Automatic Query Refinement Using Mined Semantic Relations

Jens Graupmann, Jun Cai, Ralf Schenkel
Max-Planck Institut für Informatik, 66123 Saarbrücken, Germany

E-mail: {graupman,schenkel}@mpi-sb.mpg.de

Abstract

With the Web of today being unstructured and semantically heterogeneous, keyword-based queries are likely to miss important results. Therefore, refining *and* expanding *queries plays an important role today. This paper presents a novel approach for query expansion that applies dependency rules mined from a large Web crawl, combining several existing techniques for data extraction and mining. Additionally, the rules help binding keyword queries to the input fields of forms on Web portals, opening up the tremendous resources of the Hidden Web. Preliminary, yet convincing experiments from a restricted application domain indicate the viability of our approach.*

1. Introduction

1.1. Motivation

One important aspect needed to improve the accuracy of web search engines is the inclusion of metadata, not only to analyze Web content, but also to interpret and expand user queries. Often regularities that can be exploited are contained in the original data itself. These regularities, e.g. correlations, can be seen as gainful metadata as these regularities can be leveraged for query refinement and disambiguation.

As an example, consider the keyword query 'A4 diesel 1995'. Here, the term 'A4' is ambiguous because it could mean the ISO A4 paper format, the American computer peripherals manufacturer A4, or the 'A4' model of the German carmaker Audi. In the context of the other query terms 'diesel' and '1995' it is evident that the user probably had the latter sense in mind when she posed the query. A search engine could infer this by analyzing the co-occurence of the query terms in a reasonably large corpus, e.g., a large Web crawl, and thereby increase the chance to return relevant pages, i.e., increase precision.

However, the number of relevant pages returned, i.e., recall, cannot benefit from such a disambiguation: As Audi models powered by a diesel engine always have the suffix 'TDI', the term 'diesel' is omitted on many relevant pages. Additionally, the dominating amount of information is not available from static Web pages (the so-called 'surface Web'), but hidden behind forms and dynamically created from raw data stored in large databases (the so-called 'Deep Web' that is much larger than the surface Web). Standard search engines that gather information by crawling the Web cannot index such web pages, as the engine had to fill meaningful inputs in query forms for retrieving such dynamically generated documents. The key difficulty in automatically generating such meaningful inputs is to choose appropriate values for the collection of (typically labelled) form parameters.

To assign the query terms of our example query to a query form on a 'used cars' portal with form fields like make, model, year (of production), and fuel type, a system would need the following information:

- A4 is the model of a car
- diesel is the kind of fuel
- 1995 is the year of production

Therfore a better query would be '(model,A4) (fuel,diesel) (year,1995)' consisting of so-called concept-value pairs (c, v) ([1]). Informally, a concept-value pair corresponds to a property that the query's target should have (like its make being Audi). We refer to such a query as *concept-based query*.

In addition to this enriched query format, we want to automatically expand queries to match with more forms on portals. As an example, some used car portals may not have a form field for 'model', but only for 'make'. If we have collected enough meta information, we may derive that, whenever 'model=A4' occurs in the query, we may add 'make=Audi' without getting too many false hits. We can then feed this expanded query into more forms, getting more (and hopefully better) results than without the expansion.

To automatically transform a keyword query into such a concept-based query and to automatically expand the query, a system needs meta information about car models, makes,

fuel types, etc. and correlations between them. Manually collecting this data for the domain of used cars would be a tremendous amount of work, and doing it for all possible application domains would be infeasible. While the upcoming Semantic Web with its semantically tagged Web sites may be able to solve this problem to a certain extent, it is completely unclear when the ambitioned research prototypes will be carried over to large-scale commercial Web sites.

Until then, the vast majority of Web pages will still be plain HTML without any semantic annotations. Even though some Web sites have gradually moved their content to XML, it is often nonschematic and exposes wide diversity in terms of document structures and tag names.

External ontologies (in contrast to integrated metainformation in the Semantic Web) could help to interpret non-annotated semi-structured information, but they are either too specialized, e.g. in the area of bio informatics, or too broad, like the general-purpose thesaurus WordNet [16]. Additionally, hardly any existing ontology contains instance ('Audi A4 is a car') or property information ('A4 is a car model'), and finding reasonable quantitative similarity measures for related concepts in an ontology is a difficult problem.

Therefore the only way to automatically acquire and maintain meta information is to extract it from existing, non-annotated HTML pages of today's Web. Much work has been spent so far to recognize the intended structure of HTML pages to extract the contained information. We focus on the most structured and therefore most gainful parts of HTML elements, namely tables and forms.

1.2. Related Work

Much effort has been spent in the area of semi-automatic ([2] [3] [4]) or automatic ([5] [6][7]) extraction of structured HTML-elements to deduce the intended semantic structure. In this work we adopted and combined especially the work for automatic classification of tables described in [5] and a generic approach for table recognition, described in [6]. As this is not the main focus of this work but rather a tool, it could be easily replaced by other approaches. We use information obtained by table extraction as a basis for data mining tasks. Therefore our work is highly related to Data Mining in general, especially assocation rule mining ([14]) and to Web Mining, as we use these techniques in a Web based setting([9][8]). Furthermore as we also want to use information contained in HTML forms, this work is related to many works that addressed this problem([10] [11] [12]). Finally query refinement is a goal of this work, that already has been addressed by many papers, escpecially query expansion using WordNet [24, 23, 13, 21]. Although much work has been done in these different areas, we are not aware of works that combine these means in a setting comparable to this work.

1.3. Contributions

In this paper we show how to utilize automatically extracted meta information for

- the *refinement* of queries to improve the result quality in terms of recall an precision, and
- the *transformation* of queries into concept-based queries that can be fed into Web portal forms.

The statistical meta data needed for this process is collected during crawling. Our algorithm for query refinement and form matching is surprisingly simple and efficient. Preliminary experiments show the viability of our approach in practise.

The rest of the paper is organized as follows. Section 2 presents the process of data collection. Section 3 focusses on preprocessing of the data. Section 4 shows how the data is analyzed, and Section 5 presents an algorithm for automatic query refinement. Finally Section 6 presents preliminary experiments.

2. Collecting Metadata

As every document has to pass the crawling and indexing component of a search engine, it is natural to collect meta information as a part of the indexing process. Our current prototype extracts information from HTML tables and forms as they are the most structured elements and widely used constructs on web pages, but it could be easily extended by more sophisticated analysers.

Whenever we encounter a new HTML page, we first convert all HTML tables into an internal representation that is similar to *pseudo-tables* used in [6]. As most tables are used today for layout purposes and do not bear any useful information, our algorithm detects whether the table is purely layout or contains – perhaps semanticlly meaningfull – information. We have adopted the machine learning algorithm presented in [5] that automatically classifies tables as either *non-genuine* (that are used for layout purposes only) or *genuine* (that are semantically significant as they contain structured information). The underlying classifier uses layout, content type and word group features. Following [5], we included the following features into our classifier:

- average number of colums and average number of rows
- standard deviation of the number of rows and columns
- average length of the cells and the correpsonding standard deviation
- average content type consistency (values for an attribute should have the same type)

- cumulative length consistency (values for an attribute should have similar length)

We implemented this classifier as a decision tree and trained it with manually annotated HTML pages that contained genuine and non-genuine tables.

Our algorithm completely ignores non-genuine tables and focuses on genuine tables. For each genuine tables, it first detects its *header*. If special HTML tags for headers (like `<th>`) have been used, this task is easy, but by far the largest fraction of HTML tables does not use them. Therefore, we have to analyze tags and attributes that determine the visual style, as headers are often set in a bold, italic, or otherwise special font. For this purpose we group the cells of the table according to their visual style where a group is formed by all cells with a similar layout. We then assign one of several types of tables to the current table like described in ([7]).

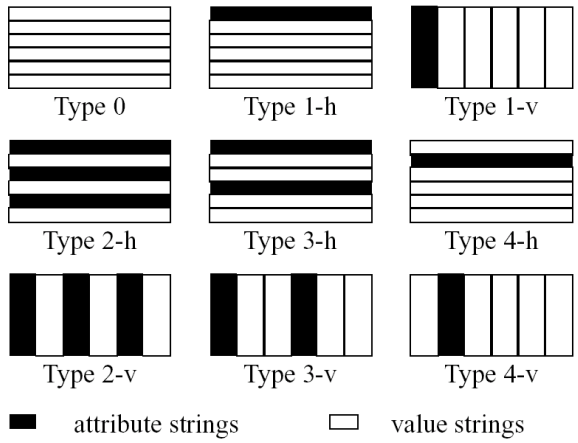

Figure 1. Types of tables

Figure 1 shows different types of tables. If for example the algorithm would generate two groups, one group for the first row and the second group for the remaining cells, type 1-h would be assigned (The suffix 'h' in the figure indicates that the table is horizontally oriented, 'v' indicates that it is vertically oriented). When the analysis process is finished and we have extracted the content of the table, we store it record-wise in our database; it forms one *instance* of information.

In our relational database we maintain a generic table to store the extracted information. Figure 2 shows an excerpt of this table containing some extracted rows (*Reckey* is the number of the record). Logically this table represents a large schema consisting of all different properties; it is extended if new properties are inserted that can not be mapped onto existing properties.

PROPERTY	VALUE	RECKEY
Make:	Audi	17
Model:	TT 1.8	17
Mileage:	5133	17
Transmission:	Manual	17
Trade Name	Aspirin	18
Generic Drug	Acetylslicylic acid	18
Brand:	Audi	19
	TT	19
Fuel:	Diesel	19

Figure 2. Extracted data

Besides tables we also analyze HTML forms and extract the information that they contain. As an example, consider a query form on a Web site of a car dealer (figure 3). From the drop-down boxes, we can extract concept descriptors (like 'make', 'model', 'power') with corresponding values, yielding a collection of possible values for the different concepts (like 'make: Acura,Audi. . .').

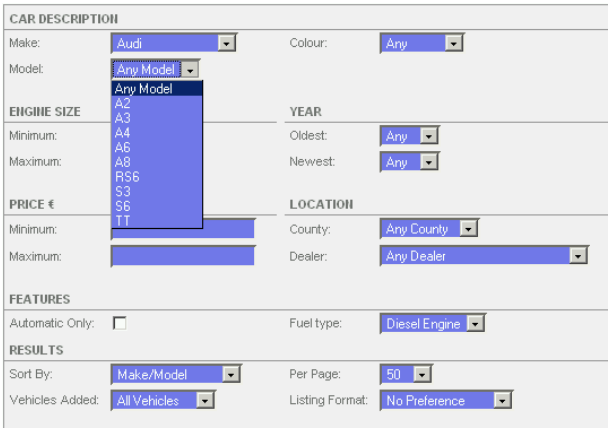

Figure 3. HTML form

For the extraction of HTML forms we use the Web Service Framework described in [19, 20]. It applies heuristic rules to analyze the forms. Typically, highlighted text next to form fields will become concept names and the type of a form field determines the corresponding value type (e.g., an enumeration type for fields with a pull-down menu restricts its valid values). Basically, this tool generates a Web Service with corresponding WSDL description for each form. As this format is easy to analyze we use it to extract the needed information.

We are currently working on extending our system with an XPath-clustering algorithm proposed in [15] that identifies structured information units within pages that are not represented in tables. This is done by grouping XPath expressions with the same prefix. Details of this approach are beyond the scope of this paper.

3. Pre-processing Metadata

Before we can actually analyze the meta data we have collected in the previous step, we have to clean it from any artefacts introduced by errors during collecting. Such errors may occur when extracting the property descriptors (e.g., determining the table headers may fail or they might be missing at all) as well as during the extraction of instance data.

To fill missing property labels, we first consider all instances derived from the same Web site together, assuming that all extracted tables have the same structure and hence all extracted instances have the same set of columns $c_1, ..., c_n$ without label. For each unlabeled column c_i, we compute the distribution p_i of its values and compare it to the distributions q_j of values in all labeled properties $l_1, ..., l_m$ in our complete database, using the well known Kullbach-Leibler divergence of two value distributions:

$$KL(p_i||q_j) = \sum_{x \in c_i} p_i(x) log \frac{p_i(x)}{q_j(x)}$$

If we find at least one candidate for which the Kullbach-Leibler divergence is smaller than a predefined threshold, we label the column with the candidate l_j with the lowest $KL(p_i, q_j)$. Otherwise the unlabeled column is completely removed from the database.

Our algorithm also makes some effort to preprocess values, e.g., by cutting additional characters (like ':', '.', or '"') and normalizing values in different formats (e.g., German vs. American dates) or with different units (e.g., PS vs. kW). However, as this information integration problem is difficult to solve exactly, we limit ourselves to simple heuristics here.

As a result we get a table with a schema comprising all attributes that we have found so far, where each column corresponds to a property. This table contains a lot of columns from many domains, but each single row (which is an instance from a single domain) has non-null entries in only a few of them. For this reason it is possible to determine clusters of rows that share many properties (not values), yielding a separation of rows from different application domains, because instances of completely different domains (like cars and books) typically won't share a lot of properties or maybe no property (like make, model etc. vs. title, author etc.) at all.

PROPERTY	VALUE	RECKEY	P_Bucket	V_Bucket	Partition
make	audi	17	1	17	1
model	tt	17	2	7	1
model	1.8	17	2	6	1
mileage	5133	17	6	113	1
transmission	manual	17	7	1	1
trade name	aspirin	18	21	3	2
generic drug	acetylslicylic acid	18	22	7	2
make	audi	18	1	17	1
model	a6	18	2	2	1
model	tdi	18	2	4	1

Figure 4. Preprocessed Data

Figure 4 shows the database from Figure 2 after preprocessing. The different properties and values are mapped onto numerical values (*V_bucket* and *P_bucket*); the column *Partition* indicates the cluster of the record.

4. Analyzing Metadata

After collecting, preprocessing and splitting the meta information we use data mining techniques to discover relevant relationships among property values.

(CAR)	MAKE	MODEL	POWER	YEAR
I_1	Audi	TDI	103	1993
I_2	-	100 CC	85	1981
I_3	-	TDI	-	-
I_4	-	-	78	1993
I_5	Audi	100	103	1994
I_6	Audi	-	103	1993
I_7	Audi	100 CC	85	1982
I_8	VW	TDI	-	-

Table 1. Extracted table content

Consider the example values shown in table 1 specifying eight extracted instances with their extracted schema (MAKE,MODEL,POWER,YEAR). I_1 and I_2 have almost no similarity by simply comparing the specified values. But if we analyze the collected data, we find out that MODEL 100CC only occurs together with MAKE Audi. Because of this fact we can simply virtually add MAKE Audi to instance I_2, yielding a certain similarity of I_1 and I_2. The same analysis for I_3 shows that the occurence of TDI limits the possible values for MAKE to AUDI and VW. In contrast, year 1993 leads to a lot more possible completions and is therefore not a good candidate to start an expansion with.

From this example it is evident that dependencies between properties exist in this table that are to a certain extent

comparable to functional dependencies in databases. However, functional dependencies are very strong since they have to hold for all possible values. In our case, dependencies may only hold for a subset of values, like the MODEL→MAKE dependency that holds only for MODEL=100CC, but not for MODEL=TDI.

The problem of discovering more general (or "fuzzy") dependencies of two properties has been studied intensively in the literature; among the proposed solutions are mining for multidimensional association rules from association analysis and stochastical means. We first construct, for each pair of properties, their contingency table; a (fictious) example is shown in Table 2 for MAKE and MODEL. We can see from this table that, out of all instances with MODEL=A4, a fraction of 0.97 or 97% had also MAKE=Audi or, if we interpret the values as conditional probabilities, we can say that the probability P(MAKE=AUDI|MODEL=A4) is 0.97. We say that the dependency MODEL=A4→MAKE=Audi has *confidence* 0.97. The *support* of the dependency is defined as the fraction of all MODEL instances whose value is A4. Our algorithm considers only dependencies whose support and confidence are beyond a configurable threshold, and ignores the others.

Property MAKE/MODEL			
(CAR)	A4	100	TDI
$AUDI$	0.97	0.61	0.38
VW	0	0.12	0.47
$OPEL$	0	0	0.02
...	0.03	0.27	0.13

Table 2. Contingency table

As an extension of this, we also consider multidimensional dependencies like MODEL=A4,YEAR=1993→ MAKE=Audi and the corresponding probabilities like P(MAKE=AUDI | MODEL=A4 ∧ YEAR=1993).

Note that we can also use association rule mining to find dependencies; the notions of support and confidence defined here are the same as in association rule mining.

5. An Algorithm for Query Expansion

In this section we present a simple, yet effective new algorithm for query expansion using mined association rules. First we map a keyword query q with keywords $t_1 \dots t_m$ onto a set of concept-value-pairs $\{(c_1, t_1)...(c_m, t_m)\}$. As multiple assignments for a query may be possible, we choose the assignment with the highest probability satisfying the constraint that all assigned properties belong to same cluster in the meta database. Here, the probability that term t_i is mapped to concept c_j is defined as

$$P[c_j \mid t_i] = \frac{\#\text{occurences of } t_i \text{ in } c_j}{\#\text{occurences of } t_i \text{ in } c_i\text{'s cluster}}$$

and we multiply the probabilities for all term mappings. Without the "same cluster" constraint, terms could be assigned to concepts from different real-world objects (e.g., the assignment MAKE=Audi AND PAPER_FORMAT=A4 could be generated, which does not make sense).

If no assignment for a term is found or if the best probability for the term is below a predefined threshold, it remains unassigned and is ignored in the remaining algorithm. As an example, the keyword query 'Audi 1995 diesel navigation' would be mapped onto the concept-based query ((make,Audi)(year,1995)(fuel,diesel)(,navi)).

In a next step we build all possible subsets of our concept-based query $((c_1, t_1)...(c_m, t_m))$ with non-empty concepts. In the example shown above this would be S={{(make,audi)(year,1995)},{(make,audi)(fuel,diesel)}, {(fuel,diesel),(year,1995)}. Now we try to find matching dependencies discovered in the previous phase.

Obviously we can use dependencies in two ways to expand our query:

- A rule whose antecendent is part of the base query can be used to infer additional query attributes that are part of the rule's consequent (type A).
- A rule whose consequent is part of the base query can be used to infer additional query attributes that could have deduced query attributes in the base query (type B).

For our example query 'Audi 1995 diesel navigation', the following rules are retrieved:
R1: (make,audi)(model,TDI)→(fuel,diesel)
R2: (make,audi)(kw,103)(year,1995)→(fuel,diesel)
R3: (make,audi)(year,1998)(model,A6)→(fuel,diesel)

We order these rules by confidence and support and only select rules above certain thresholds $c\_\min$ and $s\_\min$. For each such rule we build a new query: For rules of type A we replace the concept-value pairs of the query that match the antecendent by the concept-value pairs of the its consequence. For rules of type B we analogously replace concept-value pairs matching the consequence by its antecedent. Note that this can lead to queries that are not satifiable; we refer to this as a *conflict*.

For our example the following generated subqueries contain a conflict in Q'3:
Q'1: ((make,audi)(year,1995)(model,TDI)(,navigation))
Q'2: ((make,audi)(year,1995)(kw,103)(,navigation))
Q'3: ((make,audi)(year,1995)(year,1998)(model,A6) (,navigation))

The usage of rule 3 generates Query Q'3, that contains the query conditions (year,1995) and (year,1998). As both conditons can not be fullfilled at the same time we drop Q'3 that contains this conflict.

Note that the same attribute occuring several times is not necessarily a conflict, as some attributes can occur multiple times in the same object. For example, an instance of car could have several features like (feature,ABS) and (feature,ESP), so having several feature conditions in a single query does not mean a conflict. To distinguish these two cases, we mark attributes as "unique" that occure only once per instance in our database. We then define a conflict as an expansion of a query that contains more than one value condition for a unique attribute.

Having retrieved all non-conflicting subqueries we build the refined query Q' that is the disjunction of Q and Q'1,...Q'n. In the example, we get the following query Q':

Q'=((make,audi)(year,1995)(fuel,diesel)(,navigation))
OR ((make,audi)(year,1995)(model,TDI)(,navigation))
OR ((make,audi)(year,1995)(kw,103)(,navigation))

To use this query for an HTML form submission we try to assign each concept-value pair to a form element. Of course some conditions like 'navigation' or (model,TDI) can not be assigned as most query interfaces do not offer these choices. The form shown in Figure 3 would only accept an assignment (make,audi)(year,1995)(fuel,diesel).

To use this refined query for simple keyword search we drop the concepts again. Even though we are back to plain keyword queries now, the introduction of concepts was necessary for the expansion steps in our algorithm. In the example, we yield the following keyword query:

Q"=audi AND 1995 AND diesel AND navigation OR
audi AND 1995 AND TDI AND navigation OR
audi AND 1995 AND 103 AND navigation

6. Experiments

6.1. Setup

The system was run on a dedicated PC (Dual-Intel 3 GHz, 2 GB RAM) running Windows 2003 Server. Our software is implemented in Java (1.4.1) using the WEKA 3 Library [25] for data mining. The data was stored in an Oracle 10g database running on the same machine. For collecting our source data we used the BINGO! focussed Crawler [26].

6.2. Query refinement

We made preliminary experiments on the two application domains *preowned car advertisments* and *drugs*. More comprehensive experiments are subject of future work. We split each of our domain into a training and a testing set. On the training sets we tried to discover asscocation rules [1].

	Cars	drugs
Pages	31250	121
Records	37672	484
Rules	7841	311

Table 3. Data Sets

Table 3 shows the number of pages that that contained *genuine* tables (according to our classifier) and have been used for extraction. The number of pages of the drug domain is relatively small as most pages in this domain do not contain any HTML table or the contained tables could not be automatically extracted.

Pos.	Antecendent	Consequent	conf.	sup.
1	make=audi and kw=96 and model=1.9	model=TDI	1.0	0.060
2	make=audi and fuel=diesel and kw=96 and model=1.9	model=TDI	1.0	0.059
...	...	...	...	...
17	make=ferrari and model=360 and seats=2	doors=2	1.0	0.011
1	trade name= aspirin	generic drug= Acetosalic Acid	1.0	0.008
...	...	...	...	...
4	trade name=ASS ratiopharm	generic drug= Acetosalic Acid	1.0	0.004

Table 4. Generated Rules

Table 4 shows some rules with the highest confidence values. We generated 10 example queries for each do-

1 We conducted our experiments on german-language web pages and translated the terminology

main. For the following queries Q1 and Q2 we show the steps of our algorithm in detail.
Q1 (domain car ads): Audi A6 diesel
Q2 (domain drugs): Aspirin
The queries were mapped onto the following concept-value pairs:
Q1': Make=Audi AND Model=A6 AND fuel=diesel
Q2': Trade Name=Aspirin

In the next step we retrieved matching rules according to our algorithm, ordered by confidence. The threshold for confidence was set to c=0.5; the threshold for support was set to s=0.001. Furthermore we only retrieved up to 10 rules per query. The following rules were retrieved for Q1':
R1_1: model=TDI ⇒ fuel=diesel
R1_2: engine=1.9 AND KW=96 AND model=A6 ⇒ fuel=diesel
(rules R1_3...R1_6 are very similar to R1_2; they only differ in different values for *engine* and *KW*.)

The following rule was retrieved for Q2':
R2_1 Trade Name= Aspirin ⇒ Generic Drug Name= Acetylsalicylic acid

For Q1, 4 of the selected best rules were skipped as they caused conflicts. For query Q2 only one rule was retrieved. The following expanded queries for Q1' and Q2' have been generated.

Q1": (Make=Audi AND Model=A6 AND fuel=diesel)
OR (Make=Audi AND engine=1.9 AND KW=96)
OR (Make=Audi AND Model=A6 AND Model=TDI)
OR ...

Q2":(Trade Name = Aspirin) OR (Generic Drug Name= Acetylsalicylic acid)

For all 20 queries we measured precision among the first 10 results and recall. To determine the recall we intensely analyzed our source data.

Figure 5 shows recall and precision of the queries Q1 and Q2 in comparison to the expanded queries Q1" and Q2". Altough the precision of Q1" is slightly worse compared to the original query Q1, the expanded queries outperformed the original queries. Using Rule R2_1 we not only get pages about Aspirin but also about *ASS Ratiopharm* that is a drug of the same composition as Aspirin [2].

Figure 6 shows the improvements concerning recall and precision of the refined queries (marked with "). Again the refined queries outperform the original queries in terms of recall and precision

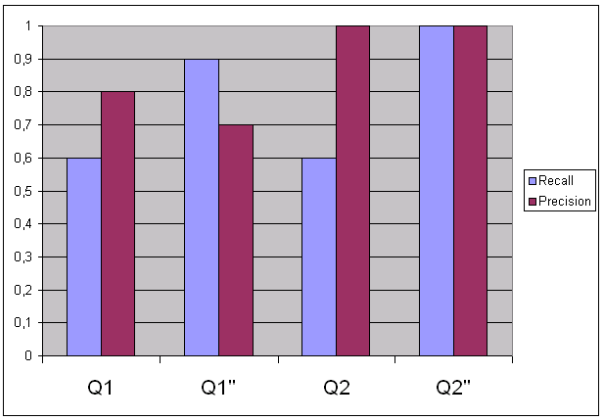

Figure 5. Recall and Precision 1

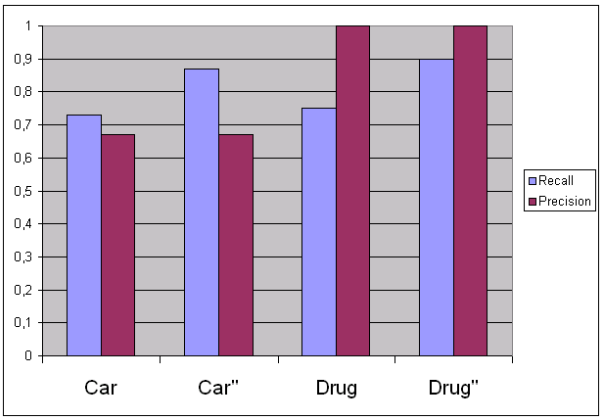

Figure 6. Recall and Precision 2

7. Conclusion and Future Work

This paper has successfully shown that information extracted from Web pages can be used for effective query expansion. Using data mining methods, our system derives dependency rules and applies them to concept-based queries. Even though we presented only preliminary experiments from two simple application domains, the results are promising and pave the way for future research in this area. We plan to integrate the system presented in this paper into COMPASS, our prototype implementation of a concept-based search engine [1]. Additionally, we want to make large-scale experiments with data from different application domains to prove that our approach can be applied generally.

2 This drug is only available in Germany

References

[1] J. Graupmann, M. Biwer, C. Zimmer, P. Zimmer, M. Bender, M. Theobald, G. Weikum. COMPASS: A Concept-based Web Search Engine for HTML, XML, and Deep Web Data. VLDB 2004, pages 1313–1316, 2004.

[2] R. Baumgartner, S. Flesca, G. Gottlob: Visual Web Information Extraction with Lixto, VLDB 2001

[3] V.Crescenzi, G. Mecca, P. Merialdo: RoadRunner: Automatic Data Extraction from Data-Intensive Web Sites, SIGMOD Conference 2002

[4] F. Ciravegna, A. Dingli, D. Guthrie, and Y. Wilks. Integrating information to bootstrap information extraction from web sites. In Proceedings of IJCAI-2003, 2003.

[5] Y .Wang, J. Hu. A machine learning based approach for table detection on the web. In *WWW 2002*, pages: 242 - 250, 2002.

[6] S.J. Lim, Y.-K. Ng. An automated approach for retrieving hierarchical data from HTML tables. In *CIKM 1999*, pages: 466 - 474, 1999.

[7] M. Yoshida, K. Torisawa and J. Tsujii. A method to integrate tables of the World Wide Web In Proceedings of the International Workshop on Web Document Analysis (WDA 2001), pages 31-34

[8] S. Brin. Extracting patterns and relations from the World-Wide Web. In *WebDB 1999*, 1999

[9] P. D. Turney. Mining the Web for synonyms: PMI-IR versus LSA on TOEFL. In Proceedings of the Twelfth European Conference on Machine Learning, 2001.

[10] K. C.-C. Chang, B. He, and Z. Zhang MetaQuerier over the Deep Web: Shallow Integration across Holistic Source, VLDB-IIWeb 2004

[11] W. Wu, C. Yu, A. Doan, W. Meng: An Interactive Clustering-based Approach to Integrating Source Query interfaces on the Deep Web. SIGMOD Conference 2004

[12] J. Qiu, F. Shao, M. Zatsman, J. Shanmugasundaram: Index Structures for Querying the Deep Web. WebDB 2003

[13] G. Amati, C. Carpineto, and G. Romano. Fondazione Ugo Bordoni at TREC 2003: Robust and Web Track. In *TREC 2003*, pages 234–245, 2003.

[14] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *ACM Sigmod Conference*, 1997.

[15] H. Davulcu, S. Vadrevu, and S. Nagarajan. OntoMiner: Bootstrapping and populating ontologies from domain specific Web sites. In *First Workshop on Semantic Web and Databases (SWDB)*, 2003.

[16] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[17] H. He, W. Meng, C. T. Yu, and Z. Wu. WISE-Integrator: An automatic integrator of Web search interfaces for e-Commerce. In *28th Conference on Very Large Data Bases (VLDB)*, 2003.

[18] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2), 1999.

[19] J. Graupmann and G. Weikum. The Role of Web Services in Information Search. *IEEE Data Engineering Bulletin*, 25(4), pages 60-65, 2002.

[20] J. Graupmann, S. Sizov, and M. Theobald. From focused crawling to expert information: an application framework for web exploration and portal generation (Demo). VLDB 2003, pages 1105–1108, 2003.

[21] S. Liu et al. An effective approach to document retrieval via utilizing WordNet and recognizing phrases. In *SIGIR 2004*, pages 266–272, 2004.

[22] S. Raghavan and H. Garcia-Molina. Crawling the hidden Web. In *26th Conference on Very Large Data Bases (VLDB)*, 2001.

[23] A. Theobald and G. Weikum. Adding relevance to XML. In *WebDB 2000*, pages 105–124, 2001.

[24] E. Voorhees. Query expansion using lexical-semantic relations. In *SIGIR 1994*, 1994.

[25] Weka 3: Data Mining Software in Java. http://www.cs.waikato.ac.nz/ml/weka/

[26] S. Sizov, M. Biwer, J. Graupmann, S. Siersdorfer, M. Theobald, G. Weikum, P. Zimmer: The BINGO! System for Information Portal Generation and Expert Web Search. CIDR 2003