Navigating Extracted Data with Schema Discovery

Michael J. Cafarella
University of Washington
Seattle, WA 98195
mjc@cs.washington.edu

Dan Suciu
University of Washington
Seattle, WA 98195
suciu@cs.washington.edu

Oren Etzioni
University of Washington
Seattle, WA 98195
etzioni@cs.washington.edu

ABSTRACT

Open Information Extraction (OIE) is a recently-introduced type of information extraction that extracts small individual pieces of data from input text without any domain-specific guidance such as special training data or extraction rules. For example, an OIE system might discover the triple **Frenzy, year, 1972** from a set of documents about movies. Because OIE is domain-independent, it promises to help users when they have a corpus of structured data, but that structure is unknown, such as when browsing a novel domain or formulating a query. We can describe the structure to the user by displaying a relational schema that fits the extracted data.

Unfortunately, the extractions do not carry full schema information: we have extracted values, but not the correct relations, their rows, or their columns. In response we propose TGen, an algorithm for schema discovery, which automatically derives a high-quality relational schema for the extracted data. Different applications have different schema-design requirements, which can be encoded as input to TGen. We show that our data-mining approach runs in minutes on millions of documents while still resulting in schemas that are useful for exploring unfamiliar data or for composing queries over extracted data.

1. INTRODUCTION

Open Information Extraction (OIE), introduced by Banko, *et al.*, is a *domain-independent* model of information extraction, in which the sole system input is a collection of texts and the output is a set of extracted triples [5]. IE systems have traditionally required domain-specific guidance ahead of time, for example in the form of a target database schema, hand-written extraction rules, or specially-chosen training data [16, 13, 1]. They have been quite successful in populating databases with information from unstructured text, but they have only been applied in situations where the schema is known in advance.

In many cases, especially with extracted Web data where there is a vast number of domains and no central administrator, users must manage data where the appropriate schema is unknown. Indeed, learning about the data's structure is sometimes the user's main goal, as when exploring a novel data set or when composing a query. Since relational schemas are the best-known and most widely-used technique for describing structured data, it seems natural to present the extracted information in relational form.

Schema discovery is the problem of constructing a relational schema that best describes the extracted data. Schema design is a well-studied area, but instead of minimizing data duplication or maximizing representational efficiency, we are interested in the schema as a tool for data understanding. To motivate the problem of schema discovery for navigation, we describe two concrete applications in which the user is unfamiliar with the data's structure but needs to learn about it. The schema should depend on not only on the data, but also on application-specific design preferences.

1.1 Structured Data Browsing

In structured data browsing, the user wants to see the most important and interesting subdomains within a set of extracted data. The user can indicate a rough topic area with, say, a search engine keyword query. TGen then operates over the extractions from the resulting pages and displays the resulting schema. To the user, a structured data browsing application should behave like a *keywords-in, structured-answer-out* search engine. For example, a keyword query for `nutrition` should display relations representing drugs, vitamins, vegetables, etc.

There are a few desirable qualities in a good relation for structured data browsing. It should have a good number of related attributes; a relation with a single attribute is not likely to be interesting, even if it contains every single extraction. However, a relation with a vast number of attributes cannot easily be browsed on-screen. Further, a relation that contains just one extraction is not interesting, though it is not necessary to contain a vast number. Finally, the relation should not contain too many NULLs, or else browsing it will be monotonous.

To produce an application-appropriate schema, our TGen algorithm requires that the application provides:

- Preferences for relations with certain numbers of attributes (*i.e.*, "width")
- Preferences for relations with certain numbers of extractions (*i.e.*, "height")
- The relative importance of filled versus NULL values

Proceedings of the 10th International Workshop on Web and Databases (WebDB 2007), June 15, 2007, Beijing, China

(*i.e.*, "density")

1.2 Query Formulation

In the query formulation scenario, the user also examines a schema for an unfamiliar domain, this time to select useful attributes for a query. For example, consider someone who wants to use nutrition extractions to find all items that contain magnesium. Formulating a query involves first finding the best attribute name from among, *e.g.*, **contains**, **has**, or **is-packed-with**. The user can compare the attributes and then issue the query, *e.g.*, **contains = magnesium**.

For query formulation, the usefulness of a given attribute depends substantially on how many extractions exhibit that attribute. It would be interesting to find a few tightly-related attributes that appear together in many extractions, but in general the number of supporting rows is more important. In general, a good relation for query formulation is likely to have more "height" and less "width" than one for data browsing.

1.3 Contributions

The contributions of this paper are as follows:

1. We propose the problem of schema discovery for extracted data, motivated by the problems of *structured data browsing* and *query formulation*. Unlike previous work in schema discovery for relational data, we are concerned with the schema as an application-driven tool for navigating the data, not in pursuing traditional schema design goals (such as minimizing data duplication).

2. We propose TGen, an algorithm that automatically generates schemas from extracted data. It incorporates application-specific schema design requirements. We show that schemas generated by TGen are useful for the above unstructured applications.

The remainder of this paper is organized as follows. We cover the precise problem definition in Section 2, and describe the algorithm in Section 3. Experimental results are presented in Section 4 and we conclude with related work in Sections 5 and 6.

2. PROBLEM DEFINITION

In this section we formalize the schema discovery task. We describe a formal Open Information Extraction model, discuss a sample schema discovery task for OIE output, and finally cover the precise schema discovery inputs, outputs, and schema scoring function.

2.1 Open Information Extraction Model

IE systems have traditionally relied on domain-specific human guidance. The KnowItAll system [13] required hand-written extraction rules for each relation, Mansuri and Sarawagi's system [16] required an existing database to populate, and many systems have relied on hand-chosen training samples [6, 1].

Wrapper induction systems, which attempt to extract information from database-backed documents have also employed training data [15, 19]. Some wrapper induction systems, like IEPAD [8], ROADRUNNER [11], and EXALG [4], use pattern discovery to extract likely database values, and so have avoided the use of training data. However, they do not attempt to find a label for each extracted value, forcing a human to later mark the different attributes.

1. Create table set $\mathbf{S} = \{\}$
2. For each input row r, let $Attr(r)$ be the set of its attributes.
3. If there exists a table $T \in \mathbf{S}$ s.t. $Attr(T) = Attr(r)$ then assign r to T.
4. Otherwise, create a new table T with attributes $Attr(T) = Attr(r)$, assign r to T and insert T in $\mathbf{S}$.

Figure 1: The NAÏVE algorithm constructs a database from extracted data.

	Title	Year	Length	Filmtype
r_0	Amelie	2001	129	color
r_1	Babe	1995	89	color
r_2	Cocktail	1988	104	color
r_3	Dracula	1931	75	bw
r_4	Evita	1996	134	color
r_5	Frenzy	1972	116	color
r_6	Gaslight	1944	114	bw
r_7	Hamlet	1990	242	color
r_8	Indiscreet	1958	100	color
r_9	John Q	2002	116	color

Figure 2: This is the database constructed by NAÏVE when the source data is perfect and uncorrupted.

Open IE systems do not require domain-specific assistance, and can thus extract data when the target domain is unknown. The recent TEXTRUNNER system uses a combination of linguistic processing, a self-trained classifier, and frequency counts to extract domain-independent *entity, relationship, entity* triples from text [5].

A good OIE data model would not only capture TEXTRUNNER output but also that of non-open IE systems, for use when there happens to be a domain-specific extractor available, or for when a traditional system can be adapted for domain-independent use (*e.g.*, it might learn extraction rules automatically). We assume the extraction system emits a set of *unstructured tuples*. Each unstructured tuple consists of a key that uniquely identifies the tuple, plus some associated attribute/value pairs. The key can be a natural-language string (*e.g.*, a person's name), can be a synthetic value computed from the source document (*e.g.*, a record identifier for a movie review), or could even be the result of a reference-reconciliation step. For example, by running an extractor over a posting on a movie review site, we might produce as its key the record identifier r, with a number of associated pairs: `title/Frenzy`, `filmtype/color`, and `year/1972`. Values are non-probabilistic, so any probabilities emitted by the extractor are thresholded prior to processing.

In some cases the correct label for an extracted value will not be clear. For example, it is easy to recognize a phone number using a regular-expression-based extractor, but it may not be obvious whether the value is a `home-phone` or an `office-phone`. We allow the IE system to communicate this ambiguity to the schema discovery system. An extracted value can be emitted with one or more possible labels.

By placing the tuple in a relational schema, the schema discovery system may be able to choose the correct attribute. For example, the set of tuples about corporations may have

	Title	Year	Filmtype
r_0	Amelie	2001	color

	Title	Year	Length	Filmtype
r_1	Babe	1995	89	color

	Title	Year	Filmtype	Noise-4
r_2	Cocktail	1988	color	noiseval

	Length
r_3	75

	Year	Length	Filmtype	Noise-2
r_4	1996	134	color	noiseval

	Title	Filmtype	Length	Noise-4	Noise-5
r_5	Frenzy	116	color	noiseval	noiseval

	Title	Year	Length	Filmtype	Noise-1
r_6	Gaslight	1944	114	bw	noiseval

	Title	Year	Length
r_7	Hamlet	1990	242

	Year	Filmtype	Length	Noise-5
r_8	1958	color	100	noiseval

	Title	Year	Length	Filmtype	Noise-2	Noise-3
r_9	John Q	2002	116	color	noiseval	noiseval

Figure 3: This database was also created by NAÏVE, but the input data has now been corrupted with delete probability 0.2 and add probability 0.2. The schema is confusing and useless for understanding the domain.

many examples of `office-phone` but very few of `home-phone`. In the case where the labelling is ambiguous, the schema discovery system can choose `office-phone`, so that the tuple matches the others.

2.2 Challenges

Unfortunately, IE-derived data is inevitably noisy. Input documents are written by humans, who naturally include many off-topic facts or miss relevant ones. The extractors themselves similarly introduce spurious facts and miss good ones.

Consider a user who has a set of extracted data about *movies* and is using a structured data browser to learn more about the domain. A sample extraction might be **r_0, Length, 129**, indicating that the movie r_0 has a running length of 129 minutes. We could assemble these extractions into a database using the NAÏVE algorithm, described in Figure 1. If we have a sample set of 40 perfectly-extracted triples, covering 4 attributes of 10 movies, the result is the single-table schema seen in Figure 2. This schema obviously describes the domain quite effectively.

However, when we delete 20% of these triples at random and add an equal percentage of spurious extractions, the output of NAÏVE can be seen in Figure 3. In contrast, this database is close to useless. There is only one underlying set in this data, but each row is in a different relation. Spurious (and rare) columns appear alongside the true (and common) ones.

The database in Figure 4 would be a better choice. All of the relevant attributes are present, and none of the spurious ones. This single table contains almost all the rows[1], but it also contains less of the extracted data and more NULLs than Figure 3. Overall, the database from Figure 4 is a better tool for users unfamiliar with the domain.

2.3 Inputs and Outputs

[1]All were included but r_3, **Dracula**, which had just one field survive the corruption step.

	Title	Year	Length	Filmtype
r_0	Amelie	2001	NULL	color
r_1	Babe	1995	89	color
r_2	Cocktail	1988	NULL	color
r_4	NULL	1996	134	color
r_5	Frenzy	NULL	116	color
r_6	Gaslight	1944	114	bw
r_7	Hamlet	1990	242	NULL
r_8	NULL	1958	100	color
r_9	John Q	2002	116	color

Figure 4: This database was created by TGEN running on the same corrupted data sent to NAÏVE in Figure 3. There are some unavoidable NULL values, as some values were deleted as part of the corruption step. Unlike the NAÏVE database, the schema accurately shows that all extracted data rows in this domain are members of the same set. Tuple r_3 has been dropped, but so have all of the 8 spurious values added by the corruption step.

The schema discovery algorithm receives as input a set of unstructured tuples **U**. To illustrate, consider two possible unstructured tuples from the above example:

```
Dracula [movie (1), monster (2)]
1931 [year (1)]
75 [length (1)]
color [bw (1)]

Evita [title (1), person (1)]
1996 [year (1)]
134 [length (1)]
color [filmtype (1)]
```

The OIE system has correctly extracted the value `Dracula`, with two possible attribute names: `movie` or `monster`, with the former being more preferable. For the second tuple, the extraction for `Evita` has two possible labels, each equally preferred.

In general an unstructured tuple U can be described as

$$U = (Key, \{V_1, V_2, \ldots\})$$

where Key is a unique key (here, r_3 and r_4) and each V_i is a *value description*, consisting of a value v and a set of (label, preference)-pairs:

$$V_i = (v, \{(L_1, P_1), (L_2, P_2), \ldots\})$$

where v is the actual value, L_i is a candidate label for that value, and P_i is its preference rank.

The input to the schema-discovery system is the set of all input tuples, **U**, and the set of all labels occurring in all tuples, denoted **L**.

The output is a *schema* **S**, and an *assignment* **A**. The schema is a set of table schemas $\mathbf{S} = \{T_1, T_2, \ldots\}$, where each table schema is defined as a set of attributes from **L** [2]. Thus, each T_i can be identified with a set of attributes, $Attr(T_i)$, s.t. $Attr(T_i) \subseteq \mathbf{L}$. The assignment, **A**, is a many-to-many relationship between the unstructured tuples in **U**

[2]We do not currently attempt to generate names for the output tables.

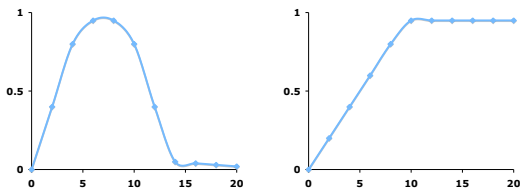

Figure 5: The structured browser width function $w()$, at left, should peak at an easily-displayed 6-8 attributes; the graph pictured runs from 0 to 20 attributes on the x-axis and $w()$ between 0 and 1 on the y-axis. The height function $h()$, at right, counts rows on the x-axis. Ten rows are enough to show that the relation is interesting, but more are not necessary.

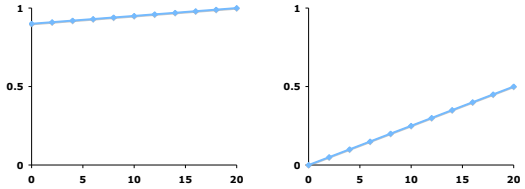

Figure 6: Relation width is substantially irrelevant to query formulation (except for near-synonyms), so $w()$ (left) has only a small preference for wider relations. (We use the same axes as in Figure 5.) In contrast, $h()$ (right) climbs steadily as the number of rows in a relation increases, indicating that popular attributes are the most important ones.

and the tables in $\mathbf{S}$, *i.e.*, it consists of a set of pairs (U, T), where $U \in \mathbf{U}$ and $T \in \mathbf{S}$.

An unstructured tuple may be assigned to more than one table (*e.g.*, by partitioning the tuple by attribute), and, of course, one table may contain more than one tuple. Normally we expect that an unstructured tuple U will be assigned to a table T if U and T share many attributes, but we make no such requirements in the problem definition. A tuple U may be assigned to a table T even if they share few attributes (or none at all!), and it may be possible for some values of an unstructured tuple to remain unassigned to any table.

Once an unstructured tuple U is assigned to a table T, the following structured row r is inserted into T: (a) $r.Key = U.Key$, (b) for every attribute $L \in Attr(T)$, if U has some value v with an associated label L, then $r.L = v$; if several values $v, v', \ldots$ have associated label L, then the most-preferred one is picked, (c) if no value in U has label L, then $r.L =$ NULL. Note that it is possible to have no single most-preferred labelling for a row, in which case we choose a labelling at random.

2.4 The Solution Score

We measure the quality of the system's output $(\mathbf{S}, \mathbf{A})$ with a schema design scoring function $\mathtt{f}(\mathbf{S}, \mathbf{A})$. This score includes a measure of how $(\mathbf{S}, \mathbf{A})$ fit the input data $\mathbf{U}$ and the application-centric design preferences.

We first define the data-centric score, which gives higher scores to schemas that have the most values, the fewest NULLs, and which best fit the application's width and height design preferences. The weighting between these factors, of course, is an input to the system. This score is given by independent contributions of each table in $\mathbf{S}$, and each row in each table, as follows.

For a table $T \in \mathbf{S}$, and each row $r \in T$, let $v(r)$ be the number of non-null values in r, and let $n(r)$ be the number of null values in r: thus $v(r)+n(r) = |Attr(T)|$. We consider it generally beneficial to include more values in the solution, and bad to include NULLs. Thus, we define a row's score to be:

$$\mathtt{fr}_T(r) = v(r) - w_N \cdot n(r) \tag{1}$$

where w_N is a user-chosen weight indicating the penalty we assign to a NULL compared to the reward for a non-NULL value. To indicate that a reasonable minimum density of 2 true data values for each NULL, we would use $w_N = 0.5$.

Second, we define the score for the entire table T as the product of three terms: the row-score "density" (*i.e.*, the table sum of row scores normalized by the number of data cells), and two user-chosen functions, $w()$ and $h()$. These functions receive as input, respectively, the number of attributes and the number of rows:

$$\mathtt{ft}_{\mathbf{S},\mathbf{A}}(T) = \frac{\sum_{r \in T} \mathtt{fr}_T(r)}{|Attr(T)||Rows(T)|} w(|Attr(T)|)h(|Rows(T)|) \tag{2}$$

A naïve choice of the functions $w()$ and $h()$ is $w(|Attr(T)|) = |Attr(T)|$ and $h(|Rows(T)) = |Rows(T)|$. In that case, the score of a table is simply the sum of its row scores. For certain applications, however, users may have better choices for $w()$ and $h()$. For the structured data browser, we might prefer the $w()$ and $h()$ functions in Figure 5. For query formulation, the functions in Figure 6 would probably be better.

Third, we define the overall score function:

$$\mathtt{f}(\mathbf{S}, \mathbf{A}) = \sum_{T \in \mathbf{S}} \mathtt{ft}_{\mathbf{S},\mathbf{A}}(T) \tag{3}$$

We can now model the schema discovery problem as one of score-optimization. Consider that the input data has an input set of rows $\mathbf{U}$ and unique attribute labels $\mathbf{L}$. For each possible table, each attribute may be present or not present; thus there are $2^{|\mathbf{L}|}$ possible unique tables. Each table may contain a row or not, so there are $2^{|\mathbf{U}|}$ possible row assignments to each table. That means we must find the highest-scoring $(\mathbf{S}, \mathbf{A})$ from among $2^{|\mathbf{L}|+|\mathbf{U}|}$ possibilities.

Both $\mathbf{U}$ and $\mathbf{L}$ can be large. $\mathbf{U}$ is as large as the unstructured tuple input set, which can easily extend into tens of thousands when extracted from just a few million documents. The set of unique attributes is slightly harder to predict. The input data is expected to have enough commonality to enable a useful schema, but there may be a large number of eccentric attribute labels. It would not be surprising to see several thousand in $\mathbf{L}$ (indeed, we see 13,377 labels in just the nutrition dataset described in Section 4). Thus the overall space of possible outputs is enormous.

The schema discovery problem is: given unstructured tuples $\mathbf{U}$ and scoring inputs w_N, $w()$, and $h()$, find the out-

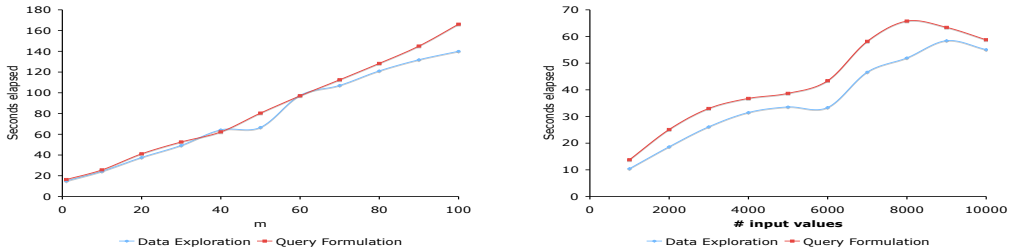

Figure 7: The left-hand image shows performance as we vary the size of the memory buffer m between 1 and 100. The right-hand shows performance as we vary the number of input values from 1000 to 10,000. In both cases, runtime for TGen grows roughly linearly.

puts $(\mathbf{S}, \mathbf{A})$ with the largest score $\mathtt{f}(\mathbf{S}, \mathbf{A})$. Not surprisingly, schema extraction is an NP-hard problem, as seen by reduction from the set-cover optimization problem.

Definition 2.1. *The set cover optimization problem is the following. Given a set U (called "universe"), a set S of subsets of U, find the subcollection $C \subset S$ that uses the smallest number of sets whose union is U.*

Using the fact that set cover optimization is NP-hard [10] we can show:

Theorem 2.2. *The schema extraction problem is NP-hard.*

3. TGEN

TGen is a data-mining style algorithm that takes as input the set of unstructured tuples $\mathbf{U}$ and repeatedly computes the single best table to add to $\mathbf{S}$. After a table is computed for $\mathbf{S}$ and the assigned value extractions are added to $\mathbf{A}$, all of the assigned values are removed from $\mathbf{U}$ before the next table computation. The algorithm terminates when the application does not require any more tables (*e.g.*, because the interface cannot display any more).

Computing a single best table from input $\mathbf{U}$ and user-chosen parameter m (the memory buffer size) works as follows:

1. Initialize $k = 0$, $\mathbf{T}_0 = \{T_{NULL}\}$ where T_{NULL} is the table with no attributes.
2. Starting at iteration $k = 1$, repeat the following:
   (a) Create a hypothesis set $\mathbf{T}_k$ of m potential table schemas, each with k attributes, as follows. For each T in $\mathbf{T}_{k-1}$, extend T in all possible ways with one more attribute. Prune to size m by choosing the m tables with the highest predicted score, using attribute co-occurrence statistics to perform predictions.
   (b) For each tuple $u \in \mathbf{U}$, for each $T \in \mathbf{T}$, test to see if u would result in a positive $\mathtt{fr}_T(u)$ score. If so, we add the score to the $\mathtt{ft}(T)$ computation.
   (c) Compute the final score for each $T \in \mathbf{T}$ and store the highest-scorer so far.
   (d) Increment k; exit when k reaches a maximum allowable table size.

The number of potential tables is exponential in $|\mathbf{L}|$, so we can rarely consider every possible table schema. We rely on the attribute statistics heuristics to choose $\mathbf{T}$ effectively; if a table T is not in the m set of hypothesis tables, it will never even be tested against the data and will never be emitted. Pruning can be inaccurate, so a larger m makes it likelier that high-quality tables will be available in the hypothesis set, but at the cost of running times and memory use.

Note that even if the highest-scoring table is present in $\mathbf{T}$, we may not find the correct assignment of tuples for it. We allocate the assignment using only local information about the score $\mathtt{fr}_T$, but even a positive value could lower the overall table score when the user-chosen interface functions are incorporated.

4. EXPERIMENTAL RESULTS

Most of our experiments were carried out on a set of **nutrition**-oriented extractions, obtained by running the TextRunner extractor over a set of 2.5 million web pages gathered from a focused web crawl. We seeded a breadth-first crawl by first picking a number of classes in the domain, then using WordNet [17] to find instances of those classes, and then performing random search queries using these instances to obtain the seeds. To ensure high-quality extractions, we removed all those that occurred fewer than 3 times, leaving 12,575 unstructured tuples and 13,377 unique attributes. The universal table for this data would be very sparse, with just 55,249 extracted values in a table of over 168M cells.

As long as TGen has a nontrivial memory buffer m, it is successful at obtaining tables that match the given interface constraints. When $m = 100$, the data browsing problem on nutrition data elicits tables with a median of 21.5 rows and 6 attributes, and an average of 25.2 and 5.5, respectively. These match the given $w()$ and $h()$ functions very closely. For the query formulation problem, we obtain only single-attribute columns, strictly sorted by the number of rows that have a value for the relevant column.

Performance for TGen is roughly linear in both the size of the memory buffer parameter and the input values, as seen in Figure 7. Runtimes on the order of minutes are not fast enough for web-search-style interactivity, but are reasonable for learning a domain for the first time.

How well does the actual schema output help with the applications? Figure 8 shows a summary of the top-10 relations when TGen is run on the nutrition dataset for the data

Rank	Topic	# Rows	Frac. good	Example Rows	# Attrs	Frac. good	Example Attrs
1	nutrients/minerals	25	0.84	vitamin c, thiamine	6	1.0	is found in, is essential for
2	health actors	10	0.80	adults, men	6	0.67	take, often have
3	*info-related words*	*14*	*1.0*	*this report, this site*	*6*	*1.0*	*is provided as*
4	research-related objects	24	0.96	conference, program	6	0.67	was supported by
5	foods	36	0.94	soybeans, carrots	6	1.0	are low in, are high in
6	health authorities	21	0.86	health officials, fda	4	1.0	said, warned
7	diseases	71	0.73	influenza, lyme disease	5	1.0	is spread by, is caused by
8	*info-related words*	*16*	*0.95*	*this lesson, this unit*	*5*	*1.0*	*presents, was published in*
9	*no clear topic*	*16*	*N/A*	*the sun, the animal*	*6*	*N/A*	*is under, was in*
10	organs	13	0.92	the pancreas, the skin	5	1.0	secretes, produces

Figure 8: Overview of the top-10 relations emitted by TGen on the nutrition dataset with $m = 100$ for the data browsing task. Rows and attributes were checked by hand to see if they are specific (*e.g.*, "people" is too vague) and on-topic. The relations in italics, at rank 3, 8, and 9, contain rows that are not specific to nutrition or do not have an obvious topic. We might remove these relations with an improved extractor or by collecting statistics about extractions that frequently occur outside the current corpus.

browsing task. As mentioned above, it generates a series of single-attribute relations when run on the query formulation task. While three of the top-10 relations are not helpful, the remaining seven are on-topic and closely match the application's design preferences.

5. RELATED WORK

There is extensive related work in the area of information extraction, discussed above in Section 2.1.

The field of schema matching and creation contains a substantial amount of work, but most of it has focused on clean data from traditional databases. Schema matching systems have used a number of different structure-matching techniques [7, 20, 21]. Doan, *et al.* examined the data as well as the structural elements [12]. Miller, *et al.* described "schema discovery" for data without any given schema, suggesting a domain-independent clustering approach [18]. TGen, in contrast, uses only extractions from text; it has no structured inputs at all. Because the inputs are assumed to be quite noisy, and the result is not designed for transactional use, TGen outputs a database that is probably "dirtier" than most traditional schema matchers could accept.

The XTRACT system attempted to find a high-quality DTD for a set of clean XML documents, choosing the candidate DTD that is both small in size without overgeneralizing beyond the inputs [14]. Its tradeoff between DTD concision and precision is somewhat similar to the competing schema design criteria embodied in the scoring function from Section 2.4.

The motivation for our work is probably closest to that of Cong and Jagadish, who created compact schema summaries that describe much more complicated schemas [9]. However, in our case there is no preexisting schema to compress (and to constrain the space of outputs), merely a set of noisy extracted triples.

The Apriori algorithm attempts to learn association rules from a database of itemsets by repeatedly generating and then testing a set of candidate rules with a pass through the database [2, 3]. TGen uses the same execution model, computing candidate tables and testing them with a pass through the unstructured tuple data. However, unlike Apriori's association rules, a table in TGen can accrue score from a tuple even when the tuple only supports some of the table's attributes (in short, TGen must allow NULLs in the table).

6. CONCLUSIONS

Schema discovery for data navigation is a novel problem whose solution enables two very helpful applications for managing unstructured data (now found in abundance on the Web). TGen is a promising start for a schema discovery system.

7. REFERENCES

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *Procs. of the Fifth ACM International Conference on Digital Libraries*, 2000.

[2] R. Agrawal, T. Imielinksi, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB 1994*, 1994.

[4] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *SIGMOD Conference*, pages 337–348, 2003.

[5] M. Banko, M. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *Procs. of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 2007.

[6] S. Brin. Extracting Patterns and Relations from the World Wide Web. In *WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT'98*, pages 172–183, Valencia, Spain, 1998.

[7] S. Castano and V. Antonellis. A schema analysis and reconciliation tool environment for heterogeneous databases. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS-99)*, pages 53–62, 1999.

[8] C.-H. Chang and S.-C. Lui. Iepad: information extraction based on pattern discovery. In *WWW*, pages 681–688, 2001.

[9] Y. Cong and H. Jagadish. Schema summarization. In *VLDB*, 2006.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms: Second Edition*. MIT Press, 2002.

[11] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: automatic data extraction from data-intensive web sites. In *SIGMOD Conference*, page 624, 2002.

[12] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, 2001.

[13] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Web-Scale Information Extraction in KnowItAll. In *WWW*, pages 100–110, New York City, New York, 2004.

[14] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. Xtract: A system for extracting document type descriptors from xml documents. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 165–176, 2000.

[15] N. Kushmerick, D. Weld, and R. Doorenbos. Wrapper Induction for Information Extraction. In *Procs. of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, Nagoya, Aichi, Japan, 1997.

[16] I. Mansuri and S. Sarawagi. A system for integrating unstructured data into relational databases. In *Proc. of the 22nd IEEE Int'l Conference on Data Engineering (ICDE)*, 2006.

[17] Miller et al. Introduction to wordnet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–312, 1990.

[18] R. J. Miller and P. Andritsos. Schema discovery. *IEEE Data Eng. Bull.*, 26(3):40–45, 2003.

[19] I. Muslea, S. Minton, and C. Knoblock. Hierarchical Wrapper Induction for Semistructured Information Sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[20] L. Palopoli, D. Sacca, and D. Ursino. Semi-automatic semantic discovery of properties from database schemes. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS-98)*, pages 244–253, 1998.

[21] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.