RJ10466 (A1005-011) May 25, 2010
Computer Science

IBM Research Report

WikiAnalytics: Disambiguation of Keyword Search Results on Highly Heterogeneous Structured Data

Andrey Balmin
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099
USA

Emiran Curtmola
Department of Computer Science
University of California San Diego
La Jolla, CA 92093
USA

WikiAnalytics: Disambiguation of Keyword Search Results on Highly Heterogeneous Structured Data

ANDREY BALMIN
IBM Almaden Research Center
and
EMIRAN CURTMOLA
University of California, San Diego

Wikipedia infoboxes is an example of a seemingly structured, yet extraordinarily heterogenous dataset, where any given record has only a tiny fraction of all possible fields. Such data cannot be queried using traditional means without a massive a priori integration effort, since even for a simple request the result values span many record types and fields. On the other hand, the solutions based on keyword search are too imprecise to exactly capture the user's intent.

To address these limitations, we propose a system, referred to herein as WikiAnalytics, that utilizes a novel search paradigm in order to derive tables of precise and complete results from Wikipedia infobox records. The user starts with a keyword search query that finds a superset of the result records, and then browses clusters of records deciding which are and are not relevant. WikiAnalytics uses three categories of clustering features based on record types, fields, and values that matched the query keywords, respectively. Since the system cannot predict which combination of features will be important to the user, it efficiently generates all possible clusters of records by all sets of features. We utilize a novel data structure, universal navigational lattice (UNL), that compactly encodes all possible clusters. WikiAnalytics provides a dynamic and intuitive interface that lets the user explore the UNL and construct homogeneous structured tables, which can be further queried and aggregated using the conventional tools.

Categories and Subject Descriptors: H.3.5 [**Information Storage and Retrieval**]: On-line Information Services; H.2.m [**Database Management**]: Miscellaneous

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: ad-hoc querying, heterogeneous structured data, search web data, keyword search

1. INTRODUCTION

Growing popularity of Wikipedia [Wikipedia] and other wikis raises the issue of querying this data to extract insights that span multiple pages. Although most of Wikipedia is free text, it also contains a large amount of structured information in tables, lists, categories, and infoboxes. A number of ongoing efforts [DBpedia

Author's addresses: Andrey Balmin, IBM Almaden Research Center (ARC), San Jose, CA 95120; email: abalmin@us.ibm.com; Emiran Curtmola, Department of Computer Science at University of California San Diego, La Jolla, CA 92093; email: ecurtmola@cs.ucsd.edu;

; Cammarano et al. 2007; Bollacker et al. 2007; Powerset] aim to harness this information.

We focus on querying Wikipedia infoboxes, which are essentially typed records of field-value pairs. Infoboxes appear on over a million Wikipedia pages and often contain the most vital information about the entity described by the page. For example, an infobox on Arnold Schwarzenegger's page (Figure 2) contains information about his office, family, birthday, party and religious affiliation, and more.

A major challenge in querying infoboxes is the diversity of their structure. Every infobox instance has an equivalent of a type – wiki template that renders the infobox WikiText into HTML. However, new templates can be introduced as well as old templates can be extended relatively easily. Moreover, enabling query processing was never a requirement for the authors of templates and infoboxes. As a result, templates often allow for many ways of representing the same information. For example, a very popular "officeholder" template has both `date of birth` and `birthdate` fields. Figure 1 conveys the heterogeneity of the infoboxes. In the Wikipedia fragment we analyzed, there are about $2,500$ distinct infobox types (wiki templates), with over $50,000$ distinct <type, field> pairs. However, there is a clear long tail in the distribution of the number of occurrences of the fields, with almost $20,000$ fields occurring in exactly one infobox and only 300 fields occurring in over $4,000$.

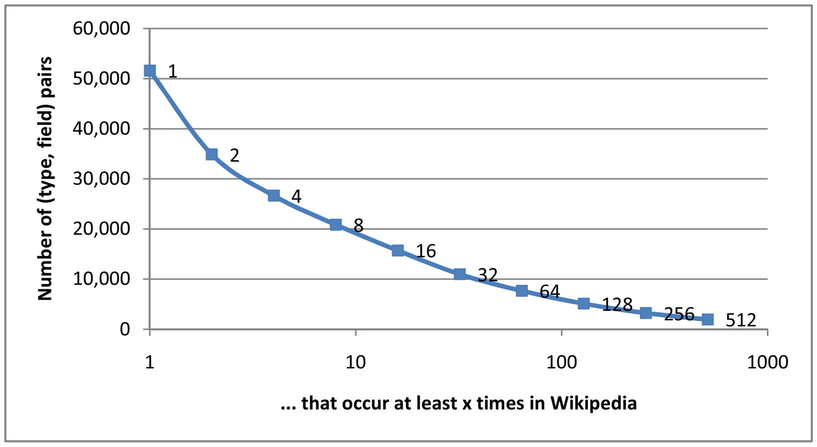


Fig. 1: *Wikipedia Infoboxes are sparse: Distribution of fields per number of infobox instances in which they occur.*

Many other types of data, such as product catalogs, patient records, and electronic forms collections, exhibit similar structural diversity. These sources are also often designed for human consumption with structural flexibility as the key feature and query processing as an afterthought. As a result, many products in a catalog may have rare or unique fields, and most fields on any given form may be optional, and different doctors will fill out the same clinical documents differently.

Such structural diversity presents only minor problems for point queries, e.g. "Schwarzenegger's age", where the answer is contained in a single infobox. After all, current state-of-the-art search engines can find Schwarzenegger's infobox and a human will be easily able to compute his age. Although "age" is not mentioned in the infobox text, we assume that synonym expansion will also look for "birthday". Structural diversity presents major problems when queries need however to access many objects (infoboxes) in order to extract lists of results from them. For example, if a user wants to construct a list of Governors of California, a good heuristic may be to look for infoboxes of type `governor` and `office` field containing value "Governor of California." However, thus constructed list will be only 90% correct. For example, Ronald Reagan's infobox has type `president`, with value "33rd Governor of California" hidden in the `order2` field. We call such results *structural outliers*. They are critical for deriving a complete and precise answer.

Supporting such queries can be viewed as an instance of data integration problem, where results may have multiple schemas, e.g. `governor` and `president`, and the same schema can be used differently in different data instances. Nonetheless, it is hard to imagine *a priori* reliable integration of information from all large clusters and outliers for the entire dataset - either heuristic or manual. The extreme heterogeneity of the dataset makes automatic or manual integration of the entire dataset infeasible. Instead, we adopt a "pay as you go" approach, where only the objects potentially relevant to the result are interactively integrated at query time.

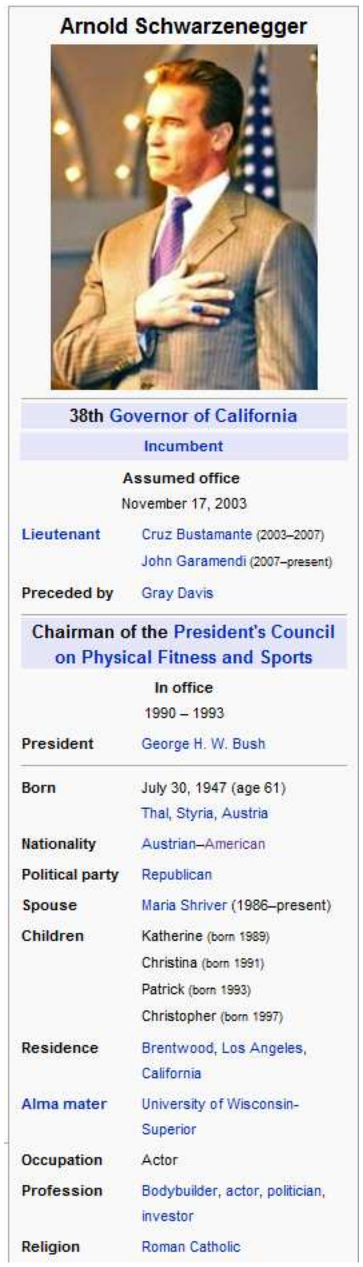
In this paper we present a system, referred to herein as WikiAnalytics, which enables users to browse multiple clusters of all potential results, and relatively easily identify the main result cluster(s) as well as the outliers. The clustering features that we use are based on the names and values of fields that contain the query keywords. We call such fields and their values *features*. Conceptually, the features define the relevant dimensions on the data specifying the matching context for the query keywords. The resulting clusters allow users to disambiguate the query based on the structure of the results. The intuition is that occurrence of the same keyword in different fields or in different values is likely to have different meanings. For example, a group of `governor` infoboxes with "California" in the `office` field is semantically different from a group where the same keyword occurs in the `birthplace` field. Furthermore, even within the "California" $\in$ `office` cluster, there is a significant difference between infoboxes with values "Governor of California" in the `office` field and "Governor of Baja California" in the same field.

In order to give users a full picture of the possible clusters of the query results we adopt a notion of *concept lattice* [Ganter and Wille 1999] over the clusters of infoboxes. Our *universal navigational lattice* (UNL) encodes all possible ways to group the records in the query result according to their features. We developed a GUI that allows users to navigate the UNL and interact with it by including and excluding the clusters from the result list.

The UNL usually grows super-linearly with the size of the result, so we introduce a pruning technique that filters out features that occurred fewer times than a user-defined *feature support threshold* (FST). Besides reducing the UNL size, pruning greatly speeds-up the UNL construction, and makes the result easier for users to comprehend and work with. In a typical session, FST is initially set relatively high,

```
{{ Infobox Governor
  | name = Arnold Schwarzenegger
  | nick = Governator
  | image = Arnold Schwarzenegger 2004-01-30.jpg
  | imagesize = 200px
  | order = 38th
  | office = Governor of California
  | term_start = November 17, 2003
  | lieutenant = {{nowrap|[[Cruz Bustamante]]<small>
    (2003-2007)< /small>}}<br/ >{{nowrap|[[John
    Garamendi]] <small>(2007-present)< /small>}}
  | predecessor = [[Gray Davis]]
  | successor =
  | order2 = Chairman of the [[President's Council
    on Physical Fitness and Sports]]
  | term_start2 = 1990
  | term_end2 = 1993
  | president2 = [[George H. W. Bush]]
  | birth_date = {{birth date and age|1947|07|30}}
  | birth_place = [[Thal, Austria|Thal]], [[Styria]],
    [[Austria]]
  | nationality = [[Austria]][[United States|American]]
  | party = [[Republican Party (United States)|
    Republican]]
  | spouse = {{nowrap|[[Maria Shriver]] (1986-present)}}
  | religion = [[Roman Catholic]]
  ...}}
```

(a) *Source code with the WikiText markup that generates Figure 2b.*



Arnold Schwarzenegger

38th Governor of California

Incumbent

Assumed office
November 17, 2003

Lieutenant	Cruz Bustamante (2003–2007) John Garamendi (2007–present)
Preceded by	Gray Davis

Chairman of the President's Council on Physical Fitness and Sports

In office
1990 – 1993

President	George H. W. Bush
Born	July 30, 1947 (age 61) Thal, Styria, Austria
Nationality	Austrian–American
Political party	Republican
Spouse	Maria Shriver (1986–present)
Children	Katherine (born 1989) Christina (born 1991) Patrick (born 1993) Christopher (born 1997)
Residence	Brentwood, Los Angeles, California
Alma mater	University of Wisconsin-Superior
Occupation	Actor
Profession	Bodybuilder, actor, politician, investor
Religion	Roman Catholic

(b) *Infobox visualization in Wikipedia.*

Fig. 2: *Sample Wikipedia Infobox: "Arnold Schwarzenegger"*

to filter out the long tail of features and allow the user to focus on large clusters of structurally homogeneous records. Then, the user can accept or reject some of these clusters, which consist entirely of results or non-results, respectively. Finally, the user can recompute the UNL with a lower FST, over the remaining objects after the exclusion of already accepted or rejected records. The last two steps can be repeated iteratively allowing the user to zoom in on progressively smaller clusters in order to look for structural outliers.

The final result of a WikiAnalytics query is a table with a key column (name of the wiki page) and a value column for every keyword specified as an extraction, by the special "!" character.

EXAMPLE 1.1. (**Running example**) For instance, user keyword query "California governor religion!" returns a data feed comprising of pairs of governors' (page) names and their religious affiliations. ◇

The resulting feeds can be joined and aggregated by mashup tools like Yahoo Pipes [YahooPipes] and Damia [Simmen et al. 2008], or visualized by services like Swivel [Swivel] and Many Eyes [ManyEyes].

In this paper, we make the following contributions.

—The WikiAnalytics system which tackles the problem of extracting lists of homogeneous and precise results from highly heterogeneous set of Wikipedia infoboxes. The system heuristically finds most of the results, but also effectively summarizes the potential results. This enables easy verification and modification of the results by the user.

—A formal framework and the algorithms [Balmin and Curtmola 2010a] for summarizing the search results based on a universal navigational lattice (UNL), which clusters the infoboxes based on *features* dynamically defined as fields and values that match the search terms. Each cluster contains infoboxes that share certain combination of features, and thus are likely to be semantically similar w.r.t the query.

—An interactive user interface [Balmin and Curtmola 2010b] that visualizes the UNL lattice and allows users to disambiguate search results by navigating the UNL and arbitrarily include and exclude clusters of infoboxes from the result. The users start the exploration by visiting larger groups of documents corresponding to single features. From each group, the user may drill down by considering one or more features to restrict the document set. The interface also enables users to dynamically prune the lattice, by controlling the granularity of the clusters they want to see.

—Our experiments with querying Wikipedia infoboxes show that the our pruning heuristics behave well in practice and facilitate on-line querying of highly heterogeneous data sets. WikiAnalytics enables average users to explore the search results and construct homogeneous structured tables, which can be further joined, aggregated, and visualized using the conventional tools.

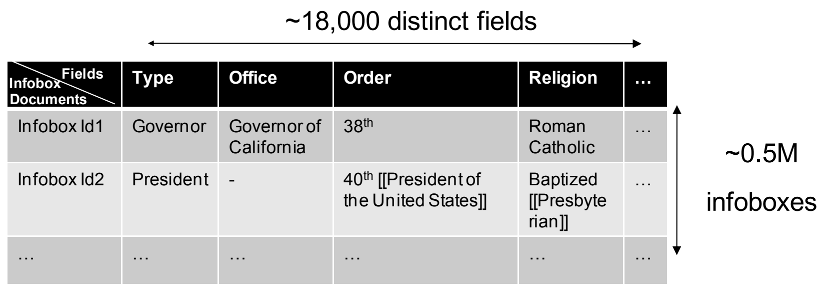
Paper Outline. The rest of the paper is organized as follows. In the next three sections, we describe our formal framework. In particular, in the next section, we formalize WikiAnalytics data and query model. Section 3 describes clustering features whereas Section 4 defines the Universal Navigational Lattice (UNL). Section 5 describes the WikiAnalytics system architecture built on top of a Wikipedia snapshot. We introduce our algorithms for UNL construction and visualization in Section 6. In the next section, we introduce our pruning techniques and describe the process of interactive query disambiguation. Experimental results are shown in Section 8. Related work is discussed in Section 9 and we summarize our study and future work in Section 10.

2. DATA AND QUERY MODEL

We model a Wikipedia infobox as a record that comprises of a set of *fields*. Given a record r, we denote by $Fld(r)$ its set of fields: $Fld(r) = \{(f.name, f.value) | f \in r\}$. We also denote by $r.N$ the set of all field names of r. Each record has a type, $r.type$ that identifies a set of possible field names for records of this type; i.e., $r.type$ maps to a superset of $r.N$. We assume that records don't have duplicate field names and that field values are strings.

We model a collection of records $\mathcal{R}$ as a *universal* table U. This table contains a column for every distinct field name in $\mathcal{R}$; i.e., the set of column names in U is $Col(\mathsf{U}) = \cup\{r.N | \forall r \in \mathcal{R}\}$. In addition, we add a special `type` column to U to represent the record type information. Each record $r \in \mathcal{R}$ corresponds to a row in U. Therefore, a table cell U_{ij} contains the field value $f_k.value$ for field f_k under record r_i, such that $f_k.name = Col(\mathsf{U})_j$. The cell value is null if no such f_k exists in record r_i.

EXAMPLE 2.1. A fragment of the universal table U for Wikipedia infoboxes is shown in Figure 3. The full universal table would comprise around 18,000 columns and approximately 500,000 rows [1] . However, only 0.08% of its cells would have non-null values. $\diamond$


~18,000 distinct fields
~0.5M infoboxes

Fields / Infobox Documents	Type	Office	Order	Religion	…
Infobox Id1	Governor	Governor of California	38th	Roman Catholic	…
Infobox Id2	President	-	40th [[President of the United States]]	Baptized [[Presbyte rian]]	…
…	…	…	…	…	…

Fig. 3: *Fragment of the Wikipedia infobox universal table.*

We consider keyword queries where a query $Q = (k_1, \ldots, k_n)$ defines three category of keywords: keywords $C(Q)$ that appear in matching records, keywords $N(Q)$ that should not appear in matching records, and keywords $R(Q)$ that identify the

[1]The infoboxes were extracted as a subset of templates embedded in pages of November 2009 snapshot of the Wikipedia.

query result as columns of U. For our running example defined on page 5, we have that $C(Q) = \{\text{“}governor\text{”}, \text{“}California\text{”}\}$ and $R(Q) = \{\text{“}religion\text{”}\}$.

A keyword k appears in record r, or $k \in r$, if k appears in either one of its field names or values: $\{f_i \in Fld(r) | k \in f_i.name \vee k \in f_i.value\} \neq \emptyset$. We say that a *record r matches a query* $Q = (k_1, \dots, k_n)$ if $\forall k_i \in C(Q) \cup R(Q)$ then $k_i \in r$, and $\forall k_i \in N(Q)$ then $k_i \notin r$.

Given a data collection of records $\mathcal{R}$, the *candidate result set* of Q on $\mathcal{R}$, denoted by $Q(\mathcal{R})$, is the set of records that match Q: $Q(\mathcal{R}) = \{r \in \mathcal{R} | r \ matches \ Q\}$. We denote by $Fld(\mathcal{R})$ the set of all field names in the collection across all records in the collection: $Fld(\mathcal{R}) = \{f.name | \forall r \in \mathcal{R}, f \in Fld(r)\}$.

To derive a complete answer set that satisfies an information need corresponding to query Q, the user may choose to iteratively refine the candidate result set by adding or removing groups of records. We denote this feedback selection process over the candidate set of records for user u with $\sigma_u(Q(\mathcal{R}))$. Finally, the query result is a table T, which is the result of extracting the query specified return values after the user selection by projecting on the fields matched to $R(Q)$; by abuse of notation, we also refer to these fields as $R(Q)$:

$$T = \pi_{R(Q) \cup key(R)}(\sigma_u(Q(\mathcal{R})))$$

The record key $key(R)$ is used only for presentation of records. We used Wikipedia page title as a key.

3. CLUSTERING FEATURES

To help users identify the final result subset of the candidate set, we cluster the records that pass the keyword search filter based on their types, as well as field names and values that contain the keywords. Our goal is to produce clusters that the user will easily identify as entirely relevant or entirely irrelevant to the query. Intuitively, the average user is interested to explore and identify useful information in the corpus based on the properties of fields that match the keywords specified in a query.

Given a candidate result $Q(\mathcal{R})$ we define the following three categories of clustering features, which will be used to summarize and slice the result set. First, it is often important to know the type of records in $Q(\mathcal{R})$. We say that $\mathsf{F}^\mathsf{t} : type = k$ is a *type feature* for record r if $r.type = k$.

Second, the keyword may occur in different fields belonging to various records. In order to distinguish which is the field f that keyword k hits, we say that $\mathsf{F}^\mathsf{c} : k \in f$ is a *containment feature* for record r if $f \in Fld(r) \wedge (k \in f.name \vee k \in f.value)$.

Finally, the field matching a keyword might correspond to a variety of distinct textual values s, which may be important in partitioning the search results. Intuitively, a partition with value “Governor of California” in the `office` field, will be relevant to our running example query, unlike a partition with value “Governor of Baja California” in the same field. In order to distinguish between such partitions, we say that $\mathsf{F}^\mathsf{e} : f = s$ is an *equality feature* for record r if $f \in Fld(r) \wedge f.value = s$.

Note that a feature corresponds to the mapping of a keyword on fields of $\mathcal{R}$. It specifies what is the matching context for query keywords into the set of records. Logically, a feature can be seen as a dynamic dimension on the whole corpus of records. For instance, the keyword “president” might represent different match contexts such as a person, a US president, a University president etc. If the feature that a person is a "president" is important, we allow the user to slice on the corpus on this specific feature of interest.

Therefore, each feature uniquely determines a cluster of records that are characterized by this feature (i.e., records with the same matching context). Formally, we define a *feature association function* $\mathsf{C}^R : \{\mathsf{F}^\mathsf{t}, \mathsf{F}^\mathsf{c}, \mathsf{F}^\mathsf{e}\}^* \rightarrow \mathcal{R}$ to cluster the set of records R according to features. For instance, $\mathsf{C}^R(F)$ is the subset of records in R that have feature F. For a given set of features, C^R associates the set of all records in R that conform with the features as follows:

$$\mathsf{C}^R(F_1, \dots, F_n) = \{r \in R | \forall i \in [1, n], F_i \ is\ a\ feature\ of\ r\}$$

Thus, we can say that $\mathsf{C}^R(F_1, \dots, F_n) = \bigcap_i \mathsf{C}^R(F_i)$.

EXAMPLE 3.1. For instance, the following set of features specifies that the field `office` should contain the word "California" and the actual value of the field is "Governor of California": $\mathsf{F^c}_1$: "$California$" $\in office$ and $\mathsf{F^e}_2$: $office$ = "$Governor\ of\ California$." Furthermore, the above selection of features determines uniquely the cluster $\mathsf{C}^R(\mathsf{F^c}_1, \mathsf{F^e}_2)$ of all records from collection R that satisfy both features. Note that in this case $\mathsf{F^e}_2$ implies $\mathsf{F^c}_1$, thus, $\mathsf{C}^R(\mathsf{F^c}_1, \mathsf{F^e}_2) = \mathsf{C}^R(\mathsf{F^e}_2)$ ◇

We introduce next the set of all features $\mathcal{F}^\mathsf{Q}$ *induced* by a query Q on $\mathcal{R}$ as $\mathcal{F}^\mathsf{Q} = \{\mathcal{F}^\mathsf{k} | \forall k \in Q\}$, where $\mathcal{F}^\mathsf{k}$ is the feature set induced by an individual keyword term k. $\mathcal{F}^\mathsf{k}$ corresponds to all cells of table U that contain a match with keyword k. It consists of the following components: the type feature set $\mathcal{F}^\mathsf{t}$ over U, the containment feature set $\mathcal{F}^\mathsf{c}$, which identifies all fields in U that have a match on k, and the equality feature set $\mathcal{F}^\mathsf{e}$, which identifies what values do the fields in U match with k. Their formal definitions is given below.

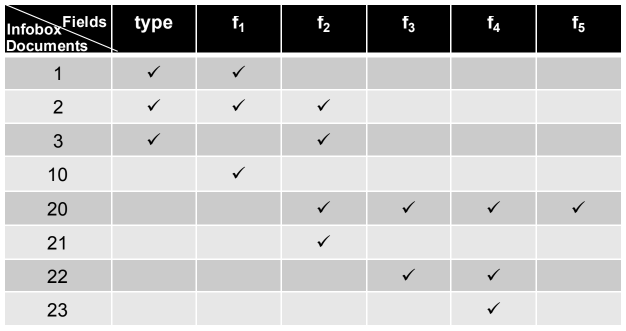
$$\begin{aligned}
\mathcal{F}^\mathsf{k} &= \mathcal{F}^\mathsf{t} \cup \mathcal{F}^\mathsf{c}(k) \cup \mathcal{F}^\mathsf{e}(k) \\
\mathcal{F}^\mathsf{t} &= \{\mathsf{F}^\mathsf{t} : type = term | \forall term \in \Pi_{type}(\mathsf{U})\} \\
\mathcal{F}^\mathsf{c}(k) &= \{\mathsf{F}^\mathsf{c} : k \in f | \forall j, f \in \mathsf{U}_{*,j} \wedge (k \in f.name \vee k \in f.value)\} \\
\mathcal{F}^\mathsf{e}(k) &= \{\mathsf{F}^\mathsf{e} : f = f.value | \forall j, f \in \mathsf{U}_{*,j} \wedge (k \in f.name\ \vee k \in f.value)\}
\end{aligned}$$

Note that such a feature set $\mathcal{F}^\mathsf{Q}$ might contain features with references to records that match with one or few keywords specified in the user query. To deliver a meaningful $\mathcal{F}^\mathsf{Q}$ to the user, we restrict the set of features induced by Q over $\mathcal{R}$ to those features that refer to a subset of records that have a higher relevance to the query.

DEFINITION 3.1. (**Restricted universal table**) Let Q be a query and $Q(\mathcal{R})$ its candidate result. The corresponding universal table restricted to $Q(\mathcal{R})$, U^Q, identifies only the information in U that corresponds to records of $Q(\mathcal{R})$: $\mathsf{U}^\mathsf{Q} = \{\mathsf{U}_i | \forall i, r_i \in Q(\mathcal{R})\}$. ◇

Finally, we revisit the definitions for feature sets $\mathcal{F}^\mathsf{k}$ and $\mathcal{F}^\mathsf{Q}$. We restrict them to apply only on the subtable $\mathsf{U}^\mathsf{Q} \subset \mathsf{U}$. For instance, $\mathcal{F}^\mathsf{Q}$ stands for the features induced by query Q on records $Q(\mathcal{R})$. We show in Section 6.1 how to compute $\mathcal{F}^\mathsf{Q}$ efficiently.

EXAMPLE 3.2. Figure 4 shows a subset of the restricted universal table U^{Q} for our running example query. The columns represent some fields *type*, $f_1, \ldots, f_5$ of $\mathcal{R}$ that contain hits with one of the query keywords, whereas the rows represent documents in $Q(\mathcal{R}) = \{1, 2, 3, 10, 20, 21, 22, 23\}$ that contain all the query keywords. We have indicated with a check mark the corresponding fields where the keywords hit the documents (while omitting the actual field contents). Table I shows the matching features $\mathcal{F}^{\mathsf{Q}}$ together with their associated clusters $\mathsf{C}^{Q(\mathcal{R})}$ over the universal table U^{Q}. ◇



Infobox Documents \ Fields	type	f_1	f_2	f_3	f_4	f_5
1	✓	✓				
2	✓	✓	✓			
3	✓		✓			
10		✓				
20			✓	✓	✓	✓
21			✓			
22				✓	✓	
23					✓	

Fig. 4: *Universal table* U^{Q} *for example 3.2.*

Features $\mathcal{F}^{\mathsf{Q}}$	Clusters $\mathsf{C}^{Q(\mathcal{R})}$
$F_1 = \{\mathsf{F}^{\mathsf{t}} : type = governor\}$	$\{1, 2, 3\}$
$F_2 = \{\mathsf{F}^{\mathsf{c}} : \text{“}Governor\text{”} \in f_1\}$	$\{1, 2, 10\}$
$F_3 = \{\mathsf{F}^{\mathsf{e}} : f_1 = \text{“}Governor\ of\ California\text{”}\}$	$\{1, 2, 10\}$
$F_4 = \{\mathsf{F}^{\mathsf{c}} : \text{“}governor\text{”} \in f_2\}$	$\{2, 3, 20, 21\}$
$F_5 = \{\mathsf{F}^{\mathsf{c}} : \text{“}governor\text{”} \in f_3\}$	$\{20, 22\}$
$F_6 = \{\mathsf{F}^{\mathsf{e}} : f_4 = \text{“}religion\text{”}\}$	$\{20, 22, 23\}$
$F_7 = \{\mathsf{F}^{\mathsf{c}} : \text{“}california\text{”} \in f_5\}$	$\{20\}$

Table I: *Features and Clusters for Example 3.2*

We introduce next the universal navigation lattice (UNL), a data structure that enables the enumeration of all clusters of records that are viable and that are of interest based on the user query. The structure ensures that no record is omitted from being available nor selected by the user.

4. UNIVERSAL NAVIGATIONAL LATTICE

Since we cannot predict which combination of features will be important to the user, we generate all possible clusterings of records by all sets of features determined by the user query. In this way, the user can explore the data corpus by navigating on dynamic dimensions defined at query time as corpus locations where the query keywords hit. We do this compactly and efficiently, by organizing the clusters into a lattice structure called the *universal navigational lattice* (UNL). The clusters in UNL are connected with each other based on the subsumption relationship of features.

Given a set of restricted features $\mathcal{F}^{\mathsf{Q}}$ that are relevant to query Q, the powerset of features of $\mathcal{F}^{\mathsf{Q}}$ forms a lattice $(L, \leq)$ with the following two binary operations: (1) join ($\vee$) is the union of features, and (2) meet ($\wedge$) is the intersection of features such that $\forall a, b \in L$ where $a = \{F_i | F_i \in \mathcal{F}^{\mathsf{Q}}\}$ and $b = \{F_i | F_i \in \mathcal{F}^{\mathsf{Q}}\}$ then $a \vee b = \bigcup F_i$ and $a \wedge b = \bigcap F_i$, respectively.

The partial order relation, $\leq$, on the elements of L is defined as the subset relationship between feature sets. The bottom element of L corresponds to the empty set, while the top element is the union of all features. Moreover, the way we defined the join operation implies that the cluster associated with $a \vee b$ corresponds to all records satisfying both feature sets a and b, that is $\mathsf{C}^{Q(\mathcal{R})}(a \vee b) = \mathsf{C}^{Q(\mathcal{R})}(a) \cap \mathsf{C}^{Q(\mathcal{R})}(b)$.

DEFINITION 4.1. (**UNL**) We define the universal navigational lattice UNL over the meet-semilattice of $(L, \leq)$ (i.e., if a, b are feature sets with non-empty cluster of records then $a \wedge b$ is also a non-empty cluster feature set) with the following properties: (i) a feature set $a \in$ UNL is the description of a non-empty cluster of records $\mathsf{C}^{Q(\mathcal{R})}(a)$; (ii) no two elements have the same cluster of records, i.e., $\forall a, b \in$ UNL then $\mathsf{C}^{Q(\mathcal{R})}(a) \neq \mathsf{C}^{Q(\mathcal{R})}(b)$. In other words, each element in UNL is a unique feature set and has a unique cluster of records. $\diamond$

Let us note that UNL is not closed under the join operation, i.e., if a, b are non-empty cluster feature sets it does not mean that a non-empty cluster for $a \vee b$ exists in UNL all the time. We consider that elements of UNL are organized on levels based on the number of features they contain. For instance, an element with four features is considered to be on level four in UNL.

5. SYSTEM ARCHITECTURE

In this section, we describe our design goals and the system architecture of WIKI-ANALYTICS. Our design is constrained by three main requirements: (i) an easy to use and an effective search interface to explore and disambiguate answers by making data selections while navigating heterogeneous collections, (ii) enable the user to select a complete and precise set of answers according to intentions expressed initially as a keyword query, and (iii) the search should not modify nor markup the original data corpus in a way to facilitate data discovery.

Based on these design decisions, we chose the architecture shown in Figure 5 consisting of the following parts: data storage and indexing, query processing, and a dynamic user interface with cluster selection.

For the first part, we extract the infoboxes from a Wikipedia snapshot of 2009 and we convert them to XML by using the Wiki2XML tool of the Texterra project[2]. Note that our model is generic enough that it can capture relational data as well as semi-structured documents. For example, for relational databases records are tuples, record types are table names, and fields are attribute-value pairs. XML data can also be shredded into records if a data modeler provides a set of XPath expressions to generate field names (unique per record) and string values. Our

[2] http://modis.ispras.ru/texterra/download/index.html

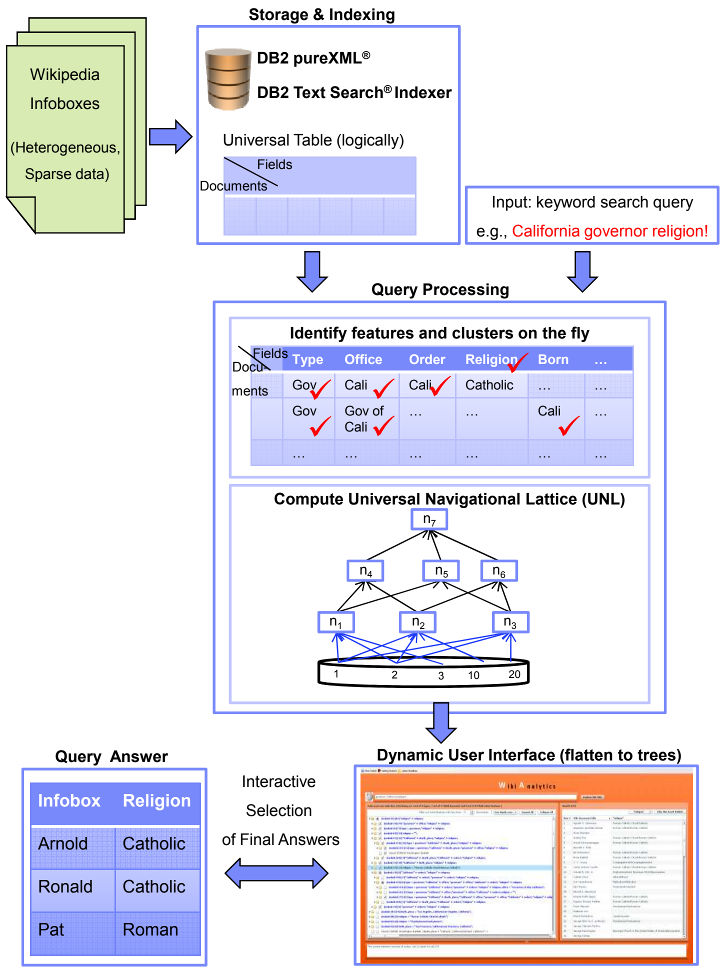

Wikipedia Infoboxes
(Heterogeneous, Sparse data)
Storage & Indexing
DB2 pureXML®
DB2 Text Search® Indexer
Universal Table (logically)
Fields
Documents
Input: keyword search query
e.g., California governor religion!
Query Processing
Identify features and clusters on the fly
Fields
Documents
Type Office Order Religion Born …
Gov Cali Cali Catholic … …
Gov Gov of Cali … … Cali …
Compute Universal Navigational Lattice (UNL)
n7
n4 n5 n6
n1 n2 n3
1 2 3 10 20
Query Answer
Infobox Religion
Arnold Catholic
Ronald Catholic
Pat Roman
Interactive Selection of Final Answers
Dynamic User Interface (flatten to trees)

Fig. 5: WikiAnalytics *Architecture*

choice or representation is XML. For instance, Figure 6 shows a sample record in XML format of Arnold Schwarzengger's infobox, which corresponds to Figure 2. In order to store and query infoboxes, we use an off-the-shelf keyword search system. Mainly, we store them in IBM's DB2 pureXML® database which comes with native XML storage and querying support. We leverage the power of DB2 Text Search® engine for XML full-text search. This allows seamless access to the structural part as well as to the textual part of the documents. We delegate the text search engine for query term expansion to handle stemming, case insensitive search, stop words removal, and other standard recall-enhancing techniques.

The query processing part consists of two modules: feature extraction and construction of the universal navigation lattice UNL. The text index produces a set of infobox documents that match all query keywords. Note that the keyword search by itself is not sufficient for building result lists since it cannot provide 100% precision. Imprecise results are tolerable for "point" queries because the user can browse

```
<infobox type="Governor">
  <field>
    <field_name>name</field_name>
    <field_value>Arnold Schwarzenegger
    </field_value></field>
  <field>
    <field_name>nick</field_name>
    <field_value>Governator
    </field_value></field>
  <field>
    <field_name>order</field_name>
    <field_value>38th</field_value>
  </field>
  <field>
    <field_name>office</field_name>
    <field_value>Governor of California
    </field_value></field>
  ...
  <field>
    <field_name>religion</field_name>
    <field_value> Roman Catholic
    </field_value></field>
  ...
</infobox>
```

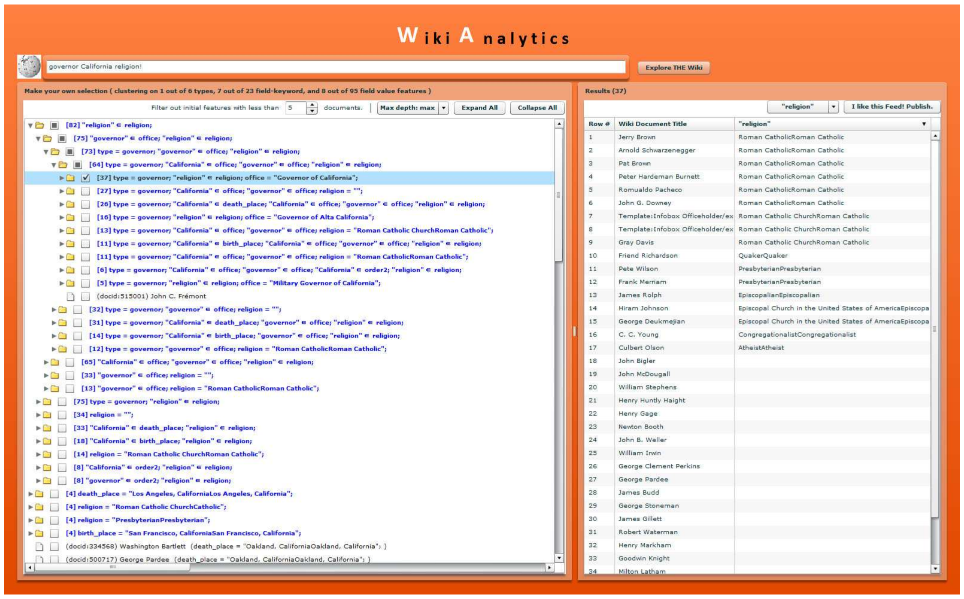
Fig. 6: *Infobox as an XML document*

a few candidates to identify a single result. However, if a user needs to compile or aggregate a list of tens or hundreds or thousands of infoboxes, browsing each candidate individually becomes infeasible. In this case, clustering results simplifies the browsing process and enables users to accept and reject semantically similar results as a group.

The first module conceptually views these infoboxes as a (sparse) universal table, with a row for each infobox and a column for every field that occurs in at least one of them. A feature with a corresponding cluster of documents is created for each field name and field value that contain a given query keyword. The second module builds the UNL lattice graph. Intuitively, UNL encodes all possible meaningful clusters of documents by all sets of features. We create relevant clusters and links between them so as to use the features as dynamic structural dimensions that slice and dice in the data collection to facilitate document exploration and selection.

In the third part we introduce techniques for visualizing the UNL lattice and for interactively allowing the selection of records in order to facilitate discovery of complete query answers. The lattice itself provides a dynamic interface that lets users control the navigation and selection over the corpus. UNL lends naturally to a bottom-up visual representation, which resembles a multi-faceted search like interface if we follow the links. We use the features defined on the corpus as dynamic navigational dimensions. The user will follow links in the lattice by selecting feature sets starting from more general groups of records characterized by less features toward more specific groups, i.e., more features at nodes therefore, more selective on the record groupings. In general, this is true since a link between the lattice elements a to b stands for $a \subset b$ (i.e., feature sets inclusion) and $\mathsf{C}^{Q(\mathcal{R})}(a) \supset \mathsf{C}^{Q(\mathcal{R})}(b)$ (i.e., cluster set inclusion). The lattice is exposed into a tree based interface. This facilitates complete access to the data without dropping any query answers and enables smart querying as well as easy data selection.

Lastly, we extract the final result for further data processing, e.g., business intelligence data analytics or data mashups. Figure 7 shows our target search GUI interface. On the left side, the interface allows users to interact and navigate over clusters of records, whereas on the right side it shows the current data feed selection.



governor California religion!

Make your own selection (clustering on 1 out of 6 types, 7 out of 23 field-keyword, and 8 out of 95 field value features)

Filter out initial features with less than 5 documents.

```
▼ [82] "religion" ∈ religion;
  ▼ [75] "governor" ∈ office; "religion" ∈ religion;
    ▼ [73] type = governor; "governor" ∈ office; "religion" ∈ religion;
      ▼ [64] type = governor; "California" ∈ office; "governor" ∈ office; "religion" ∈ religion;
        ▶ ✓ [37] type = governor; "religion" ∈ religion; office = "Governor of California";
        ▶ [27] type = governor; "California" ∈ office; "governor" ∈ office; religion = "";
        ▶ [26] type = governor; "California" ∈ death_place; "California" ∈ office; "governor" ∈ office; "religion" ∈ religion;
        ▶ [16] type = governor; "religion" ∈ religion; office = "Governor of Alta California";
        ▶ [13] type = governor; "California" ∈ office; "governor" ∈ office; religion = "Roman Catholic ChurchRoman Catholic";
        ▶ [11] type = governor; "California" ∈ birth_place; "California" ∈ office; "governor" ∈ office; "religion" ∈ religion;
        ▶ [11] type = governor; "California" ∈ office; "governor" ∈ office; religion = "Roman CatholicRoman Catholic";
        ▶ [6] type = governor; "California" ∈ office; "governor" ∈ office; "California" ∈ order2; "religion" ∈ religion;
        ▶ [5] type = governor; "religion" ∈ religion; office = "Military Governor of California";
          (docid:515001) John C. Frémont
      ▶ [32] type = governor; "governor" ∈ office; religion = "";
      ▶ [31] type = governor; "California" ∈ death_place; "governor" ∈ office; "religion" ∈ religion;
      ▶ [14] type = governor; "California" ∈ birth_place; "governor" ∈ office; "religion" ∈ religion;
      ▶ [12] type = governor; "governor" ∈ office; religion = "Roman CatholicRoman Catholic";
    ▶ [65] "California" ∈ office; "governor" ∈ office; "religion" ∈ religion;
    ▶ [33] "governor" ∈ office; religion = "";
    ▶ [13] "governor" ∈ office; religion = "Roman CatholicRoman Catholic";
  ▶ [75] type = governor; "religion" ∈ religion;
  ▶ [34] religion = "";
  ▶ [33] "California" ∈ death_place; "religion" ∈ religion;
  ▶ [18] "California" ∈ birth_place; "religion" ∈ religion;
  ▶ [14] religion = "Roman Catholic ChurchRoman Catholic";
  ▶ [8] "California" ∈ order2; "religion" ∈ religion;
  ▶ [8] "governor" ∈ order2; "religion" ∈ religion;
▶ [4] death_place = "Los Angeles, CaliforniaLos Angeles, California";
▶ [4] religion = "Roman Catholic ChurchCatholic";
▶ [4] religion = "PresbyterianPresbyterian";
▶ [4] birth_place = "San Francisco, CaliforniaSan Francisco, California";
  (docid:334568) Washington Bartlett  (death_place = "Oakland, CaliforniaOakland, California"; )
  (docid:500717) George Pardee  (death_place = "Oakland, CaliforniaOakland, California"; )
```

Results (37)

Row #	Wiki Document Title	"religion"
1	Jerry Brown	Roman CatholicRoman Catholic
2	Arnold Schwarzenegger	Roman CatholicRoman Catholic
3	Pat Brown	Roman CatholicRoman Catholic
4	Peter Hardeman Burnett	Roman CatholicRoman Catholic
5	Romualdo Pacheco	Roman CatholicRoman Catholic
6	John G. Downey	Roman CatholicRoman Catholic
7	Template:Infobox Officeholder/ex	Roman Catholic ChurchRoman Catholic
8	Template:Infobox Officeholder/ex	Roman Catholic ChurchRoman Catholic
9	Gray Davis	Roman Catholic ChurchRoman Catholic
10	Friend Richardson	QuakerQuaker
11	Pete Wilson	PresbyterianPresbyterian
12	Frank Merriam	PresbyterianPresbyterian
13	James Rolph	EpiscopalianEpiscopalian
14	Hiram Johnson	Episcopal Church in the United States of AmericaEpiscopa
15	George Deukmejian	Episcopal Church in the United States of AmericaEpiscopa
16	C. C. Young	CongregationalistCongregationalist
17	Culbert Olson	AtheistAtheist
18	John Bigler	
19	John McDougall	
20	William Stephens	
21	Henry Huntly Haight	
22	Henry Gage	
23	Newton Booth	
24	John B. Weller	
25	William Irwin	
26	George Clement Perkins	
27	George Pardee	
28	James Budd	
29	George Stoneman	
30	James Gillett	
31	Robert Waterman	
32	Henry Markham	
33	Goodwin Knight	
34	Milton Latham	

Fig. 7: *GUI: Faceted Search-like Interface over the Documents Clustered by Feature Sets*

We underline the difference with the traditional faceted search. Our choice of clustering features, which depends on the query is a key difference between our approach and prior works on concept lattice and faceted search. These works cluster objects only based on field values that are known *a priori*. In contrast, our "facets" are dynamically defined at query time. The novelty of our approach is that the record clusters capture the context of where the keywords match in the corpus. We call these matches to be dynamic features as opposed to statically predefined hierarchies of data properties, e.g., size, color, price etc. We enable browsing the data corpus and discovering meaningful records using the matching context as a guide, where the context is defined by the query keyword hits in the corpus. We describe pruning heuristic techniques for the data exploration and selection process in Section 7.

WikiAnalytics provides a web-based interface to the data. We implemented the backend using Java servlet technology to extract the features from DB2 and to construct the UNL in memory. For presentation, the lattice is flattened into trees and serialized to a Adobe Flex frontend application. The GUI allows interactive visualization of the UNL with support for (de)selection of record clusters. For more details on our system, please see [Balmin and Curtmola 2010a].

6. ALGORITHMS

In this section, we elaborate on the lattice-based algorithm called MakeGroups that computes the UNL and navigates on it. The algorithm works in three phases as follows.

Phase 1 identifies the features $\mathcal{F}^{\mathsf{Q}}$ induced by query Q over record collection $\mathcal{R}$, and computes clusters $\mathsf{C}^{Q(\mathcal{R})}(F)$ corresponding to each feature $F \in \mathcal{F}^{\mathsf{Q}}$. That is, all the categories of features (type, containment, and equality) are computed based on the location where query keywords hit in the records $\mathcal{R}$. For each distinct feature, we create a cluster comprising of all the records this feature appears in. The set of features $\mathcal{F}^{\mathsf{Q}}$ and their clusters are the basis for the UNL generation which is described in the next phase.

Phase 2 computes the UNL lattice starting with the initial set of features $\mathcal{F}^{\mathsf{Q}}$ and their associated clusters $\mathsf{C}^{Q(\mathcal{R})}$ over $\mathcal{R}$.

Finally, phase 3 facilitates navigation on the UNL lattice and generates a tree like user interface in order to enable exploration, discovery, and selection of records of interest. We describe next each phase in details.

6.1 Computation of $\mathcal{F}^{\mathsf{Q}}$

We show in this section how to compute the set of features $\mathcal{F}^{\mathsf{Q}}$ and the associated clusters $\mathsf{C}^{Q(\mathcal{R})}(F)$ for each feature $F \in \mathcal{F}^{\mathsf{Q}}$ over the corpus. The corresponding pseudocode is presented in function compute_features of Figure 8. One possibility to derive the features is to materialize the universal table U first, and later collect the matches with the query keywords. Note that this approach is infeasible due to the size and the sparsity of the data being considered.

To achieve this in practice, we leverage the power of a full-text processor over XML documents that allows to retrieve all the field value pairs (also called “hits”) for which there is at least one match with one of the query keywords. Moreover, in order to increase the precision, the search is restricted to the set of records $Q(\mathcal{R})$ that contain all keywords in any combination of field names and values. Therefore, for each keyword k we compute the corresponding fields and their hits as well as the records they appear in by using the following primitive invertedIndex(k, Q, R) in lines 2-3. This function has access to the full-text indices and returns all matching pairs (f, d) for which field f in record $r \in Q(\mathcal{R})$ contains a keyword match with one of the terms of query Q.

Since infobox fields may contain XML content, we store the records in XML columns of DB2 pureXML database and we access them using SQL/XML language. For fast access to the keywords inverted indices, we build text indices over the XML content of the records. The invertedIndex function takes advantage of all these techniques as well as of the database fielded search capability to filter the fields in an infobox document. For more details about the index and the data access technologies please see the system architecture in Section 5.

Each matched pair, a field f in record r, returned by invertedIndex on keyword k produces one of each feature category: a type feature for r's type $\mathsf{F}^{\mathsf{t}} : type = r.type$ (lines 4-6), a containment feature $\mathsf{F}^{\mathsf{c}} : k \in f$ (line 7-8), and an equality feature

$\mathsf{F^e} : f = f.value$ (lines 9-10).

All such discovered features are part of the output of algorithm phase 1. In addition to computing features, we also keep track of the clusters of records $\mathsf{C}^{Q(\mathcal{R})}$ that have contributed at the creation of each feature in the code of function update_features in Figure 8.

```
algorithm compute_features($Q$, $\mathcal{R}$)
input: query $Q$, set of records $\mathcal{R}$
output: initial features $\mathcal{F}^{\mathsf{Q}}$, initial clusters $\mathsf{C}^{Q(\mathcal{R})}$
begin
1    $\mathcal{F}^{\mathsf{Q}} = \{\emptyset\}$
2    for each keyword term $k \in Q$ do
3      for each field $f$ in record $r$ such that $(f, r) \in$ invertedIndex($k$, $Q$, $\mathcal{R}$)
4        per distinct document $r$
5          create new feature $\mathsf{F^t}$ : $type = r.type$
6          update_features($\mathsf{F^t}$, $r$, $\mathcal{F}^{\mathsf{Q}}$, $\mathsf{C}^{Q(\mathcal{R})}$)
7        create new feature $\mathsf{F^c}$ : $k \in f$
8        update_features($\mathsf{F^c}$, $r$, $\mathcal{F}^{\mathsf{Q}}$, $\mathsf{C}^{Q(\mathcal{R})}$)
9        create new feature $\mathsf{F^e}$ : $f = f.value$
10       update_features($\mathsf{F^e}$, $r$, $\mathcal{F}^{\mathsf{Q}}$, $\mathsf{C}^{Q(\mathcal{R})}$)
11   return $\mathcal{F}^{\mathsf{Q}}$, $\mathsf{C}^{Q(\mathcal{R})}$
end

algorithm update_features($F$, $r$, $\mathcal{F}^{\mathsf{Q}}$, $\mathsf{C}^{Q(\mathcal{R})}$)
input: feature $F$, record $r$, features $\mathcal{F}^{\mathsf{Q}}$, clusters $\mathsf{C}^{Q(\mathcal{R})}$
output: features $\mathcal{F}^{\mathsf{Q}}$, clusters $\mathsf{C}^{Q(\mathcal{R})}$
begin
12    if (!$\mathcal{F}^{\mathsf{Q}}.contains(F)$) then $\mathsf{C}^{Q(\mathcal{R})}(F) = \{\emptyset\}$
13    $\mathcal{F}^{\mathsf{Q}} += F$
14    $\mathsf{C}^{Q(\mathcal{R})}(F) += r.docid$
end
```

Fig. 8: *Phase 1 of Algorithm* MakeGroups *(computation of $\mathcal{F}^{\mathsf{Q}}$)*

6.2 Construction of UNL

We now describe an efficient algorithm that constructs a Universal Navigational Lattice. The pseudocode of this algorithm, called **compute_unl**, is shown in Figure 9.

We capture the lattice as a direct acyclic graph (DAG) data structure, $\mathsf{UNL} = (N, E)$. Each node $n \in N$ is characterized by a set of features $n.F$ and the corresponding cluster of records that have all these features $n.I = \mathsf{C}^{Q(\mathcal{R})}(n.F)$. A link $(n_1 \to n_2) \in E$ connects two nodes such that $n_1.I \supset n_2.I$ and $n_1.F \subset n_2.F$.

Our algorithm constructs the lattice graph UNL bottom-up, inductively, level by level such that level L_{k-1} generates the next level L_k and possibly some nodes on the higher levels depending on the outcome of **merge_nodes**. This strategy avoids generating all combinations of nodes. Therefore, it benefits from pruning the nodes as early as possible if they do not satisfy the definition of UNL, i.e. they correspond to duplicate or empty clusters of records. Interestingly, the structural heterogeneity actually makes UNL construction feasible. In the worst case the number of nodes in the lattice is a powerset of the number of features, but the sparser the features are,

```
algorithm compute_unl($\mathcal{F}^{\mathsf{Q}}$, $\mathsf{C}^{Q(\mathcal{R})}$)
input: features $\mathcal{F}^{\mathsf{Q}}$, clusters $\mathsf{C}^{Q(\mathcal{R})}$
output: universal navigational lattice UNL
begin
1   $N = \{\oslash\}$, $E = \{\oslash\}$
2   $initial\_nodes$ = consolidate_features($\mathcal{F}^{\mathsf{Q}}$, $\mathsf{C}^{Q(\mathcal{R})}$)
3   for each node $n \in initial\_nodes$ do
4     $N+ = n$
5   for each level $level = 2..|\mathcal{F}^{\mathsf{Q}}|$ do
6     for each node $n_1 \in N$ do
7       if $(|n_1.F| == level - 1)$ then
8         for each node $n_2 \in initial\_nodes$ do
9           construct $newNode$ : $\{\ F = n_1.F \cup n_2.F,\ I = n_1.I \cap n_2.I\ \}$
10          if $(newNode.I == \{\})$ then continue
11          $oldNode$ = find $oldNode \in N$ such that $oldNode.I == newNode.I$
12          if ($oldNode$ exists) then
13            $oldNode.F+ = newNode.F$
14            $E+ = (n_1 \rightarrow oldNode)$
15            $E+ = (n_2 \rightarrow oldNode)$
16          else
17            $N+ = newNode$
18            $E+ = (n_1 \rightarrow newNode)$
19            $E+ = (n_2 \rightarrow newNode)$
20  remove_generalized_triangles($N$, $E$)
22  return $(N, E)$ as UNL
end
```

Fig. 9: *Phase 2 of Algorithm* MakeGroups *(construction of* UNL*)*

the fewer clusters are constructed. We observed that the size of the lattice stays relatively small even for large numbers of infoboxes and features.

Before we start, remember that UNL is a lattice. Thus, when a new UNL node n is being generated as a result of joining nodes n_1 and n_2, we insure properties (i) and (ii) from UNL's definition. In particular, we keep only non-empty feature cluster nodes and collapse nodes that represent the same cluster of records into one node. Intuitively, the lattice should contain elements that correspond to unique set of records and unique set of features in order to avoid redundant navigation over the data. We summarize our discussion to add a new node to UNL as follows:

—R_1: if $\mathsf{C}^{Q(\mathcal{R})}(n) = \{\}$ then $n \notin$ UNL, and

—R_2: if $\exists y \in$ UNL such that $\mathsf{C}^{Q(\mathcal{R})}(n) = \mathsf{C}^{Q(\mathcal{R})}(y)$ then merge n into y if $y \neq n$ and add links from n_1 to y and from n_2 to y, and

—R_3: if such $y \nexists$ then add n to UNL and add links from n_1 to n and from n_2 to n

The algorithm starts at lines 2-4 by generating the first lattice level, and possibly other lower rank levels by consolidating the set of individual features $\mathcal{F}^{\mathsf{Q}}$, and their clusters $\mathsf{C}^{Q(\mathcal{R})}$, which were previously constructed in phase 1. The consolidation of features is required by UNL property (ii), as defined in Section 4. In particular, features are consolidated based on the clusters they represent. Thus, function consolidate_features, in line 2, builds a lattice node for each distinct cluster of records and groups all features that characterize that cluster onto that node (see Figure 11).

EXAMPLE 6.1. Building on our running example, the initial feature sets are computed as in Table I. After going through the consolidation process, the lattice consists of the following nodes $n_1 - n_6$ as displayed in Figure 10a. Each node n in the lattice has associated two sets: the set of features $n.F$ and the cluster of records $n.I$ under the context of the features. In particular, node n_2 consolidates features F_2 and F_3 (i.e., $n_2.F = \{F_2, F_3\}$) as a result of characterizing the same cluster set $n_2.I = \{1, 2, 10\}$. ◇

The algorithm continues by considering all possible groups of records by all sets of features. Lines 5-8 construct all subsequent levels of the lattice by extending the current level with one more feature as part of the previously consolidated nodes. At each step, a new lattice node is created. It consists of merging the features sets of the underlying nodes while taking a set intersection over their clusters (line 9 and **merge_nodes** in Figure 11). For each such new potential node, the algorithm applies two lattice pruning rules based on properties of the UNL definition. Rule (R_1), in line 10, disregards the node if it stands for an empty cluster. Rule (R_2), in lines 11-15, consolidates all nodes that have the same cluster by merging their feature sets. Finally, we add the new node to the graph lattice (lines 17-19) in case the cluster does not exist (the default rule R_3).

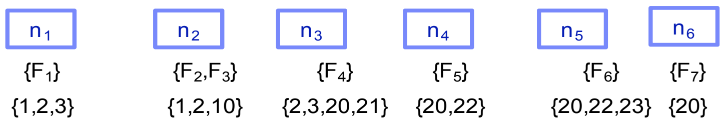
EXAMPLE 6.2. Construction of the lattice graph starts with the generation of the following nodes n_7, n_8, n_9 as a result of applying rule R_1. This is shown in Figure 10b. Each node is the result of taking the union of the features sets of the parents and of set intersecting the record clusters of the parents. For instance, $n_7.F = n_1.F \cup n_2.F$ while $n_7.I = n_1.I \cap n_2.I$. All combinations for which the cluster set is empty are not part of the lattice as stated by rule R_1. Next, the algorithm generates the combination between nodes n_7 and n_3 for which rule R_2 applies. Since their cluster intersection coincides with the cluster for node $n_9.I$, the new combination is merged onto n_9 as shown in Figure 10c. Similarly, this is observed for generating the combination between n_8 and n_3. Figure 10d shows the complete lattice graph where new nodes get generated for unique feature sets and unique clusters, or just merged with existing nodes. ◇

Note that UNL encodes all sets of features even though not every set corresponds to a node. For example, the set of features $\{F_1, F_2\}$ does not form a node in Figure 10d. However this set of features corresponds to cluster of records $n_7.I$ since $n_7.F$ is their smallest superset in UNL.

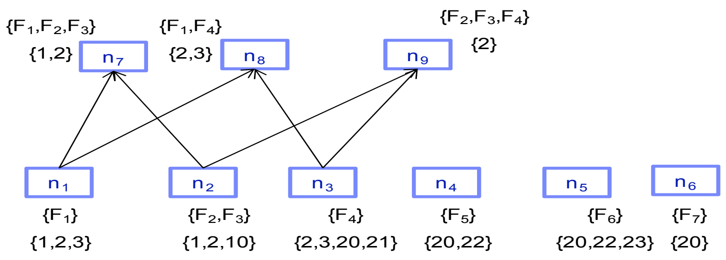
In the end, the algorithm eliminates all the redundant edges in line 20. The UNL edges represent cluster relationship, which is transitive. Consider an edge $e = (n_1, n_2)$, such that there is a path from n_1 to n_2 that does go through e. This edge does not encode any new information, since it can be reconstructed from the path. We remove such edges from UNL to keep it compact and simplify its presentation. We propose a set of effective heuristics that removes the direct links $n_1 \rightarrow n_2 \in E$ that close, what we call, *generalized triangles*

—n_1 is a parent node of n_2 on path p_1 from n_1 to n_2 ($|p_1| = 1$), and

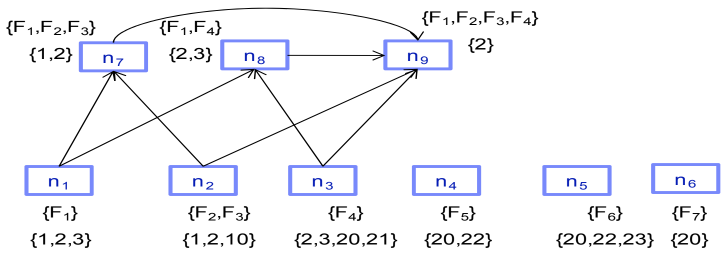
—n_1 is an ancestor node of n_2 on path p_2 from n_1 to n_2 such that $p_1 \neq p_2$ and $|p_2| > 1$



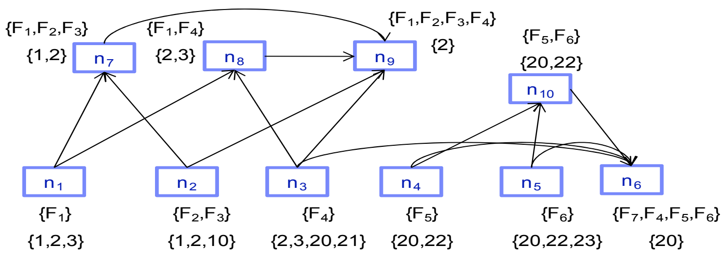
(a) *Initial lattice nodes after consolidation of* $\mathcal{F}^{\mathsf{Q}}$.



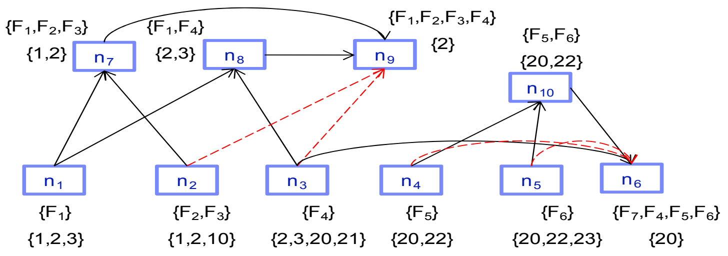
(b) *Generating nodes* n_7, n_8, *and* n_9 *using rule* R_1.



(c) *Merging new nodes on* n_9 *using rule* R_2.



(d) *All levels of* UNL *complete.*



(e) *Removing redundant links from* UNL.

Fig. 10: *Construction of* UNL *Universal Navigational Lattice Graph.*

```
algorithm consolidate_features($\mathcal{F}^{\mathsf{Q}}$, $\mathsf{C}^{Q(\mathcal{R})}$)
input: features $\mathcal{F}^{\mathsf{Q}}$, clusters $\mathsf{C}^{Q(\mathcal{R})}$
output: level 1 of the UNL
begin
    $N^{'} = \{\oslash\}$
    for each feature $F \in \mathcal{F}^{\mathsf{Q}}$ do
      if ($\exists n \in N^{'}$ such that $n.I == \mathsf{C}^{Q(\mathcal{R})}(F)$)) then
        $n.F+ = F$
      else
        $n.F = F, n.I = \mathsf{C}^{Q(\mathcal{R})}(F)$
        $N^{'}+ = n$
    return $N^{'}$
end

algorithm merge_nodes($n_1$, $n_2$)
input: nodes $n_1$, $n_2$
output: result node of merging the input nodes
begin
    $node.F = n_1.F \cup n_2.F$
    $node.I = n_1.I \cap n_2.I$
    return $node$
end

algorithm remove_generalized_triangles($N$, $E$)
input: lattice graph $(N, E)$
output: lattice graph $(N, E)$ without generalized triangle links
begin
    queue = empty
    for each root node $root \in N$ do
      queue.add(root)
    while (!$queue.isEmpty$) do
        $node$ = get next node from queue whose parents do not exist or have been processed
        $remove\_links(node)$
        for each child $c \in node.children$ do
          queue.add(c)
end

algorithm remove_links($node$)
input: currently visited node $node$
output: remove redundant links leading to $node$ in UNL
begin
    $node.ancestors = \oslash$
    for each parent $p \in node.parents$ do
      $node.ancestors+ = p.parents$
      $node.ancestors+ = p.ancestors$
    for each node $anc \in (node.parents \cap n.ancestors)$ do
      $E- = (anc \rightarrow node)$
    end
```

Fig. 11: *Details of Phase 2 of Algorithm* MakeGroups

The rule enforces that if there are multiple paths that reach node n_2 from node n_1, then discard the path that bares the “parent” relationship between the nodes. In other words, if there are ancestors of node n on a path with length greater than 1 which are also parents of n (presumably on a another path), then the links between the node and those parents are to be removed. Therefore, the heuristic promotes the longer path via other intermediate nodes to the short navigational path. The heuristic is valid because the algorithm removes redundant navigation in the lattice, i.e., the possibility to jump from features $n_1.F$ directly to features $n_2.F$ is not intuitive for navigation when there is another path with a length bigger than one that connects the same set of nodes.

Function remove_generalized_triangles in Figure 11 computes a set of indirect ancestors for each node n – the ancestor nodes of n excluding its direct parents. We do this in one pass over the lattice by employing breadth first search graph traversal starting from the roots (nodes without parents) of the lattice graph UNL, and processing the nodes only if all their descendants have been visited previously. In the end, we identify all links in E ("parent" relationships) that are simultaneously in "indirect ancestor" relationship, and remove them from E. We do this last step in **remove_links** function.

EXAMPLE 6.3. Our sample lattice graph contains four such redundant links closing generalized triangles. They are shown with dotted red lines in Figure 10e. For instance, the direct link $n_2 \rightarrow n_9$ is redundant since n_9 can be reached from n_2 via n_7. Intuitively, this is translates in better user navigation and answer discovery since the user wants to explore the collection in a gradual manner, i.e., explore records from very generic groups to more and more focused groups. Jumping directly from n_2 to n_9 omits the intermediate group of n_7. Otherwise, the user may go to n_7 instead. Eventually, n_9 can be reached if the user is not satisfied with n_7. ◇

6.3 Presentation of UNL

UNL lends itself to many ways of presentation and user interaction, such as faceted search or OLAP style slicing and dicing. To effectively answer the user query, we display the lattice in a tree like interface that allows easy navigation among all the groups: clusters are shown as parents of their sub-clusters. For each cluster we display its size and the full set of features. The user can expand or collapse any tree node, as well as select or unselect any cluster for the final result.

We obtain the tree shape by a depth-first traversal of UNL, starting from each of its root nodes. In practice, most of the lattice nodes are well connected so there are relatively few roots. The order of the child nodes in the tree is determined by the number of records for each child. We order them in descending order of the cluster size. A sample user interface is displayed in Figure 7.

EXAMPLE 6.4. In our running example, we allow the user to interact with the lattice by navigation on each tree in DFS order traversal. The user drills down and up based on the feature sets over the various clusters. In Figure 10e we distinguish five root nodes $n_1 - n_5$; therefore, there are five trees to explore. Out of the initial set of six nodes, only five remained roots. For instance, if the user is interested to find the set of records $\{20, 2\}$ then the user can go $n_4 \rightarrow n_{10} \rightarrow n_6$, select $n_6.I$, and union this selection with $n_9.I$ that was derived by navigation over $n_2 \rightarrow n_7 \rightarrow n_9$. ◇

7. USER EXPERIENCE

We describe in this section the pruning techniques in WikiAnalytics which facilitate effective user interaction to disambiguate query answers and to extract the necessary fields from the resulting records.

Constructing the Query Result. First, we describe the extraction of return values from a set of selected records S. In general, users are not necessarily interested in the full content of matched records. Usually, they are trying to construct a table with a particular set of fields that are relevant to their information need. For instance, in our working example, the user wants to compile the religious affiliations for past and current governors of California. This information can be found in the `religion` field of the governors' infoboxes.

In WikiAnalytics, the user specify a set of keywords $R(Q)$ that describe the return as part of the query Q's expression. However, for each $k \in R(Q)$ there can be multiple hits in the same record. By default, for each $k \in R(Q)$, WikiAnalytics picks a single field f that has the largest number of records with $k \in f$. However, we also give user choice lists with other candidate fields ordered by the number of field hits for k.

The query result is a table T that is extracted from the content of the fields matching to $R(Q)$ over S. This is equivalent to a projection over the user selected records on the query return part and the primary key of the collection, which in our case is a Wikipedia page name with an optional sequence number used in rare cases when multiple infoboxes are found on the same page. Thus

$$T = \pi_{R(Q) \cup key(R)}(S)$$

User Selection Process. It is rarely practical to present the full lattice as it is to the user in order to allow for disambiguation of the return records. Therefore, we propose below techniques to prune down the lattice to a manageable size.

We define the life cycle of a normal query processing for user u as a feedback process σ_u in which the user iteratively refines a set of records as part of the output selection S. This is similar to applying the relational selection operator iteratively on the candidate result set of the query, or

$$S = \sigma_u(Q(\mathcal{R}))$$

We introduce a technique for pruning features based on the size of the cluster they describe. Intuitively, features describing small clusters of records are more likely to be outliers. In order to separate homogeneous patterns from outliers, we introduce the user-controlled *feature support threshold* (FST). That is, all features with a cluster size smaller than FST are not considered during the initial UNL clustering.

We formalize this iterative life cycle process of query result disambiguation in the following steps.

—Step one identifies the larger cluster of structurally homogeneous records in the collection.
Initially, the selection process starts considering the candidate record set $Q(\mathcal{R})$, i.e., all the records that match the query. We propose a batch computation of UNL over $Q(\mathcal{R})$ for fixed FST. That is, we automatically filter out the features

with a cluster support smaller than FST from the initial UNL construction. In practice, FST depends on the application domain. For Wikipedia searches, good initial values are between 5 and 20.

—At lattice presentation time, we display the small clusters that contain fewer than FST records as a plain list. This list might be overwhelming for the user when trying to identify the structural outliers. We propose the next following steps as a solution.

—In step two, we allow automatic and manual record filtering mechanisms. First, we use a heuristic to identify the largest meaningful cluster of the results. The heuristic follows the edges UNL by always picking the largest sub-cluster, until each keyword is found in either the same type or field name for each infobox, or in the same value of the same field.
Second, the user can manually complement the current record selection with other clusters of records or individual records. He or she can also remove records from the current selection by clicking on cluster nodes. At any point, WikiAnalytics computes and displays the query result T of the current selection S.
Moreover, the user can "accept" or "reject" entire clusters from the lattice presentation thereby asserting that all or none of cluster infoboxes should be present in the result without affecting the selection. Thus, all infoboxes belonging to accepted or rejected clusters can be removed from the UNL.
We allow this "accept-reject clusters" user behavior in order to facilitate easy navigation over the lattice presentation, and filtering of non-meaningful or already processed clusters.

—Step three allows the user to decrease the FST value and refine the selection cycle by repeating steps two and three. Working on a pruned UNL has the advantage that the user can now focus on the smaller size clusters to search for the structural outliers.
The user can opt to visualize the query answers at any time during the selection process by extracting the actual return values from the selected records. The final query results are published as a data feed when the user is satisfied with the results extracted from the current selection of records S.

EXAMPLE 7.1. This example shows how we can find the complete list of answers to our working query. If we fix $FST = 5$, the automatic heuristic points to the bigger cluster consisting of the following features $\{\mathsf{F}^{\mathsf{t}} : type = governor$, F^{c} : "*religion*" $\in religion$, $\mathsf{F}^{\mathsf{e}} : office =$ "*Governor of California*"$\}$. Indeed, this cluster accounts for 35 out of 82 search results. However, there are three more records, somewhere in the outliers list, that belong in the query answer. Even with good domain knowledge, the user will find it hard to locate them. In order to focus on the outlier answers, let us "accept" the large result cluster and thus remove it from the presentation. Let us also reject three obviously irrelevant clusters: $\mathsf{F}^{\mathsf{e}} : office =$ "*Governor of Alta California*", $\mathsf{F}^{\mathsf{e}} : office =$ "*Governor of Baja California*", and $\mathsf{F}^{\mathsf{e}} : office =$ "*Military Governor of California*", which contain 16, 5, and 6 records respectively.

In step three, we recompute UNL for $FST = 0$. This will force to include in UNL all the clusters of all 20 remaining records. Analyzing this shorter list of records, it is easy to spot the three outlier records, which are governors of California that were not picked up automatically by the previous step. Ronald Reagan and Earl Warren are both outliers. They contain the information about being governor of California in the field `order2` and are of types `president` and `judge`, respectively. The last outlier is Washington Bartlett. Even though this record is of type `governor`, the state affiliation is given in its field `order` instead of `office`, which excludes it from the largest homogeneous cluster and makes it an outlier for our query. ◇

8. EXPERIMENTAL EVALUATION

In this section, we evaluate performance of the UNL construction algorithm. We show that despite the exponential complexity of the algorithm, judicious use of FST-based pruning helps us achieve on-line level performance. We have not yet performed a formal user-interaction evaluation, but we have positive experience from the in-house use of the tool in the context of a larger project [Midas].

We ran the WikiAnalytics system on a Centrino Duo 2.2GHz laptop with 2GB of RAM.

We confirm experimentally that FST plays an important role both in the efficiency of the UNL computation as well as in the selection of the complete query answers.

The tables below show how the lattice parameters vary with FST for queries with different selectivities on Wikipedia. Table II reports results for our running example, $Q1 =$"governor California religion!". As expected, the construction time and the size of UNL increase with the decrease of FST since the number of features after consolidation increases.

	FST=20	FST=15	FST=10	FST=5	FST=0
# docs in U^Q	82	82	82	82	82
# fields in U^Q	19	19	19	19	19
# features after consolidation	7	9	11	16	76
$\|N\|$	19	35	53	79	151
$\|E\|$	31	66	106	163	278
# roots	1	1	1	1	1
construct UNL	2ms	5ms	9ms	27ms	163ms

Table II: *Query Q_1 = "California governor religion!"*

To stress WikiAnalytics, we have focused on less selective queries, Q_2 and Q_3, as shown in Tables III-IV. Q_2 matches to approximately 1,000 documents, while Q_3 matches over 2,000 documents. In general, our tool is designed to extract results of a few hundred to a couple of thousand of answers at a time, as it is hard to find larger groups of semantically similar infoboxes.

We present Q_2 in Table III and we notice big impact of FST on the size and construction time of the UNL. Without use of FST, we could not support on-line ad-hoc querying. Indeed, for $FST = 0$ there are over 2,000 clusters available for browsing, and the UNL takes almost 2 minutes to compute. Bringing FST to 15, we experience reasonable search parameters, i.e., construction time of about 1 second and 465 nodes.

	FST=20	FST=15	FST=10	FST=5	FST=0
# docs in U^{Q}	1014	1014	1014	1014	1014
# fields in U^{Q}	32	32	32	32	32
# features after consolidation	33	40	71	166	1306
$\|N\|$	395	465	762	1389	2274
$\|E\|$	967	1137	1811	3159	4443
# root	2	2	2	2	2
construct UNL	1s190ms	1s200ms	3.5s	15s	1m54s

Table III: *Query Q_2 = "jazz album artist! released!"*

This effect can be noticed more clearly in Table IV illustrating query Q_3 as we decrease the query selectivity even further. In this case, starting with $FST = 20$ would be a good idea.

	FST=20	FST=15	FST=10	FST=5	FST=0
# docs in U^{Q}	2314	2314	2314	2314	2314
# fields in U^{Q}	43	43	43	43	43
# features after consolidation	50	75	112	195	2533
$\|N\|$	927	1273	1750	2338	4325
$\|E\|$	2394	3189	4286	5285	8098
# roots	7	7	7	7	7
construct UNL	6s	12s	29s	1m	13m

Table IV: *Query Q_3 = "hard rock album released!"*

9. RELATED WORK

Recently, much effort went into the management of heterogeneous datasets enabling the average user to browse and query them. On one hand, there are the web search engines and ranked keyword search with heuristics over relational databases [Bhalotia et al. 2002; Agrawal et al. 2002; Hristidis and Papakonstantinou 2002; Hristidis et al. 2003; Balmin et al. 2004; Li et al. 2006] as well as over semistructrued data [Cohen et al. 2003; Li et al. 2004; Xu and Papakonstantinou 2005; Theobald et al. 2005]. Yet, they are not powerful enough to capture the user's intention in full [Vagena et al. 2007]. On the other hand, query languages such as SQL, XPath, XQuery, and SQL/XML require up-front data integration, are complex and hard to express. Moreover, these languages require a user to have full schema knowledge, which is not feasible for highly heterogeneous data. To cover the range between the two extremes, we identify a list of complementary techniques to our approach.

Extract structure from data. There has been increasing effort directed by various communities, including semantic web, data mining, and information retrieval, on designing tools to extract data and structure from heterogeneous collections and making it available for querying especially from the semantic web, the data mining and the information retrieval communities.

Freebase [Bollacker et al. 2007] supports collaborative structured information integration. However, it forces the users to follow a central predefined schema. , which is very limiting. DBpedia [DBpedia ; Auer and Lehmann 2007] is another effort, which also complements Wikipedia infoboxes with additional information. Nevertheless, the result is just as heterogeneous as the original. It can be accessed via keyword-based interfaces or SPARQL[SPARQL]. Other efforts focus on building and leveraging ontologies for search and browsing the data [Wu and Weld 2008; Suchanek et al. 2007; McDowell and Cafarella 2006]. To improve the accuracy of search results, Powerset [Powerset] brings in natural language processing. Yet, it fails to disambiguate query answers and it does not return aggregated information from multiple pages. Same note applies to WebTables [Cafarella et al. 2008], which proposes to leverage structured data in HTML tables and return ranked relations.

However, the shortcoming with the above approaches is that once the data is extracted complex queries need to be formulated over a strict predefined (virtual) large schema.

Sequential pattern mining. Another related body of work has focused on using lattice-based techniques [Ganter and Wille 1999] to extraction knowledge. These methods leverage the formal concept lattice (also called Galois lattice) [Ganter and Wille 1999] and vary by the way the lattice is constructed with different goals and functionalities. For example, [Zaki et al. 1997; Zaki 1998] deals with extraction of association rules by mining for frequent itemsets and sequences inside databases. However, there is no intention to capture the relation between the sets. Similarly, [Ganter et al. 2005; Godin et al. 1989; Grosser and Ralambondrainy 2007; Villerd et al. 2007; Carpineto and Romano 1996] use a lattice as an effective tool for navigation, concept extraction, and hierarchical conceptual clustering. However, these lattices encode the exploration space over all possible queries of terms. A query defines a position in this lattice. Browsing is similar to jumping between query formulations. It does not allow to disambiguate the original query other than by changing the set of terms in the query. In contrast, our lattice is built dynamically for each user query. It aims to disambiguate answers by navigating on the structure of the content, i.e., our features represent logical structural points in the collection where the keywords hit.

Semantic data exploration. On the other hand, semantic data exploration by traditional faceted search [Auer and Lehmann 2007; Tunkelang 2006; Dakka and Ipeirotis 2008; Yee et al. 2003] defines a predefined set of "facets" that are intrinsic properties of the data itself such as color, price etc. The number of facets per entity is usually manageable given that these engines are very domain specific. The entities in the dataset are then classified in hierarchical buckets according to facets. At search time, these systems browse the data and retrieve the entities over the predefined dimensions. In contrast, our dimensions are dynamic, defined on the fly, based on where the query keywords hit in the entity fields. In practice, for heterogeneous data this generates a large number of facets that would make interaction impractical using the existing faceted search engines. Moreover, faceted search is best suited for "point" queries while we focus on aggregation of information from multiple result records.

SEDA project [Balmin et al. 2009; Balmin et al. 2008] has the same goal as ours.

Nevertheless, the amount of heterogeneity inherent in the data makes it difficult to reason in terms of lists of structural dimensions of the data. WikiAnalytics uses a lattice-based approach, a more generic framework than lists, to efficiently and effectively organize the search space. Similarly, [Dash et al. 2008; Wu et al. 2007] focus on integrating dynamic faceted search with OLAP processing. However, they are not suitable for large heterogeneous data or designed to return a complete set of answers.

10. CONCLUSION

Large quantities of structured data are being created by online communities in wikis and other highly heterogeneous data sources. The need to query it has sparkled the interest to enable analysis and integration of Web data. In this paper we presented WikiAnalytics, a tool to support on-line ad-hoc querying over these data.

We demonstrate effective methods within a smart interactive user interface that facilitates exploration and disambiguation of search results in order to compile complete and precise answers that span multiple records or pages. Our methods and techniques are universally applicable to any collection of loosely typed records, which characterizes Web data in general. Our approach is to use heuristics that take advantage of the results' structure to improve precision and recall. At the same time we recognize that heuristics can never be 100% correct, so we built the WikiAnalytics tool that summarizes the results as a lattice of homogeneous result subsets. This tool enables the user to effectively verify and modify the results obtained by the heuristics.

We are currently investigating extensions of our techniques to capture more expressive data models, for instance allowing arbitrary XML fragments in record fields. We also want to consider hierarchical data dimensions in the attribute values, as well as to account for the links between documents.

REFERENCES

Agrawal, S., Chaudhuri, S., and Das, G. 2002. DBXplorer: a system for keyword-based search over relational databases. In *International Conference on Data Engineering (ICDE).*

Auer, S. and Lehmann, J. 2007. What have Innsbruck and Leipzig in common? Extracting semantics from wiki content. In *4th European Semantic Web Conference (ESWC'07).*

Balmin, A., Colby, L., Curtmola, E., Li, Q., and Ozcan, F. 2009. Search driven analysis of heterogenous XML data. In *Conference on Innovative Data Systems Research (CIDR).*

Balmin, A., Colby, L., Curtmola, E., Li, Q., Ozcan, F., Srinivash, S., and Vagena, Z. 2008. SEDA: A system for search, exploration, discovery and analysis of XML data. In *International Conference on Very Large Data Base (VLDB Demo).*

Balmin, A. and Curtmola, E. 2010a. WikiAnalytics: Ad-hoc Querying of Highly Heterogeneous Structured Data. In *International Conference on Data Engineering (ICDE Demo).*

Balmin, A. and Curtmola, E. 2010b. WikiAnalytics: Disambiguation of Keyword Search Results on Highly Heterogeneous Structured Data. In *International Workshop on the Web and Databases (SIGMOD WebDB).*

Balmin, A., Hristidis, V., and Papakonstantinou, Y. 2004. Authority-based keyword queries in databases using ObjectRank. In *International Conference on Very Large Data Base (VLDB).*

Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., and Sudarshan, S. 2002. Keyword searching and browsing in databases using BANKS. In *International Conference on Data Engineering (ICDE).*

Bollacker, K., Cook, R., and Tufts, P. 2007. A platform for scalable, collaborative, structured information integration. In *Intl. Workshop on Information Integration on the Web (II-Web'07)*.

Cafarella, M., Halevy, A., Wang, Z. D., Wu, E., and Zhang, Y. 2008. WebTables: Exploring the power of tables on the web. In *VLDB*.

Cammarano, M., Dong, X. L., Chan, B., Klingner, J., Talbot, J., Halevy, A. Y., and Hanrahan, P. 2007. Visualization of heterogeneous data. *IEEE Trans. Vis. Comput. Graph. 13,* 6, 1200–1207.

Carpineto, C. and Romano, G. 1996. A lattice conceptual clustering system and its application to browsing retrieval. In *Machine Learning, vol. 24*.

Cohen, S., Mamou, J., Kanza, Y., and Sagiv, Y. 2003. XSEarch: A Semantic Search Engine for XML. In *International Conference on Very Large Data Base (VLDB)*.

Dakka, W. and Ipeirotis, P. G. 2008. Automatic extraction of useful facet hierarchies from text databases. In *International Conference on Data Engineering (ICDE)*.

Dash, D., Rao, J., Megiddo, N., Ailamaki, A., and Lohman, G. 2008. Dynamic faceted search for discovery-driven analysis. In *Conference on Information and Knowledge Management (CIKM)*.

DBpedia. http://www.dbpedia.org.

Ganter, B., Stumme, G., and (Eds.), R. W. 2005. Formal concept analysis, foundations and applications. In *Lecture Notes in Computer Science, no. 3626, Springer-Verlag*.

Ganter, B. and Wille, R. 1999. Formal concept analysis: Mathematical foundations. In *Springer-Verlag*.

Godin, R., Gecsei, J., and Pichet, C. 1989. Design of a browsing interfaces for information retrieval. In *International ACM Conference on Reasearch and Development in Information Retrieval (SIGIR)*.

Grosser, D. and Ralambondrainy, H. 2007. Concept analysis on structured, multi-valued and incomplete data. In *5th International Conference on Concept Lattices and Their Applications (CLA'07)*.

Hristidis, V., Gravano, L., and Papakonstantinou, Y. 2003. Efficient IR-style keyword search over relational databases. In *International Conference on Very Large Data Base (VLDB)*.

Hristidis, V. and Papakonstantinou, Y. 2002. Discover: keyword search in relational databases. In *International Conference on Very Large Data Base (VLDB)*.

Li, F., Yu, C., Meng, W., and Chowdhury, A. 2006. Effective keyword search in relational databases. In *ACM International Conference on Management of Data (SIGMOD)*.

Li, Y., Yu, C., and Jagadish, H. V. 2004. Schema-Free XQuery. In *International Conference on Very Large Data Base (VLDB)*.

ManyEyes. http://manyeyes.alphaworks.ibm.com/manyeyes.

McDowell, L. K. and Cafarella, M. 2006. Ontology-driven information extraction with OntoSyphon. In *5th International Semantic Web Conference (ISWC'06)*.

Midas. Midas: Information Integration. In *http://www.almaden.ibm.com/cs/projects/integrationP/*.

Powerset. http://www.powerset.com/.

Simmen, D. E., Altinel, M., Markl, V., Padmanabhan, S., and Singh, A. 2008. Damia: data mashups for intranet applications. In *ACM International Conference on Management of Data (SIGMOD)*.

SPARQL. In *http://www.w3.org/TR/rdf-sparql-query/*.

Suchanek, F. M., Kasneci, G., and Weikum, G. 2007. Yago: A core of semantic knowledge. In *International World Wide Web Conference (WWW)*.

Swivel. http://www.swivel.com.

Theobald, M., Schenkel, R., and Weikum, G. 2005. An Efficient and Versatile Query Engine for TopX Search. In *International Conference on Very Large Data Base (VLDB)*.

Tunkelang, D. 2006. Dynamic category sets: An approach for faceted search. In *SIGIR'06 Faceted Search Workshop Program*.

Vagena, Z., Colby, L., Ozcan, F., Balmin, A., and Li, Q. 2007. On the effectiveness of flexible querying heuristics for XML data. In *XSym*.

Villerd, J., Ranwez, S., abd D. Carteret, M. C., and Penalva, J. M. 2007. Using concept lattices for visual navigation assistance in large databases: Application to a patent database. In *5th International Conference on Concept Lattices and Their Applications (CLA'07)*.

Wikipedia. In *http://en.wikipedia.org*.

Wu, F. and Weld, D. S. 2008. Automatically refining the Wikipedia infobox ontology. In *WWW*.

Wu, P., Sismanis, Y., and Reinwald, B. 2007. Towards keyword-driven analytical processing. In *ACM International Conference on Management of Data (SIGMOD)*.

Xu, Y. and Papakonstantinou, Y. 2005. Efficient Keyword Search for Smallest LCAs in XML Databases. In *ACM International Conference on Management of Data (SIGMOD)*.

YahooPipes. http://pipes.yahoo.com/.

Yee, K. P., Swearingen, K., Li, K., and Hearst, M. 2003. Faceted metadata for image search and browsing. In *ACM Conference on Computer-Human Interaction (CHI'03)*.

Zaki, M. J. 1998. Efficient enumeration of frequent sequences. In *Conference on Information and Knowledge Management (CIKM)*.

Zaki, M. J., Parthasarathy, S., Ogihara, M., and Li, W. 1997. New algorithms for fast discovery of association rules. In *KDD*.