systems, which have something to do with searching in the maps. Well known are Google Maps[1], YellowPages.com[2] or Yahoo! Local Maps[3]. Search domain of these and other existing systems is generally built by users, who submit the location and description of their businesses into the system. Search domain of such systems is also built from specialized catalogues (e.g. restaurant or hotel catalogues). Our approach was to build search service, which automatically builds its search domain by crawling the web. There is a GeoPosition[4] plug-in for Nutch[5] web-search system, which automatically retrieves geographic position of the document from its content, but with the limitation of one geographic position per document. In addition, geographic position must be explicitly defined by special meta tag [5]. In our solution, we also use Nutch web-search system, but we are able to automatically extract more than one geographic position from the document's textual content and index the document by all extracted geographic positions. In the following chapter, we present our web-search system called DistanceSearch.

III. DISTANCESEARCH

DistanceSearch[6] is implemented as a plug-in for Nutch 0.9 web-search system, but with slight modifications of some Nutch core classes. There is originally Lucene 2.1.0 used in Nutch 0.9, but we switched to Lucene 2.9.1, because we had some problems with the older version. The crawl part of the

[1] http://maps.google.com/
[2] http://www.yellowpages.com/
[3] http://maps.yahoo.com/
[4] http://wiki.apache.org/nutch/GeoPosition
[5] http://lucene.apache.org/nutch/
[6] http://distancesearch.ui.sav.sk/

system runs on a Hadoop cluster of 8 nodes, the search part runs on a single node. More about Experiment environment can be found in section IV.A. We have crawled about 408,000 documents from Slovak web catalogues, where in about 58,500 documents 128,000 addresses and 250 GPS coordinates were extracted.

A. Information Extraction

As mentioned earlier, we extract geographic information from document's textual content. Targets of the extraction are postal addresses and GPS coordinates. These entities are probably the most precise information describing a geographic position of the document. Moreover, most contact information in web documents contains postal addresses and/or GPS coordinates. Addresses and GPS coordinates have a formal form of writing, so they can be easily extracted by regular expressions. Various existing methods and solutions for information extraction (e.g. GATE[7]) and semantic annotation [7] are available. Based on our previous work and experience [8], we have developed a simple extraction model (XMLRegExp[8]), based on Java regular expressions, which supports named back references and regular expression macros. This model lets us easily define basic regular expression macros and combine them into more complex patterns without bothering with back reference numbering. Back reference brackets in expressions can be named and their names will be keys to extracted values covered by expression between named brackets. Extraction patterns are defined by meta language in XML (TABLE I.). There are two types of extraction patterns in the model. The patterns with a name attribute, which are macros that can be referenced by their name from other patterns. Second type is a pattern without name attribute, which cannot be referenced from other patterns. It is directly called during extraction and it can refer to macros. Macros can be referenced from any level of inclusion, but there cannot be direct or indirect cyclic references. If we imagine macros and patterns as a connected graph according to their inclusion, it must be a tree.

An example of Slovak address extraction pattern is shown in TABLE I. Slovak addresses are written in the form street_name street_number, postal_code city_name. There are six macros and one extraction pattern. As we can see, the last pattern is the extraction pattern, which refers to STREET_NAME, STREET_NUMBER, POSTAL_CODE and CITY_NAME macros. These macros refer also to other macros.

The address extraction pattern is built from the defined macros according to their inclusion in other macros to the pattern. The pattern is then used to extract key-value pairs. Keys are defined by class attribute and tell, what kind of information is extracted. We can see that the address pattern extracts five entities: street_name, street_number, postal_code, city_name and address. Naming of back reference brackets can be done by groups/group tags within pattern tag, which is not presented here. Enhanced regular expressions are in bold.

TABLE I. AN EXAMPLE OF EXTRACTION PATTERN FOR SLOVAK ADDRESS EXTRACTION

```
<pattern name="POSTAL_CODE" class="postal_code">
<regexp><![CDATA[[0-9]{3}\s*[0-9]{2}]]></regexp>
</pattern>

<pattern name="WORD">
<regexp><![CDATA[(\p{Lu}\p{Ll}*|\p{Ll}+)]]></regexp>
</pattern>

<pattern name="NAME">
<regexp><![CDATA[(\p{Lu}\p{Ll}+)]]></regexp>
</pattern>

<pattern name="STREET_NAME" class="street_name">
<regexp><![CDATA[\p{#NAME}\.?( +\p{#WORD}\.?)?(
+\p{#WORD}\.?)?(+\p{#WORD}\.?)?]]></regexp>
</pattern>

<pattern name="STREET_NUMBER" class="street_number">
<regexp><![CDATA[([1-9][0-9]{0,3} *[/-] *)?([1-9]|[1-9][0-
9])(/?[a-zA-Z])?]]></regexp>
</pattern>
<pattern name="CITY_NAME" class="city_name">
<regexp><![CDATA[\p{#NAME}( +\p{#NAME})?]]></regexp>
</pattern>

<pattern class="address">
<regexp><![CDATA[\p{#STREET_NAME}
+\p{#STREET_NUMBER}(,\s*|\s+)\p{#POSTAL_CODE}\s+\p{#CITY_NAME}]]
></regexp>
</pattern>
```

With this information extraction model, we have tried to extract as much addresses and GPS coordinates as possible. Values extracted as postal addresses needed to be converted to geographic coordinates (geo-coded). The geo-coding process is described in the next chapter. After we have geo-coded extracted postal addresses, we indexed the document by all its associated geographic coordinates, so it could later be found "near" associated geographic places.

B. Geo-Coding

Extracted postal addresses needed to be geo-coded into latitude and longitude values. This is the geo-coding process. We used Google[9] and Yahoo![10] geo-coding services to geo-code extracted addresses. Most of the false extractions (e.g. strings formatted similarly to addresses) were eliminated in this process, because they were not geo-coded. We cached geo-coding results to cut down the number of geo-coding service requests, which is useful especially when recrawling previously crawled documents. We used MySQL database for this purpose.

We have also built our own geo-coding service based on data from the openstreetmap.org[11], but we didn't use it, because we didn't have sufficiently precise geospatial data to geo-code addresses at the level of building numbers. We were able to geo-code addresses only at the level of street names. We decided to use Google and Yahoo! geo-coding services, which are able to geo-code addresses at the level of building numbers and also of postal codes.

[7] http://gate.ac.uk/

[8] http://xmlregexp.sourceforge.net/

[9] http://code.google.com/intl/en/apis/maps/documentation/geocoding/index.html

[10] http://developer.yahoo.com/maps/rest/V1/geocode.html

[11] http://wiki.openstreetmap.org/wiki/Planet.osm

C. Document Indexing

While we were looking for a suitable indexing method, we had to consider Lucene's indexing capabilities inside Nutch as well as the tightness between the indexing and the searching process, as well as the fact that the search reference point is not static. We considered three options: the first one was to index document by latitude and longitude, but we wanted to index a single document by all its geographic positions, so we had to look for another solution.

The second option was to join latitude and longitude values into one string and to index document by it, but this would lead to not very effective searching, because we would have to split every indexed position and compute its distance from the given reference point [1].

The third option was to recursively split geographic surface into hierarchical segments with unique indexes and use them to index document's geographic positions [2]. We have chosen the Hierarchical triangular mesh (HTM) method, which recursively splits spherical surface into triangles with unique indexes, where indexes of the bigger triangles are prefixes of the smaller triangle indexes inside [3][4]. We use the HTM indexes to index geographic positions associated with the document. Except the spatial mapping feature of the HTM method, there is another useful feature, which we use in searching. If there are two points with common prefixes, they are close to each other. The more longer prefix they share, the closer they lie to each other, but this does not apply in the opposite way, because there could be a case of two points, very close each other, where these two points do not have a common prefix. This could happen in the border areas of big triangles, where each point lies in different big triangle, but near each other in smaller triangles. As we will see in the next chapter, this is not a problem when searching. The precision of indexing with the HTM method is increasing with the level of triangle splitting. The precision of indexing Earth's surface with level 25 is near 0.6 m.

D. Document Searching

We search documents according to user query and area of interest specified in a map by rectangular viewport, where the centre of the viewport is the reference point of the search. The distance is limited by the viewport bounds. After the user submits a search query (e.g. "hotel"), we detect indexes of the biggest triangles, which fit into the queried viewport, and use these triangle prefixes to search documents within the viewport (Figure 1). The documents, which are related to the viewport area, have common prefixes with the indexes detected in the viewport. We extend Nutch query by prefix clauses and get relevant documents from the area specified by the map viewport only. Then we order the results by distance using the haversine formula distance computation. Results are displayed in the map viewport as well as in the list (Figure 2).

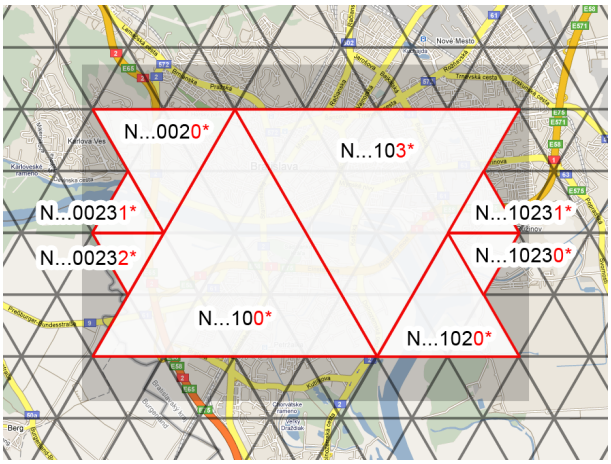

Figure 1. Usage of HTM method for searching and indexing. Indexes of the bigger triangles N...100 and N...103 are prefixes of the smaller triangle indexes inside them. Indexes of their smaller triangles are in the N...100* and N...103* form.

Figure 2. Search interface with map viewport on the left and search results on the right.

IV. Experiment and Evaluation

A. Experiment environment

One of very important DistanceSearch features should be scalability. Thus we need to scale at least crawling, information extraction and indexing. For a real system it would be important to scale also geo-coding and searching, but we did not focus on scaling these two tasks since they perform on much smaller amount of data.

Regarding crawling and indexing, the MapReduce distributed architecture [6] introduced by Google is now known as the golden standard. We discuss it further below together with available implementations.

Concerning information extraction, only very few approaches such as SemTag [9] or KnowItAll [10] work with large data sets, but to our best knowledge only SemTag uses distributed architecture to scale up information extraction (or semantic annotation) task. In our previous work [8] we have implemented and tested information

extraction based on regular expression on Grid and MapReduce distributed architectures.

The MapReduce [6] architecture developed by Google was used with success in information retrieval tasks. Information extraction and pattern based annotation use similar methods such as information retrieval. Google's MapReduce [6] architecture seems to be a good choice for both information retrieval and information extraction for several reasons:

- information processing tasks can benefit from parallel and distributed architecture with simply programming the Map and Reduce methods
- architecture can process terabytes of data on PC clusters with handling failures
- most information retrieval and information extraction tasks can be ported into MapReduce architecture, similarly to pattern based information extraction algorithms. For example, distributed grep[12] using regular expressions, one of basic examples for MapReduce, is similar to our pattern approach using regular expressions as well.
- input and output of Map and Reduce methods are key-value pairs. Porting of our approach is thus straightforward, which was also proved in [8].

Several open source implementations of MapReduce are available:

- Hadoop[13], developed as Apache project with relation to Lucene and Nuch information retrieval systems, implemented in Java. Hadoop is well tested on many nodes. Yahoo! is currently running Hadoop on 10 000 nodes in production environment[14].
- Phoenix[15], developed at Stanford University, implemented in C++.
- Disco[16] is a project started by Nokia. The Disco core is written in Erlang. Users of Disco typically write jobs in Python.

As already mentioned, in our experiment we have used well known Lucene[17] library hidden in related Nutch project. Nutch offered us all needed information retrieval capabilities as well as modular architecture where DistanceSearch plug-in could be integrated. In addition, the Nutch project was an initiator of Hadoop project and thus Nutch is quite well integrated with Hadoop architecture. Thus we have decided to use Hadoop implementation of MapReduce.

[12] Grep is a flexible search-and-replace function that can search one or more files for specified characters and/or strings

[13] http://hadoop.apache.org/

[14] http://developer.yahoo.net/blogs/hadoop/2008/02/yahoo-worlds-largest-production-hadoop.html

[15] http://mapreduce.stanford.edu/

[16] http://discoproject.org/

[17] http://lucene.apache.org

B. Experiment

Since Nutch can also run on Hadoop, we have installed web crawler of the DistanceSearch system on the cluster of 8 nodes (TABLE II.). Crawler runs independently of searcher.

TABLE II. CLUSTER NODE CONFIGURATION

Processor	Intel® Core™ 2 Quad CPU Q9550 2.83GHz
System memory	4 GB
HDD	WDC WD7500AACS-0 (750 GB)
OS	Linux 2.6.24-19-generic x86_64 GNU/Linux

We have configured Hadoop to simultaneously run 8 jobs on every node, i.e. 64 parallel jobs on whole cluster. Searching is realized locally on master node, where we run Apache Tomcat 6.0.20 web server with searcher application. We have launched crawling from the following Slovak web catalogues: http://www.zoznam.sk/, http://www.centrum.sk, http://www.best.sk/, http://szm.sk/, http://www.atlas.sk/, http://www.katalog.sk/, http://www.azet.sk/.

URL filter was set to accept only pages from .sk domains and to accept only text documents (e.g. html, htm, php, txt). The Nutch crawler was executed with max. depth parameter set to 10 and max. 100 000 documents per level. The results of crawling are in the TABLE III. together with the geo-entity extraction results.

TABLE III. RESULTS OF DISTRIBUTED CRAWLING, EXTRACTION AND GEO-CODING.

Crawl duration	20h 46m 7s
Crawled data size (index + cache)	8.3 GB
Index size	535.7 MB
Documents crawled	408 096
Documents without entity extraction	349 561 (85.66 %)
Documents with entity extraction	58 535 (14.34 %)
Extracted entities	245 673
Geo-coding requests	21 463
Average number of entities per document	0.60
Recognized GPS coordinates	256
Not geo-coded address extractions	123 731
Geo-coded address extractions	121 686

Crawl was started from 7 Slovak web catalogues mentioned above.

There were 245 673 geographic entities (not unique) extracted from the set of crawled web documents of size 408 096 documents. This gives 0.6 entities per document, but it does not mean that 60 % of all crawled documents contained spatial data, because we had 85.66 % documents without geo-entity extraction. There were 14.34 % documents with geo-entity extractions, which gives 4.20 geo-entities per document. But these numbers must be considered according to geo-entity extraction recall measure, because some geo-entities were for sure missed. Because there were too many documents in the crawled set, we have done manual evaluation of address extraction on another and smaller set of documents. We have executed the crawler with max. depth parameter set to 10 and max. 500 documents per level. Crawling was launched from two

Google's catalogue pages[18] related to Slovakia and its capital city Bratislava (TABLE IV.). Low precision of address extraction (50.37 %) was affected by high rate of false address extractions. There were extracted strings similar to address structures, which had nothing to do with real postal addresses, but most of these extractions were eliminated by geo-coding and were not converted to geographic positions. Here is the recall evaluation more important, which tells us that about 87.5 % of all addresses were extracted.

TABLE IV. EVALUATION OF ADDRESS EXTRACTION ON SMALLER SET OF DOCUMENTS.

Crawled documents	3183
Relevant documents	380
Addresses in documents	847
Correct address extractions	741
False address extractions	730
Missed address extractions	106
Precision	50.37 %
Recall	87.49 %

TABLE V. COMPARISON WITH SIMILAR SEARCH SERVICES

	DS	**GP**	**GM**	**YLM**
Search domain	web	web	db	db
Geodata extraction from document's text	✓	✗	✗	✗
Geodata extraction from special HTML meta tag	✗	✓	✗	✗
Search in map viewport	✓	✗	✓	✓
Results on map	✓	✗	✓	✓
Sorted results by distance	✓	✗	✗	✗

DS - DistanceSearch, GP - GeoPosition, GM - Google Maps, YLM - Yahoo Local Maps

V. CONCLUSION AND FUTURE WORK

We have created a search system, which is able to search web documents by the geographic distance. Geographic position is automatically discovered from the textual content of the web document and associated with this document. There can be more than one geographic position associated with one document. We plan to distribute the search process from single cluster node to more nodes, like in crawling process. Moreover, we want to extend sorting of the results, so they will be ordered by some ratio between the relevance according to query and to their distance to reference position. We also see a research challenge in associating a geographic position to pages, which do not contain any geographical information, but these pages are related to the page where the geographic information was found. This case is common to small websites, for example company presentation sites, where there is almost always a contact page with company location information and there are also pages describing the company's business. The point is to relate spatial information found on the contact page (but not only on this kind of page) to the other relevant pages within the website. We believe that this could give better results when searching for some business within a map viewport, because contact pages generally contain only brief information such as the name of the company, address, telephone numbers, email addresses, contact persons, etc. More information which can be more precisely covered by search query is on other than contact pages. The main problem within this challenge is to cluster related web pages, because all pages within one domain are not always contextually related to each other.

[18] http://www.google.sk/Top/World/Slovensky/Regionálne/Európa/Slovensko/, http://www.google.sk/Top/World/Slovensky/Regionálne/Európa/Slovensko/Bratislavský_kraj/Bratislava/

ACKNOWLEDGMENT

This work is supported by projects SMART ITMS: 26240120005, SMART II ITMS: 26240120029, VEGA 2/0184/10.

REFERENCES

[1] Gospodnetic, O. and Hatcher, E. Lucene in Action. Greenwich: Manning Publications Co, 2005. 415 s. ISBN 1-932394-28-1

[2] Sahr, K., White, D. and Kimerling, A. J. Geodesic Discrete Global Grid Systems. Cartography and Geographic Information Science, Vol. 30, No. 2, 2003, pp. 121-134.

[3] Kunszt, P. Z., Szalay, A. S. and Thakar, A. R. Dept. of Physics and Astronomy, Johns Hopkins University, Baltimore, MD 21218 in Mining the Sky: Proc. of the MPA/ESO/MPE workshop, Garching, A.J.Banday, S. Zaroubi, M. Bartelmann (ed.); (Springer-Verlag Berlin Heidelberg), 631-637 (2001).

[4] Szalay, A., Gray, J., Fekete, G., Kunszt, P. Z., Kukol, P. and Thakar, A. Indexing the Sphere with the Hierarchical Triangular Mesh; Technical Report MSR-TR-2005-123, 2005, http://research.microsoft.com/pubs/64531/tr-2005-123.pdf.

[5] M. Jaekle. 17. 2. 2006. GeoPosition, http://wiki.apache.org/nutch/GeoPosition.

[6] Dean J., Ghemawat S.: MapReduce: Simplified Data Processing on Large Clusters, Google, Inc. OSDI'04, San Francisco, CA (2004)

[7] Uren V., Cimiano P.,Iria J., Handschuh S., Vargas-Vera M., Motta E.,Ciravegna F.: Semantic annotation for knowledge management: Requirements and a survey of the state of the art. Journal of Web Semantics, 4(1) (2005) 14–28

[8] Michal Laclavik, Martin Seleng, Marek Ciglan, Ladislav Hluchy: Ontea: Platform for Pattern based Automated Semantic Annotation; In Computing and Informatics, Vol. 28, 2009, 555–5

[9] Dill S., Eiron N., et al.: A Case for Automated Large-Scale Semantic Annotation; Journal of Web Semantics (2003)

[10] Etzioni O., Cafarella M., Downey D., Kok S., Popescu A., Shaked T., Soderland S., Weld D., Yates A.: Web-scale information extraction in knowitall: (preliminary results); In WWW '04: Proceedings of the 13th international conference on World Wide Web, 2004, 100-110, http://doi.acm.org/10.1145/988672.988687