Semantic Approach for Web Information Monitoring

Wiesław Lubaszewski, MichałKorzycki, Krzysztof Dorosz
Department of Computer Science
AGH University of Science and Technology
Kraków, Poland
{lubaszew, korzycki, dorosz}@agh.edu.pl

ABSTRACT: *Web Monitoring is an important task for crime detection and crime investigation. This paper describes the rationale, functionality and the architecture of the MPI system, which was developed for Web information monitoring based on advanced semantic scripts based on the Conceptual Dependency model.*

Keywords: Web Monitoring, Information Retrieval, Natural Language Processing, Web Crawling, Text Similarity, Conceptual Dependency, LSA

Received: 08 November 2012, Revised 3 January 2013, Accepted 10 January 2013

1. Introduction

It is known that the knowledge derived from verbal information can play an important role in threat detection, crime investigation and business activity. Today, there is an urgent need for appropriate tools to extract and process text information stored in the Internet and text corpora derived from the Web. By appropriate we understand matching a clearly defined set of requirements that have been defined as crucial by the potential end users. Those requirements are, in no particular order:

Accuracy – the system mechanism should address the meaning regardless of the form the information has been presented in, i.e. the text samples below should be both recognized as referring to the act of "*tea drinking*":

A. *Johnny drank some tea.*

B. *The sunlight flooded the room, which was filled with the aromatic fragrance of the tea. Johnny slowly reached for the cup. He savored the first sip for a long time. The sun of the Incas magnanimously smiled upon him.*

Practicality – for various (not only economic) reasons we are not able to store the whole Internet on disk and then perform a search. This implies that we must develop a mechanism, which will perform a search on a single text extracted from a website. If the text matches the search criteria it will be stored on disk. The latter determines the cascade architecture for the system: text-extraction from the accessed web site, information extraction and semantic evaluation, that is followed by storing the retrieved information on disk only in the case of a successful evaluation.

Rough Text Orientation – if we work with a text derived from the Internet we can not expect the comfort given by a set of training data. We can rather expect texts, which are ill formed, incomplete, full of spelling errors, etc. This means we must find an appropriate model for a information pattern and appropriate lexical tools.

End-User Orientation – that in itself is a complex issue, but we successfully were able to reduce it into two more detailed requirements, which mean that the tool must be:

Easy to Use – the development of information patterns for a search, which meets the accuracy criteria formulated above should not require advanced linguistic knowledge from the user. This can be achieved by creating an intuitive interface, supported by a rich library of pre-defined patterns for general events that need to be customized.

Rapid Deployment Ready, which means that the search should start within hours since the search need is recognized.

Integration Readiness – using well defined and open formats permits the easy integration with third party tools. Frequently in a criminal investigation it is not enough to find an unknown website or specific information within a known website, e.g. the time and place of a probable hooligan's meeting. In some cases there is a need to find a person related to a particular content, e.g. a server administrator or the text author. In such cases our search engine would be augmented to an **update monitor**, i.e. a mechanism which will look for the update of a monitored website. If a website update occurs, the update monitor will report the date and time of the update to a 3rd party **traffic controller**, i.e. a mechanism, which will look for the IP address from which the update has been made.

2. Rationale Behind the Taken Approach

Most pattern learning systems developed at the end of the 90s and the beginning of the new Century used annotated training data to learn patterns to find the information in new texts. But due to the manual labor involved in annotation a corpus, **the annotated training sets are very restricted and relatively small** [24] [25] – to quote Yarowsky [23] - '*The need for domain-specific training data has several disadvantages. Because of manual labor involved in annotating a corpus, and because a new corpus must be annotated for each domain, most annotated IE* [*Information Extraction*] *corpora are relatively small.* [...] *Consequently, the IE patterns learned from manually annotated training sets typically represent only a subset of the IE patterns that could be useful for the task'.* That leads in turn to an approach relying on unannotated data to improve performance [26]. Unfortunately, the learning from unannotated data must be an iterative process. First, we must obtain *seed patterns* derived from an external source, e.g. for victims and targets of terrorism [28] [31] or *seed examples* of a given class, if we look for named entities extraction patterns [27]. Then we must search the Web to extract new patterns, which quality we must assess again, even if we use control mechanism to improve extraction quality [32]. In other words, the machine-learning approaches lead to solutions that are costly in terms of manual labor required to process the end-results and are not rapid-deployment ready – due to the need for a lengthy process of training the system. In addition, patterns extracted automatically are relatively simple, then there is a real danger that learning complex patterns, which will meet our accuracy principle, would be error prone.

Therefore we have to look for a model, which would bring the highest possible precision and will not affect the recall, in the same time preserving the requirements that we listed for an appropriate semantic search system. We believe that such a model for an effective semantic search system:

must use the same mechanism to extract information from both a single sentence and a story,

must be an associative one, i.e. able to infer on the basis of semantic connectivity knowledge, to work properly with a text, in which textual gaps occur,

must be domain independent.

To find a model, which would perform that way we must reach back to text understanding theories developed in the 70s [29][30]. As a model for an associative semantic information pattern we use the script structure developed by the Conceptual Dependency theory [20] [4]. In general, one may present a CD script as a semantic prototype, which describes an event as a structure built from events. Accordingly to the CD theory we may present '*tea drinking*' as the sequence of events: '*tea making*', '*cup operating*', '*tea sipping*' etc.. Then if we take a closer look at our texts we shall find that '*tea sipping*' exactly matches text B, but even the whole sequence of events does not match the text A. Therefore if we want to built a script-like information pattern to

match both texts, we must extend the original script structure and distinguish a synthetic section 'tea drinking' to match text A and an analytic section represented by the sequence: '*tea making*', '*cup operating*', '*tea sipping*' to match text B.

The script-like information pattern permits the inclusion of advanced expert knowledge into the search process. It is easy to create – one may use a general pattern prepared in advance, only to tune it. And - what is probably most important – script-like information pattern does not require preloading any information from the Web, but can be used directly on the Web to process data in real-time.

3. MPI – the Monitor for Polish language Internet

In general, each system designed to extract information from the Web should consist of a web crawler with an embedded text extractor, an information extractor and a text corpus to store results. If one thinks about Web information monitoring, where monitoring means a persistent look-up of the Web content, one should adopt this general architecture to match a monitoring goal. The **MPI (Monitor for Polish language Internet)** system to be described here has four monitoring goals: (A) to find a website, which brings specific content, (B) to find expected content in a specific website, (C) to find an update of a specific website and (D) to find similar contents in a specific website. Therefore the MPI system consists of three major components: a web crawler with an embedded extractor of consistent text, four functionally different information monitors, each of which matches a particular monitoring goal and a text repository, where each text is marked by links, a symbolic address and IP address of its server, etc. Each information monitor consists of dedicated crawling strategy and an appropriate information extractor.

4. Web Crawler

A crawling system is a very basic and important element of any Web information monitoring system. Crawling gives the possibility of processing Web content in a systemized, automatic manner. There are many well-described approaches to the Web crawling problem [12]. FishSearch [6] and a more advanced variation of this algorithm, SharkSearch [14] are historical examples of more refined notion of scoring links by their query similarity. Naïve Best-First crawlers [17] present a simple extension of the BFS approach with cosine similarity based on term frequency. Modern generic indexing Web search crawlers like Google [1], Bing and Yahoo use wide search strategies driven by many performance indicators, which allow achieving large links coverage and freshness [15]. There are also dedicated crawlers, i.e. topic driven focused crawlers [2] and context crawlers [7]. The difference between them is that unlike the focused crawlers, context crawlers are trying to build a classificatory for a typical context in which relevant documents can appear, making the search wider.

If we compare focused crawling (also named topic or topical crawling) to general-purpose crawlers or Web indexers, we can say, that instead of crawling as many webpages as possible to cover a vast amount of URL target space and create an index for all possible ad-hoc queries, a focused crawling is driven by a pre-defined query [2]. The goal of that crawling is to selectively explore documents in the Web, retrieving only those, which are relevant to the given query (also called the topic). To achieve this goal, focused crawling incorporates dedicated crawling strategies, which provide in an ideal situation optimal links ordering in crawling frontier. This part of a crawler is especially important, because it gives the crawler an ability to intuitively drive crawling to those unknown URL that are most probable to point on relevant documents, but at the same time it avoids searching a vast amount of probable not interesting addresses [8]. Considering the time limitation for single topic crawling and the scale of the Web, an efficient crawling strategy is crucial even for finding any results at all. To achieve a good performance of focused crawling many different approaches were examined [17] [18]. The approaches differ in methods for topic representation and recognition as well as in measures for crawler performance evaluation To represent a topic many techniques are used, e.g. boolean models with bag of words and keywords [3] or statistical techniques including naive Bayesian methods [13] and LSI [5]. Despite the fact that focused crawling reduces significantly the scope of the browsed Web sub-graph, it still should deal with a very large size of frontier and the size of a visited links cache [10].

Despite the fact that there are many ready to use implementations of generic web crawling systems, each application needs to be separately customized. In other words, each time there is a need of a very specific tool suited to a specific role. The MPI system specifies four different types of system goals, which cannot be satisfied by single crawling strategy due to the enormous complexity of the WWW network (e.g. black holes avoidance). Therefore, different crawling strategies are defined on the top of a standard crawling library.

5. Text Extractor

The global goal of the MPI is monitoring of textual information. Therefore there is a need for an efficient method, which would extract a text from the WWW structure. The purpose of this method is to split web site content into coherent texts, which would increase the performance of linguistic algorithms used to process a text. Some different approaches to solve this problem were described in [9]. At this stage of the MPI system development, the text extraction method is implemented as a hybrid of two described methods: the energy method and the anchors analyzing method. Both of them use only structural information from HTML DOM. The general idea of the implemented hybrid method is to split a HTML DOM tree (of a given page) into parts that collect relatively large amounts of localized text, but in this same time these DOM parts are relatively distant from each other in the original input. After that a second phase is processed, during which all anchors are analyzed. This should help to determine, if selected anchors are a part of a coherent text. If they are not, texts are split further.

6. Information Monitors

At present, we have four information monitors, each of which matches a particular monitoring goal: (6.1) web site finder, (6.2) an event monitor, (6.3) an update monitor and (6.4) a text similarity monitor.

6.1 Website Finder

The Website Finder matches the monitoring goal 6.1., i.e. it is devoted to find in the Internet a still unknown WWW service, which contains highly specific information**,** e.g. sectarian texts or "*prescription*" for home made explosives hidden in an Internet based cook book, etc. It consists of a dedicated crawling strategy and a content extractor

6.1.1 Crawling the Internet

If one is going to find a www service, which contains highly specific information, then one needs a focused crawling. In general we use a strategy that is able to cover a large selection of domain/hosts, while keeping the ratio of pre-selected links per domain relatively small. Starting from seeds generated from popular Web indexers, crawling is focused by a given query using an evaluation given by the Context Extractor.

6.1.2 Content Extractor

It is known that the human ability to express the same information in many different ways causes a major threat to search accuracy. Both texts below provide a good example:

A. *Johny drank some tea*.

B. *The sunlight flooded the room, which was filled with the aromatic fragrance of the tea. Johny slowly reached for the cup. He savoured the first sip for a long time. The sun of the Incas magnanimously smiled upon him*.

There is no doubt that human beings would easily find that both texts are about '*tea drinking*'. This result is presumably based on the knowledge that a 'sipping' is a part of the 'drinking' process and that tea is a '*drinkable liquid*'.

The most frequently used key word based search engine is unable to reach human-like efficiency and it is out of the matter, how we create the search pattern - is it human typed or learnt by an algorithm from the training corpus. It would omit the text B, if we restrict the search pattern by conjunction of the key words "drink & tea". If we do not restrict the pattern we will get an extreme overflow of results, most of them matching only one key word and the text B would be hidden behind an extreme number of false-positives in the output set of texts.

We argue that there is a need for an associative content extractor, which can associate information explicitly missed in the text with an information pattern to preserve the accuracy of content recognition. In other words, we need an information extractor, which will extract the same information from both texts above. We also argue that the key concept is an information pattern model.

6.1.3 A Script-Like Information Pattern

The Content Extractor reads the text sentence by sentence, trying to match an event defined in a search pattern. If it matches an event, which is a synthetic pattern, a text matches the pattern. If it matches an event, which belongs to an analytic pattern, the extractor continue a text reading trying to match the next analytic event until it reaches the end of the text. If a text matches script-

like pattern, the content extractor sends a text to a repository.

In real use the precision of content recognition reaches 96,67% [22] and we believe that this measure is adequate to the system goal.

6.2 Event Monitor
The Event Monitor matches the monitoring goal 6.2 i.e. it can be used when a specific information needs to be found on a particular website, for example the time and place of a probable meeting of hooligans.

6.2.1 Crawling for a Site Restricted Search
In this type of monitoring goal, a depth-first search (DFS) strategy is used. The crawling is limited to a given set of domains. The domains are searched as deeply as possible. In order to avoid black holes (aka. spider traps) some limitations on the maximum depth per domain or number of processed pages can be given.

6.2.2 Content Extractor
Because of the fact that the information to be found is expressed in a text form, the Event Monitor uses the same content extractor as the Website Finder.

6.3 Update Monitor
The Update Monitor matches the monitoring goal 6.3, i.e. it looks for man-made updates. There are two reasons to monitor a man-made update of a particular site.

The first task is to find a person related to a particular content, e.g. a server administrator or the text author. In this case the monitor must look for newly added text.

The second task is to find, if a particular site is a platform for hidden communication. In this case the monitor must look for man-made text update.

If a website update occurs, the update monitor will report the date and time of the update to the network analyst, who will look for the IP address from which the update has been made.

6.3.1 Crawling for a Site Restricted Update
The crawling policy for the Update Monitor is based on a BFS strategy. This is made possible by limiting the crawling to only one domain, host or page. In this policy, a time period must be specified. This timeout applies to the time after which the re-crawling of the URL pool must be done. The crawling is executed top-bottom through the links available as long as one of the stop conditions are met. A stop condition occurs, if there are no more pages to crawl or a specified time limit has elapsed. This strategy is adaptive to the system resources, allowing us to fetch as many pages from top to bottom as possible in a given time period, with the given resources available.

6.3.2 Site Update Extractor
A man-made update of a website means the adding of a new text to the website. To monitor such an update is a relatively simple task. Having stored all the website texts in the repository in the first run, the update extractor works in the loop matching a newly crawled text against each text in the repository. If the newly crawled text does not match any text in a repository, the update is found.

6.3.3 Text Update Extractor
By a man-made update of a text we consider the insertion of a new word into a text or a substitution of existing word in the text, if both the insertion and substitution make no visible distortion of grammatical or semantic text structure, as for example the substitution:

*He savoured the first sip for a long time. The **sun** of the Incas magnanimously smiled upon him.*

*He savoured the first sip for a long time. The **moon** of the Incas magnanimously smiled upon him.*

This is a really tough task. The extractor periodically matches the newly crawled text against each text in the repository. But matching of the check sum assigned to the text by the first run against the check sum assigned by each iteration is not enough.

The same content provided by the crawler can differ slightly from day to day. This is mainly caused by automatic add-ons to the text such as: dates, counters, random quotes, a calendar information and so on. Therefore there is a need for heuristics, which should recognize the aforementioned automatic add-ons. The global heuristics, which are based on the structural dependencies of HTML DOM, may truncate automatically an update, which occur repeatedly in a particular structural context, can be helpful but are not efficient enough.

The automatically added text to the content can be divided into two separate groups, depending on the frequency of its update. One group are text that are constant over a period of time (for example a day) – such as dates, quotes of the day, names celebrated on a given day. The other groups are add-ons taken from a database – such as a database of quotes. The former are usually well defined and easily recognizable, and what's more important shared between most of the pages of a site in a given timeframe – what can be recognized with a high accuracy (we obtain a precision of over 80% on analyzed examples for this category). The latter type is much harder to discern and categorize as it varies from page to page and site to site. There is a need for local heuristics even related to a particular website to reduce the number of false-positives. And there is no doubt that the final assessment of a supposed text update must be made by a human.

6.4 Text Similarity Monitor

There are cases in which one cannot use structured information patterns, due to the fact that the information to be extracted is not precisely known. In such a situation one can use short texts or text samples as an information pattern to look for texts, which bring similar information in a particular text corpus, e.g. e-mail server resources, server logs, etc.

6.4.1 Crawling for Site Download

The crawling policy of the monitor of type 6.4 is very similar to the 6.3 type strategy. The only difference is that no re-crawling timeouts are present in this case. Pages are fetched as long as links are available in a links pool or as long as a specified limit of pages has not been reached.

6.4.2 Text Similarity Extractor

We use the Latent Semantics Analysis (LSA) as the model for the text similarity extractor. The tool will accept any user provided text and will return a number of texts selected from the text corpus. All returned texts contain information similar to the information of the input text. If one takes for example the input text from the corpus of the Polish News Agency PAP:

The two Siamese twins, Weronika and Wiktoria were born on May 26, 1999, *in the hospital of the Academy of Medicine in Lublin. They have been in Philadelphia since August* 16. *They survived a successful operation to separate them in November. The first date to release the two from the hospital was February* 11. *It was later extended for a week due to Weronika's fever.*

Then it can be associated with other texts of the corpus like:

The French actor Gerard Depardieu, has had a bypass operation in a suburban Paris hospital – as reported by the doctor who performed the operation. Similarity: 0.527882

or

A swarm of bees has attacked more than 300 *people in Pakistan, including many doctors who were participating in a charity march, which was being held to collect money for a hospital.* Similarity: 0.338710

The example is taken from [16].

The LSA is a statistical algorithm, which extracts word co-occurrences in the frame of a text. As a result each word in the corpus is related to all co-occurred words and all texts in which it occurs. This makes a base for an associative text comparison. But due to its statistical nature the LSA is not able to distinguish co-occurrences, which are corpus independent semantic dependencies (elements of a semantic prototype) from co-occurrences, which are corpus dependent factual dependencies [21]. This fault brings an unexpected value – the LSA can associate facts described in texts in a way, which may by omitted (rejected) by human reasoning based on semantic and factual stereotypes and driven by *the principle of the least effort*. This ability makes the LSA algorithm a valuable tool in man-made analytics – a analyst can find in the LSA output an unexpected or unpredictable factual association. It seems to be clear that we cannot measure this aspect of LSA performance by the classical precision/recall pair of measures.

7. Text Corpus

The basic format used for storing specific corpora contents is the text-based format, similar to the popular **mbox** format. All texts specific to a search task are stored in a flat file. The beginning of each text is marked by a **From** followed by a space and the task for which the text was downloaded. After this line follows a set of lines containing the text metadata (the **header**). The metadata describes such properties as a time of download, task id, relevance level to the script that accepted it etc. etc. A blank line is appended to each text and lines beginning with "*From*" are escaped. All texts in the corpus share at least the following header types:

***task*:** the id of the task that generated the text – see. The Task

***title*:** the title of the page on which the text was found

***url*:** the address from which the text has been downloaded

***timestamp*:** the time when the text has been encountered on the World Wide Web

***flow*:** this header is changed on each processing stage – it consists of a list of the system components that are supposed to process the text

***digest*:** a hexadecimal digest of the message, trimmed from any white spaces. This permits us to avoid "*spider traps*" and redundant data.

In addition to this raw data format, multiple indexing files are generated "*on-the-fly*". Those indexes contain pre-processed corpora, in order to speed up text browsing and look-up. Those indexes range from a purely technical nature (such as the set of text digests to reject redundant entries in the corpus), to domain specific (such as the aggregate index – aggregating similar texts from the same site).

8. System Architecture and Implementation

The base architecture of the MPI System is asynchronous in nature. Each message processed by the system goes through a predetermined (but not immutable) flow that guides it through a set of processing units. These processing units are both consumers as producers of the main queuing system that is used for message exchange. On each processing stage, the messages, as composed of a body and headers, can be modified or rejected. This operation is carried by the extractor modules – each attached to a specific information pattern. The modifications consists mainly of updating the message headers (such as adding a message rating – the measure of its relevance to the matching scripts). The rejection of a particular message depends on the level of relevance to the matching script – a message rated below the acceptance threshold is rejected. The final stage of processing for a message consists of one of the Corpus Units, which are responsible for the persistent storage, indexing and lookup of messages. Each Corpus Unit is responsible for one of these manners of task types – 6.1 (website finder), 6.2 (event monitor), 6.3 (update monitor) or 6.4 (text similarity monitor).

8.1 Internal Task Structure

Every search job performed by a system is recognized as a crawling task. It has the following properties that must be provided by the user:

• a type (one of the available ones: A, B, C or D),

• a name and a description, e.g. B23,

• an information pattern to be evaluated with, e.g. portfel.yml; in the case of a C or a D kind of task this information is discarded, so it can be left empty,

• a set of initial links to start the link traversing, e.g. www.aghh.edu.pl,

• a whitelist rule set, i.e. links always accepted,

• a blacklist rule set, i.e. links always rejected,

• a defined priority level from 1 to 100 (typically 50),

• a threshold value of the maximal number of links to be processed,

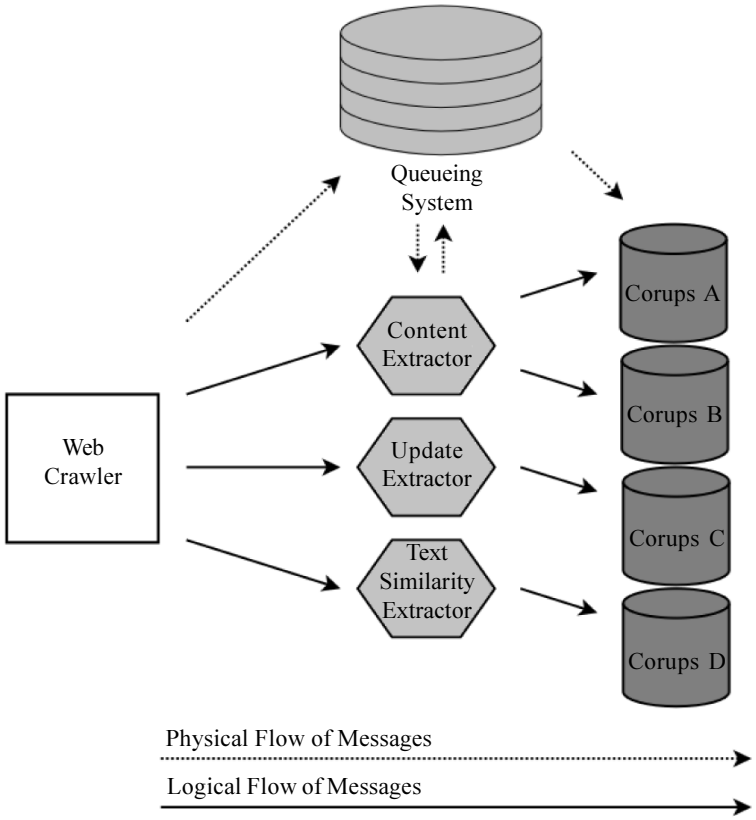

Figure 1. The System Architecture

• a threshold value of the maximum traversing depth (typically 5),

• an explicit crawling end date,

• a refresh rate given in hours, typically 24,

• a timestamp for restarting the starting time, for example, if it is set to 2010-05-07 15:00:00 and the refresh rate to 24 hours, the crawling task will restart itself every day at 15:00.

All the information about a task can be provided through a dedicated GUI wizard. The tasks with all their data are then stored in the database and are identified by a task ID. A task ID is an identifier having following format <TASK TYPE><SEQUENCE NUMBER> (e.g. B23, A1, C45). This identifier is unique for the whole system, and can be used as a referral for a precise description of an actual single crawling job. This is an important feature, as there is a possibility of running more then one crawling job with identical properties.

After adding a crawling task to a task queue the system sets also the following properties (that cannot be edited by the user):

• a task identifier,

• a creation and modification date,

• a closing date,

• state (active, paused, closed),

• an owner.

The enlisted data are also stored with a crawling task in the database and provides information about what kind of jobs were run in a specific period by which user.

Tasks that stop after matching the given stop criteria are switched to a "*close*" state and can never be deleted from the database.

Deleting a crawling task after running is not supported in the system. Tasks can be only closed, which leaves a full log of jobs that were started with the system.

Each task defines its own *flow* list. On each stage of the processing, this list is reduced by truncation of its last element. If the list becomes empty – the text processing finishes. Then the corpus daemon stores the text to disk.

The results of the processing are available online through a web interfaces, which can be used for both managing the web search tasks and retrieving the task results.

9. Conclusion

MPI monitors were tested separately since April 2008. The test results and user feedback have proven that the MPI may have useful applications – ranging from business intelligence to forensic analysis.

One must stress that the software described in the paper is designed to search WWW services open to public access. Therefore, it does not violate the law and does not violate the privacy rights of a WWW service owner or author. In addition, the task construction and the text corpora format makes auditable any single use of MPI, which may reduce a threat of a potential misuse of the system.

10. Acknowledgment

This paper was supported by EC under grant FP7-218086, the INDECT Project.

References

[1] Brin, S., Page, L. (1998). The Anatomy of a Large-Scale Hypertextual Web Search Engine, *In*: Proceedings of the Seventh International

[2] Chakrabarti, S., van den Berg, M., Dom, B. (1999). Focused crawling: a new approach to topic-specific web resource discovery.

[3] Cho, J., Garcia-Molina, H., Page, L. (1998). Efficient crawling through url ordering. Computer Networks and ISDN Systems. *In*: Proceedings of the Seventh International World Wide Web Conference, 30 (1-7) 161 – 172.

[4] Cullingford, R. (1981). SAM, In Schank, R., Riesbeck C. K. eds. (1981), Inside Computer Understanding, Five Programs Plus Miniatures, L. Erlbaum, Hillsdale, NJ.

[5] Deerwester, S. C., Dumais, S. T., Landauer T. K., Furnas, G. W., Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41, 391–407.

[6] De Bra, P. M. E., Post, R. D. J. (1994). Information retrieval in the World Wide Web: Making client-based searching feasible. *In*: Proc. 1st International World Wide Web Conference.

[7] Diligenti, F., Coetzee, S. L., Gilesand, C. L., Gori, M. (2000). Focused crawling using context graphs. *In*: Proc. 26th International Conference on Very Large Databases (VLDB 2000), p. 527–534, Cairo, Egypt.

[8] Dorosz, K. (2012). Focused crawling strategies for information monitoring in the Polish language Internet. PhD thesis, AGH-UST.

[9] Dorosz, K. (2008). Ekstrakcja spójnych wiadomoœci tekstowych z Internetu na potrzeby algorytmów lingwistycznych, Automatyka 2008/2, Wydawnictwo AGH, Kraków.

[10] Dorosz, K. (2009). Usage of dedicated data structures for url databases in a large-scale crawling. *Computer Science Journal*, AGH University of Science and Technology, 10, 7–17.

[11] Figiel, A. (2009). Tekst jako wzorzec informacyjny – automatyczna ocean podobieñstwa tekstów za pomoc¹ Latent Semantic Analysis [The Text as informational pattern - a Latent Semantic Analysis as a tool for texts comparison]. *In*: W. Lubaszewski ed. Słowniki komputerowe i automatyczna ekstrakcja informacji z tekstu [Machine Readable Dictionaries and Automatic Extraction of Information], Kraków.

[12] Pant, G.. P., Menczer, F. (2004). Crawling the web. *In*: Web Dynamics: Adapting to Change in Content, Size, Topology and Use. Edited by M. Levene and A. Poulovassili. Springer-Verlag.

[13] Zhang, H., Lu, J. (2007). A fuzzy approach to ranking hyperlinks. *In*: Proceedings of the Fourth International Conference on Fuzzy Systems and Knowledge Discovery - 03, 406–410, Washington, DC, USA. *IEEE Computer Society*.

[14] Hersovici, M., Jacovi, M., Maarek, Y. S., Pelleg, D., Shtalhaim, M., Ur, S. (1998). The shark-search algorithm — An application: Tailored Web site mapping. *In*: WWW7.
[15] Cho, J., Garcia-Molina, H. (2003). Effective page refresh policies for web crawlers.ACM Transactions on Database Systems, 28 (4), December.
[16] Lubaszewski, W., Korzycki, M. (2011). INDECT deliverable D9.15. System for Enhanced Search: A Tool for Associative Information Retrieval, Krakow.
[17] Menczer, F., Gautam, P., Padmini Srinivasan, Miguel, E., Ruiz. (2001). Evaluating topic-driven web crawlers.
[18] Passerini, A., Frasconi P. and Soda G, (2001), Evaluation methods for focused crawling. *In*: Proceedings of the 7th Congress of the Italian Association for Artificial Intelligence on Advances in Artificial Intelligence, AI*IA 01. London, UK: Springer-Verlag.
[19] Schank, R. (1975). Conceptual Information Processing, North Holland, Amsterdam
[20] Schank, R. C., Abelson, R. (1977). Scripts, Plans, Goals and Understanding, An Inquiry into Human Knowledge Structures, Lawrence Erlbaum, Hillsdale N.J.
[21] Dorosz, K., Korzycki, M. (2012). Latent semantic analysis evaluation of conceptual dependency driven focused crawling. In Andrzej Dziech and Andrzej Czyzewski, editors, *Multimedia Communications, Services and Security*, V. 287 of *Communications in Computer and Information Science*, p. 77–84. Springer Berlin Heidelberg.
[22] Lubaszewski, W., Dorosz, K., Korzycki, M. (2009). D4.4. system for enhanced search: A tool for pattern based information retrieval. Technical report, INDECT Project, FP7-218086-Collaborative Project.
[23] Yarowski, D. (1995) *Unsupervised Word Sense Disambiguation Rivaling Supervised Methods*, *In*: Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics, Cambridge, MA.
[24] Collins, M., Singer, Y. (1999). *Relational Learning of Pattern-matching Rules for Information Extraction*. *In*: Proceedings of Joint SIGDAT Conference on Empirical in Natural Language Processing and Very Large Corpora, College Park, MD.
[25] Riloff, E., Jones R. (1999). *Learning Dictionaries for Information Extraction by MultilLevel Bootstraping*. *In*: Proceedings of Sixteenth National Conference on Artificial Intelligence, Orlando, FL.
[26] Nigam, K., McCallum, A., Thrun, S., Mitchell, T. (2000). *Text Classification from Labeled and Unlabeled Documents using EM*, Machine Learning, 39 (23).
[27] Pasca, M. *Weekly-Supervised Discovery of Named Entities Using Web Search Queries*, CIKM'07, Lisboa.
[28] Patwardhan, Riloff. (2006). *Learning Domain-Specific Extraction Patterns from the Web*, *In*: Proceedings of the Workshop on Information Extraction Beyond The Document Pages, Sydney.
[29] Schank, R. (1975). *Conceptual Information Processing*, North Holland, Amsterdam.
[30] Schank, R. C., Abelson, R. (1977). *Scripts, Plans, Goals and Understanding, An Inquiry into Human Knowledge Structures*, Lawrence Erlbaum, Hillsdale, N.J.
[31] Wiebe, J., Riloff, E. (2005). *Creating Subjective and Objective Sentence Classifiers from Unanntotated Texts*. *In*: Proceeding of the 6th International Conference on Computational Linguistics and Intelligent Text Processing, Mexico City.
[32] Downey, D., Etzioni, O., Soderland, S., Weld, D. (2006). *Learning Text Patterns for Web Information Extraction and Assesment*. *In*: Proceedings of the Workshop on Information Extraction Beyond The Document Pages, Sydney
[33] Allan, J. (2004). *HARD Track Overview in TREC* 2003, The Twelfth Text REtrieval Conference (TREC 2003) Proceedings, NIST.
[34] Buckley, C., Robertson, S. (2009). *Relevance Feedback Track Overview*: *TREC* 2008. The Seventeenth Text REtrieval Conference (TREC 2008) Proceedings, NIST.