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

Andrey Balmin
IBM Almaden Research Center
abalmin@us.ibm.com

Emiran Curtmola
UC San Diego
ecurtmola@cs.ucsd.edu

ABSTRACT

Wikipedia infoboxes is an example of a seemingly structured, yet extraordinarily heterogeneous 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 capture 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.

1. INTRODUCTION

Growing popularity of 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 [16, 11, 9, 24] 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 1) 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 and 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 2 conveys their heterogeneity. There are about $2,500$ distinct infobox types (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 $4,000$.

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

Structural diversity presents major problems when queries need 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 with 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.

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. 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. Conceptually, the features define the relevant dimensions on the data specifying the matching context for the query keywords. 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* [17] 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 work with. In a typical session, FST is initially set relatively high, 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. For example, query "California governor religion!", which is our running example throughout this paper, returns pairs of governors' (page) names and religious affiliations. The resulting feeds can be joined and aggregated by mashup tools like Yahoo Pipes[1] and Damia[25], or visualized by services like Swivel[2] and Many Eyes[3].

This paper focuses on the formalism and algorithms behind WIKIANALYTICS, in particular its clustering features and UNL. For more details on system architecture and user interaction, please refer to [5] as well as our extended version [6]. The rest of the paper is organized as follows. In the next section, we formalize WIKIANALYTICS data and query model. Section 3 describes clustering features while in Section 4, we define the Universal Navigational Lattice and give algorithms for its construction and visualization. Experimental results are shown in Section 5. Related work is discussed in Section 6 and we conclude in Section 7.

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.

[1] http://pipes.yahoo.com/
[2] http://www.swivel.com
[3] http://manyeyes.alphaworks.ibm.com/manyeyes

{{ **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]]
...}}

Figure 1: *Sample Wikipedia Infobox in WikiText format.*

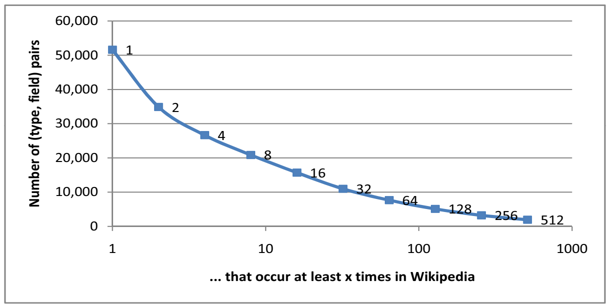
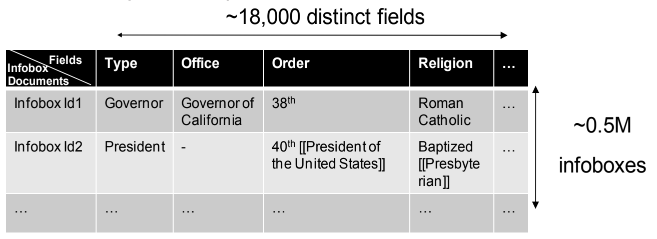


Figure 2: *Wikipedia Infoboxes are sparse: Distribution of fields per number of infobox instances in which they occur.*

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* [4]. *However, only 0.08% of its cells would have non-null values.* ◇


~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 [[Presbyterian]]	…
…	…	…	…	…	…

Figure 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 result as columns of U. For our running example above, 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* $Q = (k_1, \ldots, k_n)$ if $\forall k_i \in C(Q) \cup R(Q)$ then $k_i \in r$, and $\forall k_i \in N(Q)$ then $k_i \notin r$.

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

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.

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 $f \in Fld(r) \wedge f.value = s$.

Each feature uniquely determines a cluster of records that have this feature. Formally, we define a *feature association function* $\mathsf{C}^R : \{\mathsf{F}^{\mathsf{t}}, \mathsf{F}^{\mathsf{c}}, \mathsf{F}^{\mathsf{e}}\}^* \rightarrow \mathcal{R}$ that clusters a set of records R according to features. For instance, $\mathsf{C}^R(F)$ is a set of records that have feature F. For a given set of features, C^R associates the set of all records conforming with the features as follows:

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

Thus, $\mathsf{C}^R(F_1, \ldots, F_n) = \bigcap_i \mathsf{C}^R(F_i)$.

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 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 $\mathcal{F}^{\mathsf{t}}$ 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 are:

$$\mathcal{F}^{\mathsf{k}} = \mathcal{F}^{\mathsf{t}} \cup \mathcal{F}^{\mathsf{c}}(k) \cup \mathcal{F}^{\mathsf{e}}(k)$$

$$\mathcal{F}^{\mathsf{t}} = \{\mathsf{F}^{\mathsf{t}} : type = term | \forall term \in \Pi_{type}(\mathsf{U})\}$$

$$\mathcal{F}^{\mathsf{c}}(k) = \{\mathsf{F}^{\mathsf{c}} : k \in f | \forall j, f \in \mathsf{U}_{*,j} \wedge (k \in f.name \vee k \in f.value)\}$$

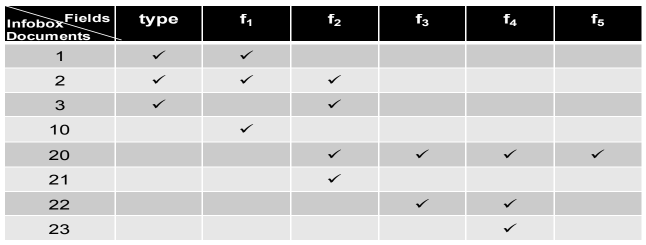
$$\mathcal{F}^{\mathsf{e}}(k) = \{\mathsf{F}^{\mathsf{e}} : f = f.value | \forall j, f \in \mathsf{U}_{*,j} \wedge (k \in f.name \vee \vee k \in f.value)\}$$

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 U^{Q} restricted to $Q(\mathcal{R})$ identifies only the sub-part of U that corresponds to records of $Q(\mathcal{R})$: $\mathsf{U}^{\mathsf{Q}} = \{\mathsf{U}_i | \forall i, r_i \in Q(\mathcal{R})\}$.* $\diamond$

Finally, we revisit the definitions for $\mathcal{F}^{\mathsf{k}}$ and $\mathcal{F}^{\mathsf{Q}}$ feature sets. 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 $Q(\mathcal{R})$.

EXAMPLE 3.1. *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 1 shows the matching features $\mathcal{F}^{\mathsf{Q}}$ together with their associated clusters $\mathsf{C}^{Q(\mathcal{R})}$ over U^{Q}.* $\diamond$



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

Figure 4: *Universal table U^{Q} for example 3.1.*

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 1: *Features and Clusters for Example 3.1*

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. 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.

4.1 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 6.

We capture the lattice as a direct acyclic graph (DAG) data structure, 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 $(n_1 \rightarrow n_2) \in E$ 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. This strategy avoids generating all combinations of nodes. Therefore, it benefits from pruning the nodes as early as possible if they do not agree with the definition of UNL, i.e. they correspond to duplicate or empty clusters of records.

The algorithm starts at lines 2-4 by generating the first lattice level, and possibly other nodes by consolidating the set of individual features $\mathcal{F}^{\mathsf{Q}}$, and their clusters $\mathsf{C}^{Q(\mathcal{R})}$, which were previously constructed by scanning the search engine results. The consolidation of features is required by UNL property (ii), as defined above. 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. Due to space constraints, please check out [6] for the pseudocode of supporting functions.

EXAMPLE 4.1. *Building on our running example, the initial feature sets are computed as in Table 1. After going through the consolidation process, the lattice consists of the following nodes $n_1 - n_6$ as displayed in Figure 5a. 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 $n_2.I = \{1, 2, 10\}$.* $\diamond$

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). For each such new potential node, the algorithm applies two pruning rules based on the properties of 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).

EXAMPLE 4.2. *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 5b. 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 5c. Similarly, this is observed for generating the combination between n_8 and n_3. Figure 5d shows the complete lattice graph where new nodes get generated for unique feature sets and unique clusters, or just merged with existing nodes.* $\diamond$

Note that UNL encodes all sets of features even though not every set corresponds to a node. For example, $\{F_1, F_2\}$ is not a node but the set corresponds to $n_7.I$ since $n_7.F$ is their smallest superset.

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.

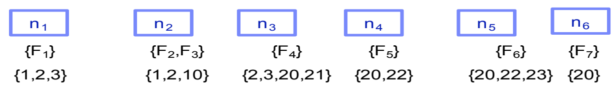
Function remove_generalized_triangles computes a set of indirect ancestors for each node n – the ancestor nodes of n excluding its direct parents. We do this in one pass over the lattice by employing breadth first search graph traversal starting from the roots (nodes without parents) of the lattice graph 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 remove from E.

EXAMPLE 4.3. *Our sample lattice graph contains four such redundant links closing generalized triangles. They are shown with dotted red lines in Figure 5e. 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.* $\diamond$

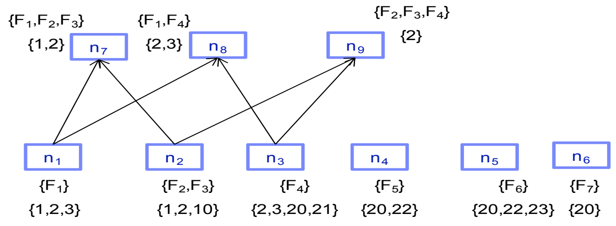
4.2 Presentation of UNL

UNL lends itself to many ways of presentation and user interaction, such as faceted search or OLAP style slicing and dicing. We chose tree-based presentation approach, similar to directory folders, where 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.

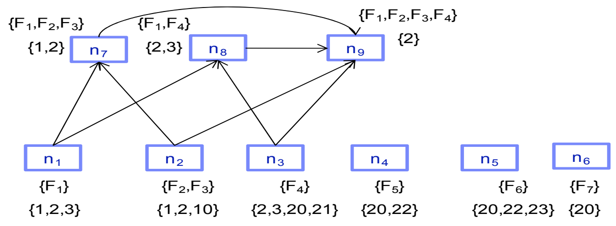
We obtain the tree shape by a depth-first traversal of UNL, starting from each of its root nodes. In practice, most of the lattice nodes are well connected so there are relatively few roots. For every node, its children are ordered in the descending order of the cluster size. A sample user interface is explained in more details in [6].



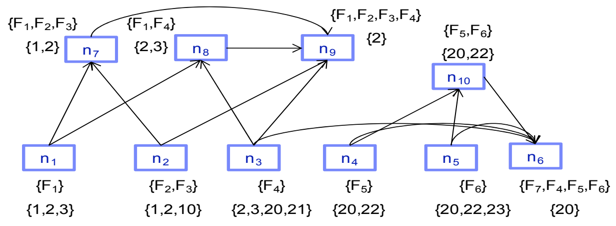
(a) *Initial lattice nodes after consolidation of* $\mathcal{F}^{\mathsf{Q}}$.



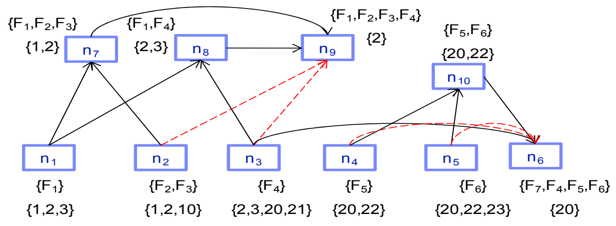
(b) *Generating nodes* n_7, n_8, *and* n_9 *using rule* R_1.



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



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



(e) *Removing redundant links from* UNL.

Figure 5: *Construction of* UNL *Universal Navigational Lattice Graph.*

5. 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 [23].

We ran the WIKIANALYTICS system on a Centrino Duo 2.2GHz laptop with 2GB of RAM. The tables below show how the lattice parameters vary with FST for queries with different selectivities on Wikipedia. Table 2 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.

To stress WIKIANALYTICS, we have focused on less selective queries. For instance, Q_2 as shown in Table 3 matches to approximately 1,000 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 3 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.

```
algorithm compute_unl($\mathcal{F}^{Q}$, $C^{Q(\mathcal{R})}$)
input: features $\mathcal{F}^{Q}$, clusters $C^{Q(\mathcal{R})}$
output: universal navigational lattice UNL
begin
1  $N = \{\emptyset\}, E = \{\emptyset\}$
2  $initial\_nodes$ = consolidate_features($\mathcal{F}^{Q}$, $C^{Q(\mathcal{R})}$)
3  for each node $n \in initial\_nodes$ do
4    $N += n$
5  for each level $level = 2..|\mathcal{F}^{Q}|$ 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
```

Figure 6: UNL *construction algorithm*

	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
$\vert N\vert$	19	35	53	79	151
$\vert E\vert$	31	66	106	163	278
# roots	1	1	1	1	1
construct UNL	2ms	5ms	9ms	27ms	163ms

Table 2: *Query* Q_1 = *“California governor religion!”*

6. 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 [8, 2, 20, 19, 7, 21] as well as over semistructrued data [13, 22, 32, 27]. Yet, they are not powerful enough to capture the user’s intention in full [29]. 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. To cover the range between the two extremes, we identify a list of complementary techniques to ours.

	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 3: *Query Q_2 = "jazz album artist! released!"*

There is an increasing effort in designing tools to extract data and structure from heterogeneous collections and making it available for querying from various communities. Freebase [9] supports collaborative structured information integration. However, it forces the users to follow a central predefined schema. DBpedia [16, 3] 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[1]. Other efforts focus on building and leveraging ontologies [30, 26]. To improve the accuracy of search results, Powerset [24] brings in natural language processing. Yet, it fails to disambiguate query answers and it does not return aggregated information from multiple pages. Same applies to WebTables [10], which proposes to leverage structured data in HTML tables and return ranked relations.

Another related body of work has focused on using lattice-based techniques [17] to extraction knowledge. For example, [33] deals with extraction of association rules by mining for frequent itemsets and sequences inside databases. Similarly, [18, 12] use a lattice as an effective tool for hierarchical conceptual clustering. However, these lattices encode the exploration space over all possible queries of terms. Browsing is similar to jumping between query formulations. 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.

On the other hand, semantic data exploration by traditional faceted search [3, 28, 14] 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. 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. [15, 31, 4] 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.

7. CONCLUSION

Large quantities of structured data are being created by online communities in wikis and other highly heterogeneous data sources. In this paper we presented WIKIANALYTICS, a tool to support online 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.

8. REFERENCES

[1] Sparql. In *http://www.w3.org/TR/rdf-sparql-query/*.

[2] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: a system for keyword-based search over relational databases. In *ICDE*, 2002.

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

[4] A. Balmin, L. Colby, E. Curtmola, Q. Li, and F. Ozcan. Search driven analysis of heterogenous XML data. In *CIDR*, 2009.

[5] A. Balmin and E. Curtmola. WIKIANALYTICS: Ad-hoc Querying of Highly Heterogeneous Structured Data. In *ICDE Demo*, 2010.

[6] A. Balmin and E. Curtmola. WIKIANALYTICS: Disambiguation of keyword search results on highly heterogeneous structured data. In *IBM Research Report RJ 10466*, May 2010.

[7] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Authority-based keyword queries in databases using ObjectRank. In *VLDB*, 2004.

[8] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.

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

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

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

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

[13] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A Semantic Search Engine for XML. In *VLDB*, 2003.

[14] W. Dakka and P. G. Ipeirotis. Automatic extraction of useful facet hierarchies from text databases. In *ICDE*, 2008.

[15] D. Dash, J. Rao, N. Megiddo, A. Ailamaki, and G. Lohman. Dynamic faceted search for discovery-driven analysis. In *CIKM*, 2008.

[16] DBpedia. http://www.dbpedia.org.

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

[18] R. Godin, J. Gecsei, and C. Pichet. Design of a browsing interfaces for information retrieval. In *SIGIR*, 1989.

[19] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB*, 2003.

[20] V. Hristidis and Y. Papakonstantinou. Discover: keyword search in relational databases. In *VLDB*, 2002.

[21] F. Li, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, 2006.

[22] Y. Li, C. Yu, and H. V. Jagadish. Schema-Free XQuery. In *VLDB*, 2004.

[23] Midas: Information Integration. In *http://www.almaden.ibm.com/cs/projects/integration/*.

[24] Powerset. http://www.powerset.com/.

[25] D. E. Simmen, M. Altinel, V. Markl, S. Padmanabhan, and A. Singh. Damia: data mashups for intranet applications. In *SIGMOD*, 2008.

[26] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A core of semantic knowledge. In *WWW*, 2007.

[27] M. Theobald, R. Schenkel, and G. Weikum. An Efficient and Versatile Query Engine for TopX Search. In *VLDB*, 2005.

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

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

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

[31] P. Wu, Y. Sismanis, and B. Reinwald. Towards keyword-driven analytical processing. In *SIGMOD*, 2007.

[32] Y. Xu and Y. Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In *SIGMOD*, 2005.

[33] M. J. Zaki. Efficient enumeration of frequent sequences. In *CIKM*, 1998.