

# Self-Supervised Web Search for Any-k Complete Tuples

Alexander Löser, Christoph Nagel, Stephan Pieper, Christoph Boden

University of Technology Berlin

Einsteinufer 17

10587 Berlin

[firstname.lastname]@campus.tu-berlin.de

## ABSTRACT

A common task of Web users is querying structured information from Web pages. In this paper we propose a novel query processor for systematically discovering any-k relations from Web search results with conjunctive queries. The ‘any-k’ phrase denotes that retrieved tuples are not ranked by the system.

For realizing this interesting scenario the query processor transfers a structured query into keyword queries that are submitted to a search engine, forwards search results to relation extractors, and then combines relations into result tuples.

Unfortunately, relation extractors may fail to return a relation for a result tuple. We propose a solid information theory-based approach for retrieving missing attribute values of partially retrieved relations. Moreover, user-defined data sources may not return at least k complete result tuples. To solve this problem, we extend the Eddy query processing mechanism [14] for our ‘querying the Web’ scenario with a continuous, adaptive routing model. The model determines the most promising next incomplete row for returning any-k complete result tuples at any point during the query execution process.

We report a thorough experimental evaluation over multiple relation extractors. Our experiments demonstrate that our query processor returns complete result tuples while processing only very few Web pages.

## Categories and Subject Descriptors

H.2.4 [Database Management Systems]: *Text Analytics*

## General Terms

Algorithms, Performance, Experimentation and theory

## Keywords

Text analytics, Join execution on web search results

## 1. INTRODUCTION

A typical task of a Web user is discovering relations between entities from Web pages; i.e. consider the following query:

*Extract and join relations between products, acquisitions and employee sizes of companies from top 100 pages of CNN.com as seed data into result tuples. Retrieve complementary pages from Yahoo.com until 500 complete result tuples exist.*

This query might be issued by an analyst who is observing the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

BEWEB 2011, March 25, 2011, Uppsala, Sweden.

Copyright 2011 ACM 978-1-4503-0610-2/11/03 ...\$10.00

<b>Joining 2 Relationships (Information about Persons)</b>	
2-06:	Attribute( <u>Person</u> ,Born,Gender) ⋈ Career( <u>Person</u> ,Company, Position); <i>washingtonpost.com(100); yahoo(10)</i>
2-07:	Conviction( <u>Person</u> , Charge) ⋈ Career( <u>Person</u> ,Company, Position); <i>cnn.com(100); yahoo(10)</i>
<b>Joining &gt; 2 Relationships (Information about Companies and Persons)</b>	
3-01:	Product( <u>Company</u> ,Product) ⋈ Career( <u>Person</u> , <u>Company</u> , Position) ⋈ Attribute( <u>Person</u> ,B_Place, B_Date); <i>money.cnn.com(100); yahoo(10)</i>
3-08:	Recommendation(Analyst, <u>Company</u> ,Trend,Date) ⋈ Bankruptcy ( <u>Company</u> , Status, Date) ⋈ Product( <u>Company</u> ,Product); <i>ft.com(100); yahoo(10)</i>
3-09:	HQ( <u>Company</u> , Location) ⋈ Employees ( <u>Company</u> , EmployeeSize) ⋈ Product( <u>Company</u> ,Product); <i>ft.com(100); yahoo(10)</i>
4-01:	HQ( <u>Company</u> ,Location) ⋈ Employees( <u>Company</u> ,EmployeeSize) ⋈ Product( <u>Company</u> , Product) ⋈ Bankruptcy( <u>Company</u> ,Status,Date); <i>cnn.com(100); yahoo(10)</i>
5-01:	Born( <u>Person</u> ,Born) ⋈ Conviction( <u>Person</u> ,Conviction) ⋈ Employer( <u>Person</u> ,Company) ⋈ Travel( <u>Person</u> ,Location) ⋈ Career( <u>Person</u> ,Company, Position); <i>nytimes.com(100); yahoo(10)</i>

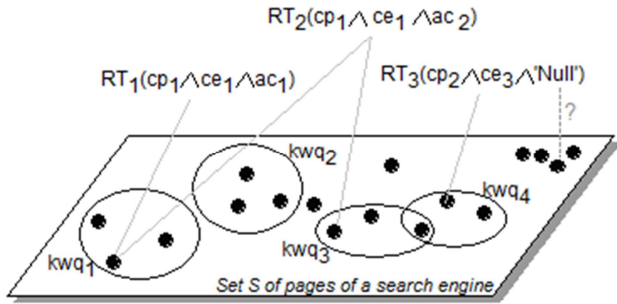
Figure 1: Queries joining tuples from two or more relation extractors. Join attributes are underlined.

market for interesting acquisition options. The query intention is retrieving *at least 500 tuples* about any company, its products, its headquarters, and its employee information from Web pages.

**Discovering any-k tuples through Web search.** Discovering such result tuples with structured queries on Web pages is a grand challenge. For instance, current Web search engines are not capable of extracting relations from search results. Due to the nature of the business models of commercial Web search engines, it is unlikely that full access to internal databases will ever be granted to individual users. Typically only the top 1000 results for a given search query can be retrieved. Therefore a common practice of a Web user is typing in some more or less arbitrarily chosen keywords as queries, reading some of the top ranked result pages and copying text phrases that potentially represent relevant tuples into a spreadsheet. We call this explorative process discovering any-k tuples, where tuples are equivalent to rows in the spreadsheet.

Our system automatically fulfills the task of retrieving a set of any-k complete result tuples. This set may then be forwarded directly to the user or may be forwarded to an application for specific filtering [9][31] or ranking [2] [30] approaches.

Query 3-09 in Figure 1 represents a structured query that expresses these requirements for the user query from above. The query creates a table listing company employee information, their products and their headquarter locations. More formally speaking, we want to generate k complete result tuples that consist of relations (or tuples) that are extracted from Web pages by a relation extractor and joined based on similar attribute values



**Figure 2:** Consider four keyword queries  $kwq_1 \dots kwq_4$  that retrieve tuples for answering the example query Q 3-01. Each keyword query returns a set of three result pages; these pages are forwarded to an extractor that returns tuples  $cp \in CP$ ,  $ce \in CE$  and  $ac \in AC$ . For instance,  $kwq_1$  returns three pages from which the query processor extracts tuples  $cp_1$ ,  $ce_1$ ,  $ac_1$  and combines them into result tuple  $RT_1$ . Query  $kwq_2$  does not return any relevant pages. Then, the query processor issues query  $kwq_3$ , extracts tuple  $ac_2$  from retrieved pages and combines tuple  $ac_2$  and previously retrieved tuples  $cp_1$ ,  $ce_1$  into result tuple  $RT_2$ . Query  $kwq_4$  returns tuple  $cp_2$  and  $ce_3$  which are combined by the query processor into incomplete result tuple  $RT_3$ .

(underlined in Figure 1). The table is populated with initial result tuples that are extracted from top-100 pages of the news site *ft.com*. The query processor may query the Web search engine Yahoo as a complementary source for retrieving missing attribute values. Thereby, the query processor makes predictions on the likelihood that specific pieces of missing information from incomplete tuples can be found at the complementary Web source and, based on that, optimizes the sequence of Web requests that need to be made in order to obtain  $k$  complete result tuples. This process is adaptive.

**Our Contributions:** The long term vision of our work is to answer structured queries from natural language text on Web pages, called Web text [13][22][23][24]. For realizing this novel application scenario we give three core contributions:

(1) *End-to-end Web query processor:* We propose a *query processor* to systematically discover any- $k$  complete result tuples from Web pages. In this paper we focus on select-project-(full outer) join queries that integrate tuples from two or more relation extractors. Our query processor leverages the index of a Web search engine for retrieving tuples from potentially billions of pages.

(2) *Solid theoretical framework ensures any- $k$  complete tuples:* We view completeness as a major measure of data quality and thus aim to return any- $k$  complete result tuples. For solving this very practical problem we propose a solid information theory-based approach for identifying the most promising source for retrieving missing attribute values of partially retrieved tuples.

(3) *Novel extension to the Eddy operator for Web querying.* We design our model as a novel extension of the Eddy operator for the Web querying domain. An Eddy is a dataflow operator in traditional a database system that allows an adaptive tree reordering in a tuple by tuple fashion. Our extension implements a continuous, adaptive routing model for determining the most promising next Web source to query at any point during the execution. We implement our query processor for the in-memory database *HSQLDB* [19]. In a broad study we test our theoretical findings on multiple relation extractors and identify optimization

strategies that effectively reduce the amount of processed Web pages.

**Overview:** In the following sections we outline our query processing algorithms for any- $k$  complete tuples in detail (Section 2). Then we report on the performance of our algorithms (Section 3) and discuss related work (Section 4). Finally, we summarize our contributions and propose our future work (Section 5).

## 2. QUERY PROCESSOR

In this section, we present a framework for the study of the tuple discovering problem. In Section 2.1, we abstract important problems for automating the discovery of complete tuples on the Web. Based on this interaction model, we present a generic query routing algorithm for an adaptive query processor in Section 2.2. Finally, in Sections 2.3-2.5, we discuss routing policies that can ensure complete result tuples and significantly reduce query processing costs.

### 2.1 Problem analysis

Theoretically, the problem of discovering any- $k$  result tuples on Web search results can be formalized as follows: We assume a query processor is downloading a subset of pages from a set of pages  $S$  indexed by the search engine (see the rectangle in Figure 2). We represent each Web page in  $S$  as a point (dots in Figure 2). Every potential query  $kwq_i$  from the query processor returns a subset of  $S$ . From each subset of  $S$  we may be able to extract none, one or multiple relations. The query processor joins these relations from one or multiple subsets into result tuples  $RT_k$ . Unfortunately; some result tuples may be missing certain attribute values that could not be extracted (denoted by ‘NULL’). However, the set of pages  $S$  may contain additional pages (denoted by a dotted line) from which the query processor could extract these missing values.

**Ordered sequence of keyword queries:** Our goal is to find an ordered sequence of keyword-queries which likely return pages from which the query processor can discover any- $k$  result tuples. Extracting relations from pages is costly and time consuming. Therefore we would like to minimize the cost by processing as few pages as possible.

**NP-complete problem:** Determining an ordered sequence of keyword queries that minimizes the costs for discovering any- $k$  result tuples is equivalent to the problem of answering a query against a schema using a set of views. This problem is NP-complete for conjunctive queries and conjunctive view definitions [17].

How should a query processor determine an ordered set of keyword queries? In practice, we face the following difficulties that we need to address to solve the formalization from above:

**Continuous, adaptive routing model:** Query optimizers in current database systems are designed to pick a single efficient execution plan for a given query based on statistical properties of the data. However, in our setting the query processor transforms a structured query into a set of keyword queries to extract an initial set of result tuples. Next the query processor chooses an incomplete result tuple and generates a new keyword query to retrieve the missing attribute values. This adaptive process of generating keyword queries, extracting tuples and joining them into result tuples must be continued by the query processor until at least  $k$  result tuples are complete.

**Estimate how likely a keyword query returns a missing attribute value:** For optimizing query execution costs the query processor must estimate how likely a keyword query and an extractor will return a missing attribute value for a particular result tuple. The probability that a keyword query returns a missing attribute value is correlated with the search engine rank of the source Web page, the content of that particular page, the overall set of indexed pages of the search engine and the mechanics of the extractor.

- *Biased search engine ranking.* The ranking mechanism of a search engine can deny a page from appearing within the top pages [4]. Furthermore, search engines like Yahoo or Google limit the access to their database to top-1000 pages per keyword query.
- *Insufficient content.* There may be relations for which no textual content exists on the Web. Consider a query joining two relations of type CompanyProduct and Bankruptcy. In reality, every company has at least one product, but only a few have declared bankruptcy, thus these particular result tuples can never be completed.
- *Failing extractors.* Relation extractors can return incomplete tuples. The rationale for this incompleteness is to indicate the existence of a relation on a Web page to human evaluators, even though precise values for each attribute of the relation could not be extracted. For instance, relation extractors may fail because of a *vocabulary mismatch* between the Web page and the rules of the extractor. Object reconciliation [12] may fail if not enough context information for resolving objects exist.

In the next Sections we present our solution for each requirement.

## 2.2 Generic routing algorithm

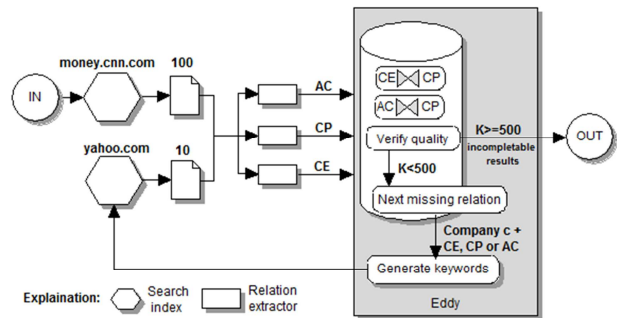
Our generic routing algorithm is implemented as an extension to an Eddy [14] which can potentially adapt at the granularity of a missing relation. An Eddy processes a query by routing relations through operators that are specific to that query. However, an Eddy does not actively request missing relations to meet user defined quality requirements, such as returning complete result tuples. Therefore we extend Eddies with a technique called *quality based routing (QBR)* that adds quality requirements into routing decisions. With QBR an Eddy automatically identifies tuples of insufficient quality and generates additional queries for obtaining missing relations. QBR is low-overhead and it is adaptive, revisiting its decisions as changes in result tuples are detected.

The following paragraphs describe this continuous process.

**Obtain initial set of pages.** First, the query processor bootstraps top-k pages from the data source that is specified in the INITIAL\_DATASOURCE clause in the query. If the data source does not return any page, query processing is terminated. Otherwise the query processor proceeds with step 1.

**Step 1: Generate result tuples.** In this step the query processor forwards pages to relation extractors and combines new tuples with potentially existing result tuples.

- *Extract tuples from pages.* The query processor forwards pages to relation extractors. For each page none, one or multiple tuples are extracted.



**Figure 3:** We illustrate the quality based routing policy for query 3-01. First, 100 initial pages are retrieved from money.cnn.com and relation extractors AC, CP and CE return relations from these pages. The Eddy joins relations and determines if 500 complete result tuples exist. Next, the Eddy generates a keyword query for retrieving a missing relation. The keyword query retrieves 10 pages from complementary search engine 'yahoo.com'. The Eddy continues this process until 500 or more result tuple are retrieved or of no more result tuple can be completed.

- *Combine new tuples with existing result tuples.* The query processor enables n-way full outer joins for combining new tuples into result tuples. If k is relatively small (such as in our scenario) this operation can be executed in memory. For relatively large result tuple sets (such as millions of result tuples), the join-operation can be optimized with existing adaptive query processing techniques, such as progressive optimizations [1] or lottery scheduling [14].
- *Verify quality and terminate query if necessary.* The full outer join may create new result tuples or can add new attribute values to existing result tuples. Therefore the query routing policy monitors the quality of result tuples. The query processing is terminated either if no more result tuples can be completed or if any-k result tuples meet the required quality requirements (see Section 2.5).

Otherwise, the query processor proceeds with step 2.

**Step 2: Retrieve missing attribute values.** In this step a single missing attribute value is identified. The query processor generates an additional keyword query for obtaining additional pages that likely contain the missing values.

- *Identify next missing attribute value.* The query routing policy identifies a candidate result tuple that does not meet the quality requirements of the query. From the candidate result tuple a missing attribute value is identified that can be likely obtained. (See Section 2.3-2.7).
- *Generate keyword query.* In [24] we proposed a keyword generation strategy. First we observe if sentences sharing instances of the same relation also are likely to share similarities in textual content. Next, we only keep phrases from these sentences that indicate the existence of a relation using an efficient lexico-syntactic classifier [10]. Finally, we determine for each phrase their clarity with respect to other relations and their significance for a particular one [20]. For instance, for retrieving missing attributes B\_PLACE and B\_DATE for the tuple ATTRIBUTE('Martin Winterkorn', Null, Null) the method presented in [24] generates the keyword query + 'Martin Winterkorn' + ('born' OR 'born in' OR 'born at' OR 'native of').

- The query processor executes the keyword query against the search engine from the COMPLEMENTARY\_DATASOURCE clause and proceeds with step 1 again.

In the remainder of this Section we propose multiple routing policies to customize our generic routing algorithm.

### 2.3 Baseline: Zig-Zag routing policy

The Zig-Zag routing policy [5] selects a random result tuple that misses an attribute value. Despite its simplicity, a Zig-Zag routing policy can cause relatively high costs. For instance, a Zig-Zag routing policy is unaware of the quality of result tuples. Therefore this policy may process result tuples that only achieved a low degree of quality and at the same time may ignore result tuples for which it would be rather easy to complete them. Hence, this policy frequently picks a result tuple that cannot be completed with additional information from the Web. Finally, this policy can create a death spiral by repeatedly selecting and creating new low quality result tuples.

Therefore we consider the Zig-Zag routing policy as a baseline scenario since it is expected to perform suboptimally.

### 2.4 Choosing ‘good’ join attributes

In this Section we propose our novel approach for estimating the likelihood of returning a missing tuple. Our idea is to compute the information gain ratio (IGR) for each combination of a relationship-type and a join predicate that can be potentially used by the Eddy routing algorithm. The IGR gives an estimate whether a particular combination will return a relation or not.

**Identifying join attributes:** The query processor explores correlations between values of an attribute  $A_E$  from which a keyword query is generated and values of an attribute  $A_1, \dots, A_n$  of a relation  $R_E(A_1, \dots, A_n)$  that is returned from extractor E. In our setting the observed correlation between two attribute value distributions might not reflect the ‘real’ distribution that could be observed on a much larger sample of Web pages. We reflect this uncertainty with a specification from Information Theory which is based on the concept of *gain ratio* [28], described next. The IGR is typically used in decision-tree learning algorithms (such as ID3 [28]) to determine the attribute that best classifies a given data set. We utilize the IGR for identifying the most promising join attribute and the most promising relationship-type for retrieving missing tuples.

**Definition 1 (Entropy):** Let  $c_1 \dots c_p \in D.C$  be instances of attribute  $C$  of relationship-type  $D$  where the attribute values  $c$  are different from NULL. We use the entropy [28] of  $D.C$ , denoted by  $H(D.C)$ , which is an information-theoretic metric, to capture the information content of  $D.C$ .  $H(D.C)$  is defined as:

$$H(D.C) = - \sum_{c \in c_1 \dots c_p} \frac{c}{|D.C|} * \log_2 \frac{c}{|D.C|}$$

Where  $c_1 \dots c_p \in D.C$  are part of previously extracted instances of the relationship-type  $D.C$ . These instances were extracted using  $n$  keyword queries created from the attribute values  $b_1 \dots b_n \in D.B$

**Definition 2 (Gain ratio):** Let  $D.B$  be an attribute that is used to generate keywords for retrieving additional pages from the search engine. Let  $b_1 \dots b_n$  be the distinct values of  $D.B$ . Following [28], we define as Information Gain the increase in information about

pages containing the attribute  $D.C$ , gained by the additional knowledge about values of attribute  $D.C$ :

$$IG(D.C, D.B) = H(D.C) - \sum_{b \in b_1 \dots b_n} \frac{|D.C_b|}{|D.C|} * H(D.C_b)$$

Following [28], we normalize the information gain to get the *Gain ratio*.

$$Gain\ ratio(D.C, D.B) = \frac{IG(D.C, D.B)}{H(D.B)}$$

### 2.5 Completeness-based routing policy

**Overview:** To overcome the shortcomings of the Zig-Zag policy, we propose a new greedy routing policy called *completeness based routing policy (CBRP)*. A CBRP returns any-k complete result tuples. When a CBRP is chosen as routing policy, only those result tuples which already achieved a high degree of completeness are chosen for further processing. For each of these result tuples, the *likelihood* of retrieving a missing attribute is estimated with the *Gain ratio* and incorporated into the selection decision. Finally, keyword queries are only generated for result tuples that *de facto* can be completed. Thereby, fewer pages need to be processed by relation extractors and the overall system throughput is significantly higher compared to that of a Zig-Zag policy. We implement these requirements as follows:

**Step 1: Avoid result tuples that cannot be completed:** A keyword query may return irrelevant pages on which an extractor cannot spot a missing relation for a result tuple. In that case, we assume that the missing relation is either not available on the Web or cannot be captured by our system. Therefore, we label the result tuple as incompletable and exclude it from the query routing process. If a subsequent keyword query happens to return a Web page from which a missing relation for the result tuple in question can be successfully extracted, we retract the label and include the result tuple into the routing process again.

**Step 2: Maximize completeness:** If a result tuple already achieved a high degree of completeness only a few missing attribute values have to be retrieved from the Web to complete it. Therefore our routing policy prefers result tuples that have achieved a maximum degree of completeness. We define result tuple completeness with respect to a query  $Q$  as follows:

**Definition 3 (Completeness):** Consider a set of attributes  $A_Q$  of a query  $Q$ . Function *fullness(a)* returns 0 if attribute  $a \in A_Q$  contains a NULL-value (if no value has been extracted for an attribute) and otherwise returns 1. We define ‘*completeness<sub>Q</sub>*’ of a result tuple  $t$  as the average fullness over all attributes  $A_Q$  of a query  $Q$ .

$$completeness_Q(t) := \frac{1}{|A_Q|} \sum_{a \in A_Q} fullness(a)$$

If a result tuple  $t$  reaches a *completeness<sub>Q</sub>(t)=1*, the result tuple is ‘complete’, otherwise the result tuple is ‘incomplete’.

**Step 3: Maximize Gain ratio:** Multiple incomplete result tuples may achieve the same maximum degree of completeness. For these candidate result tuples the query processor enumerates combinations of join attributes and relation extractors. Then, the query processor selects the combination which achieves the maximum Gain ratio to compute the next keyword query for

retrieving a missing relation. The maximum gain ratio is selected, because it indicates the combination of a relation extractor and join attribute that most likely returns a missing relation.

**Step 4: Continuously update classifier:** Each time new pages are returned the query processor updates the Gain ratio. The query processor takes existing result tuples as training data and re-computes the Gain ratio for all combinations of join predicates and relation extractors. The *Gain ratio* for a particular combination of join attributes and relation extractors is high, when the keyword queries computed from the join attribute values return pages from which a relation extractor can easily extract missing attribute values for a relation. Optimizing this process is subject for our future work.

### 3. EXPERIMENTAL EVALUATION

We designed a prototype system for discovering any-k complete tuples. In this Section we report our results.

#### 3.1 Evaluation setup, queries and metrics

**Benchmark queries:** Currently no benchmark for discovering any-k complete tuples from Web pages exists. Therefore we conducted experiments on seven queries shown in Figure 1.

Our benchmark is tailored towards discovering any-k complete tuples from search engines with a special focus on news Web sites. We expect that Web users frequently visit these pages to discover previously unseen connections between companies and persons. Therefore our queries simulate the human process of reading news pages, discovering initial result tuples from news pages and complementing missing tuples from additional pages. Each query obtains 100 initial pages from various news search engines, such as *cnn.com*, *ft.com*, *washingtonpost.com* or *nytimes.com*. We expect that news Web pages cover only a fraction of the information that is published on the Web. Therefore, each query complements missing tuples from the index of the *Yahoo search service* [15].

The benchmark queries utilize 11 relation extractors with 14 different attribute types from the extraction service *OpenCalais.com* [16]. Queries vary in the number of relation extractors, the number of attributes and in join predicates.

**Measurements:** Our goal is reducing the number of processed pages. For each query we measure the average number of processed pages to retrieve a single complete result tuple (*P/RT*). In addition, for each query we measure the number of complete tuples (*RT*) for processing 5000 pages. We also measure the proportion of the size of any tuples (*Buf*) vs. complete result tuples that answer the query (*%RT/Buf*). Finally, we compared the speedup of each strategy to our base line.

- Q-ID** : Query ID
- #J** : Joins per query
- #A** : Distinct attributes per query
- k** : Expected complete result tuples per query (k=500)
- RT** : De facto returned complete result tuples per query
- Buf** : Number of complete and incomplete tuples in buffer
- %RT/Buf**: Percentage of complete tuples in buffer
- P/Q** : Processed pages per query
- P/RT** : Processed pages per single complete result tuple
- Speedup** : P/RT (Baseline) / P/RT (competitor strategy)
- Status** : Query returned k result tuples with 5.000 pages?

**Setup:** We implemented routing policies (see Section 3) and the keyword generation strategy from [24] for the java-based in-

Strategy	Q-ID	#J	#A	k	RT	Buf	%RT/Buf	P/Q	P/RT	Speedup	Status
UK-RND	2-06	1	5	500	500	17593	2,84	3505	7,0	1,00	ok
DK-RND	2-06	1	5	500	<b>535</b>	18049	2,96	3030	5,7	1,24	ok
<b>DK-COMB</b>	2-06	1	5	500	504	<b>10657</b>	<b>4,73</b>	<b>2790</b>	<b>5,5</b>	<b>1,27</b>	<b>ok</b>
UK-RND	2-07	1	4	500	298	17653	1,69	5000	16,8	1,00	dnf
DK-RND	2-07	1	4	500	474	13355	3,55	5000	10,5	1,59	dnf
<b>DK-COMB</b>	2-07	1	4	500	<b>531</b>	<b>4961</b>	<b>10,70</b>	<b>2099</b>	<b>4,0</b>	<b>4,24</b>	<b>ok</b>
UK-RND	3-01	2	5	500	276	28995	0,95	5000	18,1	1,00	dnf
DK-RND	3-01	2	5	500	<b>729</b>	12139	6,01	2091	2,9	6,29	ok
<b>DK-COMB</b>	3-01	2	5	500	504	<b>4380</b>	<b>11,51</b>	<b>651</b>	<b>1,3</b>	<b>13,98</b>	<b>ok</b>
UK-RND	3-08	2	7	500	97	7669	1,26	5000	51,6	1,00	dnf
DK-RND	3-08	2	7	500	106	3854	2,75	5000	47,2	1,09	dnf
<b>DK-COMB</b>	3-08	2	7	500	<b>544</b>	<b>2796</b>	<b>19,46</b>	<b>750</b>	<b>1,4</b>	<b>37,43</b>	<b>ok</b>
UK-RND	3-09	2	4	500	<b>550</b>	4323	12,72	4919	8,9	1,00	ok
DK-RND	3-09	2	4	500	371	4407	8,42	5003	13,5	0,66	dnf
<b>DK-COMB</b>	3-09	2	4	500	500	<b>2716</b>	<b>18,41</b>	<b>3549</b>	<b>7,1</b>	<b>1,26</b>	<b>ok</b>
UK-RND	4-01	3	6	500	502	4331	11,59	1859	3,7	1,00	ok
DK-RND	4-01	3	6	500	<b>868</b>	3116	27,86	1914	2,1	1,68	ok
<b>DK-COMB</b>	4-01	3	6	500	532	<b>1367</b>	<b>38,92</b>	<b>570</b>	<b>1,1</b>	<b>3,46</b>	<b>ok</b>
UK-RND	5-01	4	6	500	501	43013	1,16	4856	9,7	1,00	ok
DK-RND	5-01	4	6	500	528	39112	1,35	3234	6,1	1,58	ok
<b>DK-COMB</b>	5-01	4	6	500	<b>1242</b>	<b>23682</b>	<b>5,24</b>	<b>657</b>	<b>0,5</b>	<b>18,32</b>	<b>ok</b>

Figure 4: Results for different routing policies and keyword strategies.

memory database HSQLDB [19]. Experiments are conducted on a T61 laptop (Windows 7) with 4 GB of RAM and a core duo CPU with 2.2 GHz. Each experiment is executed six times and the average value is computed across experiments.

#### 3.2 Evaluated strategies

The most interesting comparative experiments are concerned with different keyword generation strategies and different routing policies for the same query.

**UK-RND (Expert keywords, Zig-Zag policy):** This strategy is our baseline. The strategy utilizes Zig-Zag joins [5] and user-defined keywords (see Figure 6).

**DK-RND (Generated Keywords, Zig-Zag policy):** This strategy also utilizes Zig-Zag Joins [5] and generated keywords from [24] (see Figure 6).

**DK-COMB (Generated Keywords, CBRP):** This strategy implements the *completeness based routing policy* from Section 3 and utilizes generated keywords from [24].

**Parameter:** We set k=500 (minimal complete result tuple size) and n=10 (top pages). The query execution is terminated either, if k is reached or if 5.000 pages are processed.

#### 3.3 Evaluation results

Figure 4 shows our results from which we obtain the following interesting observations.

**DK-COMB is a clear winner:** This strategy always returns any-k complete result tuples for our queries. Contrary, the Zig-Zag policy is significantly less successful. Strategy DK-RND failed to return at least k=500 tuples for two queries, 2-07 and 3-08. Worse, strategy UK-RND even failed for three queries, 2-07, 3-01 and 3-08. These measurements show that generated keyword queries and a completeness based routing policy ensure sufficient result tuples.

**Drastic reduction of query time for DK-COMB:** For filling missing attribute values, our routing algorithm performs the cycle of querying the search engine and extracting missing relations. This process can itself grow into a very time- and resource-intensive process. Our experiments demonstrate that strategy DK-

COMB can drastically speed up this process. For instance, strategy DK-COMB requires fewer pages per query ( $P/Q$ ) and per result tuple ( $P/RT$ ) than any other strategy. As a result, we observe a speed up between our baseline UK-RND and strategy DK-COMB of more than an order of magnitude for queries 3-01, 3-08 and 5-01.

**DK-COMB is effective:** The proportion of complete result tuples to all tuples in the buffer ( $\%RT/Buf$ ) is comparably high for strategy DK-COMB in contrast to other strategies. This measurement indicates that the query processor can *de facto* join extracted relations into result tuples. Strategy DK-COMB processed only a single page to return a complete result tuple ( $RT/P$ ) for queries 3-01, 3-08, 4-01 and 5-01. Note, for some queries, the size of completed result tuples ( $RT$ ) exceeds  $k=500$ . This occurs when multiple tuples are returned from the last keyword query. Then the size of the full outer join of returned relations and result tuples in the buffer exceeds  $k$ .

**Generated keywords offer moderate speedup:** In most cases the generated keywords achieve the highest speedup, specified in Figure 4. Moreover, we observe the most efficient buffer usage ( $\%RT/Buf$ ) for the generated keyword method. Our evaluation clearly shows the effectiveness of our keyword generation strategy, because mostly the generated keywords return the best results. Hence, we use the generated keywords for the *completeness based routing policy* (CBRP) strategy.

### 3.4 Continuously adapting parameters

**Cold start scenario:** In practice, the query processor may even not have a trained classifier for estimating the Gain ratio for each relation extractor and attribute type. In this case the database is empty and no additional information about the value distribution is available. We consider such a cold start scenario as worst case for our query processor and investigate if the query processor can learn these parameters quickly.

**DK-COMB adapts quickly:** The first diagram in Figure 5 shows a snapshot after processing the first 20 result tuples. Strategy DK-COMB requires about 250 pages until the first result tuple is returned, which is slightly more than strategy UK-RND. However, strategy DK-COMB collects statistics about the completeness and value distribution of attribute types in existing result tuples. The query processor benefits from this additional information when obtaining the second result tuple (94 pages). For subsequent tuples we observe that the query processor processes significantly fewer pages than for any other strategy because of the constantly improving distribution learning.

The second diagram in Figure 5 shows the proportion of complete result tuples ( $RT$ ) for up to 5000 processed pages for all strategies. From the first result tuple on, strategy DK-COMB processes significantly fewer pages than strategy UK-RND. The trend becomes even more apparent with the growing size of complete result tuples: After processing 2000 pages, DK-COMB returns more than 300 result tuples while UK-RND only returns less than 100 and DK-RND slightly more than 200 tuples. DK-COMB outperforms UK-RND after processing approximate 800 pages due to better estimations of information gain ratio.

**Zig-Zag policy is significantly less adaptive:** Contrary, the Zig-Zag policy in strategy UK-RND does neither utilize information about the completeness of result tuples nor information about the attribute value distributions. Therefore, strategy UK-RND frequently generates a keyword query from a result tuple that has

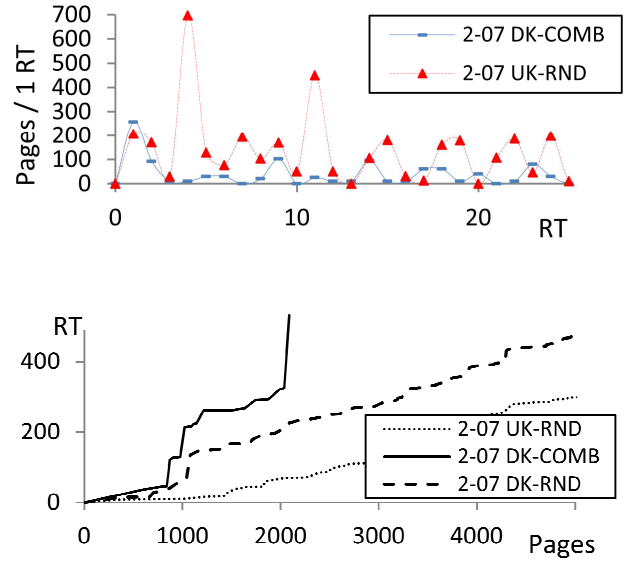


Figure 5: Execution details for query 2-07. The first Figure shows execution costs per complete result tuple for obtaining the first 20 result tuples. The second Figure shows the costs (in terms of extracted pages) for obtaining at least 500 complete result tuples.

not yet reached a high degree of completeness. As a result, this keyword query is less likely to return missing values to complete the tuple. Worse, the keyword query probably returns many irrelevant pages from which the relation extractor returns new incomplete result tuples. Therefore, the probability that a randomly selected result tuple has a high degree of completeness actually decreases with each keyword query executed from strategy UK-RND. Obviously illustrated by the low buffer usage ( $\%RT/Buf$ ) of this method.

We conclude that strategy UK-RND (as proposed by authors of [5]) is ineffective, while strategy DK-COMB effectively returns missing tuples. Moreover, DK-COMB adapts quickly in a cold-start scenario, where no information about promising join predicates exist. Note, we observed a similar behavior for other queries, whose details have been omitted due to the limited space. Therefore DK-COMB is the preferred strategy for discovering any- $k$  relations from Web search results with conjunctive queries.

### 3.5 Towards alternative quality indicators

In the result tuples, we were able to observe typical characteristics of 'general' Web content. For instance, we noticed that tuples in results are redundant, contradicting and contain few false positives. Few preliminary operators have been proposed so far that exploit this redundancy to cleanse complete tuples and to remove false positives. For instance, authors of [9] [30] envision operators for fusing or for ranking tuples. Moreover, instead of returning complete result tuples, the query processor may return *interesting relations* [2]. For example, a good set of results for the Entity "Einstein" might include a mix of biographical facts like "Einstein was born in Germany" and other interesting facts like "Einstein's favorite color was blue". On the other hand, "Einstein turned 15" or "Einstein wrote the paper" might be less interesting because they express little useful information. Another metric is *nearness* [13]. Relations are joined only into a result tuples if they

are extracted from the same page or same Web site. In our future work we will forward complete result tuples to these operators and will investigate their effect on the quality of result tuples.

#### 4. RELATED WORK

We review related work on executing structured queries across ‘text databases’, quality-driven query planning and text analytics.

**Queries for specific-k relations:** In our previous work [24] we focused on queries for *discovering specific-k relations*. Queries of that type execute a left outer join across a list of given entities and relations retrieved from Web pages. Contrary, queries for discovering *any-k* relations execute a full outer join on previously unseen relations that are discovered during query execution. Therefore, different routing policies are required. For instance, the routing policy must compute the degree of completeness or the chance of retrieving a missing relation during query execution and must incorporate these statistics into query routing decisions.

**Binary joins across relations in ‘text databases’:** Authors of [3][4][5] propose strategies for executing SELECT-PROJECT-JOIN queries across “text databases” By a “text database” they refer to a document collection that can be accessed through a (filtered) scan (for instance, the entire index of a Web search engine). In order to estimate an optimal join order for binary joins, attribute value distributions (that typically follow a Zipf-distribution) are estimated from a sample of documents from the “text database” [11]. Downloading large samples requires random access to all pages of a Web search engine. However, our query processor has only sequential, keyword-based access to an ordered list of pages. This list may even change over time for the same keyword query. Consequently, each keyword query returns a single “text database”. Estimating statistics for each keyword query is not possible in practice. Therefore we propose routing policies that can adaptively generate multiple keyword queries and select a query plan on a per-tuple-base. If such statistics cannot be computed in practice, authors suggest Zig-Zag joins [5] across binary relations. Our experiments demonstrate that Zig-Zag joins process up to an order of magnitude more pages per result tuple than other routing policies.

**Automatic keyword query generation.** A query processor requires discriminative keywords that likely return relevant pages from a Web search engine. In [24] we obtained such keywords in an adaptive, iterative fashion. We observe syntactic and lexical features from sentences in which an extractor extracted a tuple. Based on these features we trained a classifier. Keywords are updated when new pages are returned from the search engine. Therefore, generated keywords match the language of pages that are discovered during query execution. A similar method is presented by the authors of [8]. However, this method requires intensive domain specific training of the classifier prior executing a query as well as expert domain knowledge to generate a set of seed tuples.

Unfortunately, we do not have full access to documents. Therefore that approach is not practicable for our setting. Worse, the set of keywords is fixed and does not adapt to new documents that the query processor discovered during query execution. Authors of [18] extract candidate keywords from meaningful attribute names and predicate values from a structured query. We do not expect meaningful attribute names in a structured query. Rather we trust that Web page authors have already developed a

Relationship	Top-5 keywords from domain experts	Top-5 keywords after finishing query execution
<b>Arrestment</b> (Person,Date)	arrested, jail, puts, arrest, caught	arrested in, arrested on, arrested, charged with, arrest of
<b>Attribut</b> (Person,Born,Gender)	born, birth, years, old, age	born in, born on, born at, native of, raised in
<b>Bankruptcy</b> (Company, Status, Date)	bankruptcy, insolvency, applied insolvency, bankrupt, may file for	filed for, emerged in, protection in, declared, forced into
<b>Career</b> (Person,Company, Position)	said, spoke, worked at, is joining, left	became, board of, joined, organized by, worked at
<b>Conviction</b> (Person,Charge)	charges, conviction, sues, accuses, ligitates	convicted of, sentenced to, pleaded, conviction in
<b>Education</b> (Person, Organization)	joining, left, student, achieved, major	member of, served, joined, came to, lead
<b>Employees</b> (Company, EmployeeSize)	employees, employees in	employs over, employs around, employs, employs about
<b>HQ</b> (Company, Location)	located, headquarter, based, founded,	based in, company of, headquarters in, offices in, located
<b>Product</b> (Company,Product)	demand, financed, launched, is introducing, family of products	introduced, launched, to manufacture, creator of, to sell
<b>Recommendation</b> (Analyst,Company, Trend,Date)	downgrades, upgrades, recommends, buy, hold	rating on, downgraded by, buy from, raised, upped
<b>Travel</b> (Person,Location)	visits, travels, travel schedule, trip, went	went to, traveled to, returned to, trip to, flew to

**Figure 6:** As most competitive ‘opponent’ for our keyword generation strategy we consider a human that is aware with technical limitations of relation extraction techniques. To simulate this human opponent, two experienced engineers studied the ‘mechanics’ of extraction systems Texrunner [10] and DIAL[21] and observed words from common news sites for expressing relation. Based on this knowledge we manually crafted the best possible keywords. In addition, we show keywords that our system [24] generated during query execution.

small set of common words and grammatical structures to express important relationships in natural language.

**Query planning and query processing:** Among the large amount of literature, feedback loops for learning correct cardinality estimates [1] are most relevant for us. However, these strategies assume full access to structured data in an RDBMS to compute a query plan. Branch and bound based query planning algorithms have been designed for integrating a small set of static databases [6]. In our scenario we are confronted with a sheer endless number of volatile data sources and may discover new sources during query processing. Instead of computing a single plan we update statistics during query execution and route queries adaptively along multiple plans [14]. Top-k query processing approaches [29] merge ranking lists from different search engines. Unfortunately, the ranking of a Web search engine does not reflect the existence of a missing tuple. Instead; we select missing tuples that can be likely retrieved with a keyword query.

**Information extraction and text analytics:** The CIMPLE [7] project describes an information extraction plan with an abstract, predicate-based rule language. More recently, open information extraction techniques generalize relationships with relationship-independent lexico-syntactic patterns [10]. We assume that these systems and the extraction mechanics are ‘black boxes’ for a Web searcher. Therefore we observe common patterns from extracted relations that likely reflect the internal mechanics of a relation extractor. We use these patterns for generating keyword queries.

**‘Semantic search’ and ‘linked data’:** Neither current Web search engines nor Business intelligence applications are able to answer queries for any-k complete tuples. As a proof of concept we designed the semantic search engine www.GOOLAP.info [27]. The search engine discovers any-k tuples from news pages and blogs to populate its database. Another important scenario is the

enrichment of ‘linked data’ [26]. Linked data services, such as *Freebase* [25], rely on community efforts to submit relations. However, these services may only return relations that are ‘extracted’ by community editors. Queries for any-*k* tuples can automatically discover complementary tuples for *Freebase*.

## 5. SUMMARY

We proposed a novel query processor for systematically discovering any-*k* tuples from Web search results with conjunctive queries. Because of the nature of data on the Web, relation extractors may return incomplete tuples. Therefore, we present a solid theoretical model and powerful, adaptive query planning techniques to iteratively complete these tuples. Our experiments demonstrate that our query processor returns complete result tuples and processes very few Web pages only.

In our future work we will incorporate data cleansing [9] and tuple ranking techniques [2] into query processing to ensure not only complete, but also informative and concise result tuples. To allow a broad user community executing structured queries on Web search engines, we will publish our implementation for the in-memory database *HSQLDB* [19].

## ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Union’s Seventh Framework Programme (FP7/2007-2013) under grant agreement n° FP7-ICT-2009-5-257859, ‘Risk and Opportunity management of huge-scale *BUSINESS* community cooperation’ (*ROBUST*).

## 6. REFERENCES

- [1] Markl V., Raman V., Simmen D.E., Lohman G.M., Pirahesh M.: Robust Query Processing through Progressive Optimization. *SIGMOD Conference 2004*: 659-670
- [2] Kasneci G., Ramanath M., Suchanek F.M., Weikum G.: The *YAGO-NAGA* approach to knowledge discovery. *SIGMOD Row 37(4)*: 41-47 (2008)
- [3] Jain, A., Doan, A., and Gravano, L. 2008. Optimizing SQL Queries over Text Databases. *ICDE. IEEE Computer Society, Washington, DC*, 636-645.
- [4] Jain, A. and Srivastava, D. 2009. Exploring a Few Good Tuples from Text Databases. *ICDE. IEEE Computer Society, Washington, DC*, 616-627.
- [5] Jain, A., Ipeirotis, P. G., Doan, A., and Gravano, L. 2009. Join Optimization of Information Extraction Output: Quality Matters! *ICDE. IEEE Computer Society, Washington, DC*.
- [6] Naumann, F., 2002. *Quality-Driven Query Answering for Integrated Information Systems*. Springer
- [7] Shen, W., DeRose, P., McCann, R., Doan, A., and Ramakrishnan, R. 2008. Toward best-effort information extraction. *SIGMOD '08. ACM, New York, NY*, 1031-1042.
- [8] Agichtein, E. and Gravano, L. 2003. *QXtract*: a building block for efficient information extraction from Web page collections. *SIGMOD '03. ACM, New York, NY*, 663-663.
- [9] Galhardas, H., Florescu, D., Shasha, D., Simon, E., and Saita, C. 2001. *Declarative Data Cleaning: Language, Model, and Algorithms. Very Large Data Bases. Morgan Kaufmann Publishers, Rome, CA*, 371-380.
- [10] Etzioni, O., Banko M., Soderland S., Weld, D.S.: Open information extraction from the Web. *Commun. ACM 51(12)*: 68-74 (2008)
- [11] Ipeirotis, P. G., Agichtein, E., Jain, P., and Gravano, L. 2006. To search or to crawl?: towards a query optimizer for text-centric tasks. *SIGMOD '06. ACM, New York, NY*
- [12] Xin Dong, Alon Y. Halevy, Jayant Madhavan: Reference Reconciliation in Complex Information Spaces. *SIGMOD Conference 2005*: 85-96
- [13] Löser, A, Lutter S., Düssel, P, Markl V. 2009. Ad-hoc Queries over Web page Collections – a Case Study. *BIRTE Workshop at VLDB 2009*.
- [14] Avnur R., Hellerstein J.M.: Eddies: Continuously Adaptive Query Processing *SIGMOD Conference 2000*: 261-272
- [15] YahooBoss service. <http://developer.yahoo.com/search/boss> (Last visited 01/03/10)
- [16] OpenCalais. <http://www.opencalais.com> (Last visited 01/01/11)
- [17] Levy A., Mendelzon A.O., Sagiv Y., Srivastava D.: Answering Queries Using Views. *PODS 1995*: 95-104
- [18] Liu J., Dong X., Halevy A.Y.: Answering Structured Queries on Unstructured Data. *WebDB 2006*
- [19] *HSQLDB*. <http://hsqldb.org/> (Last visited 01/01/11)
- [20] Fung G., Yu, J., Lu, H.: Discriminative Category Matching: Efficient Text Classification for Huge Document Collections. *ICDM 2002*: 187-194
- [21] Feldman R., Regev Y., Gorodetsky M.: A modular information extraction system. *Intell. Data Anal. 12(1)*: 51-71 (2008)
- [22] Löser, A, Hüske F., Markl V. Situational Business Intelligence. *BIRTE Workshop at VLDB 2008*
- [23] Löser, A. Beyond Search: Web-Scale Business Analytics. *WISE 2009*: 5
- [24] Löser, A., Nagel, C., Pieper, S. Augmenting Tables by Self-Supervised Web search. *BIRTE Workshop at VLDB 2010*
- [25] *Freebase*. [www.freebase.com](http://www.freebase.com) (Last visited 01/01/11)
- [26] Bizer C., Heath T., Berners-Lee T.: Linked Data - The Story So Far. *Int. J. Semantic Web Inf. Syst. 5(3)*: 1-22 (2009)
- [27] *Goolap.info*. [www.goolap.info](http://www.goolap.info) (Last visited 01/01/11)
- [28] Mitchell T. *Machine Learning*. McGraw-Hill, 1997.
- [29] Ilyas I., Beskales G., and Soliman M.A.: A survey of top-*k* query processing techniques in relational database systems. *ACM Comput. Surv. 40(4)*: (2008)
- [30] Jain A, Pantel P.: FactRank: Random Walks on a Web of Facts. *COLING- 2010*
- [31] Boden C., Häfele T., Löser A.: Classification Algorithms for Relation Prediction. *DaLi Workshop at ICDE 2011* (forthcoming)